Google Wave 入門

はじめに

2009 年 5 月 28 日、Google I/O 二日目開場前のモスコーン・コンベンション・センター 3 階ホール前は少し異様な雰囲気に包まれていました。前日のキーノートで参加者全員に Android ケータイが配られるというまさにサプライズが行われたカンファレンスの二日目です。「今日は何が発表されると思う？」「昨日 OpenSocial のセッションがほとんどなかったからその辺じゃない？」開場したらすぐに飛び込んでなるべく前のほうの席を陣取ろうと、ホールのシャッター前で座り込んでいる私たちの周りでは様々な噂が飛び交っています。共通しているのは皆一様に前日の興奮を引きずっていることと、今日は昨日以上の何かが起こるかもしれないという予感です。

そのキーノートで初めて発表されたのが Google Wave でした。正直に言いましょう。初めてその画面がスクリーンに映し出されたときの印象はあまりよいものではありませんでした。一言で言えば「デザインが少し垢抜けない GMail」。わざわざキーノートを使って発表する新サービスがこれか。そう感じたのは私たちだけではないようで、あの一瞬、周囲には軽い失望を感じたような空気が確かに流れました。しかし、ご存知の通り、その失望はそのあとに続いた圧倒的なデモの連続によって見事に覆されます。スピーカーたちの軽妙な掛け合いと共に一つまた一つとデモが公開されるにつれ会場のボルテージが上がっていき、最後には会場は総立ちで猛烈な拍手、熱狂と言ってよい状態でした。

キーノートがあれほどまでに盛り上がったのはなぜなのでしょう。確かにデモの数々は圧巻でした。しかし、非常に多岐に渡っている上にそれぞれの関連が明確とは言い難く、あの時点で「Google Wave とはなにか？」と問われたとして答えられる聴衆はいなかったはずです。にも関わらず捧げられたあの惜しみない賞賛、それは Google の「ネット上のコラボレーションを根本から再定義する」という無謀としかいいようのない目標設定と、それが少なくとも実現不可能ではないと示したことに対するものだったと私は思っています。

Google Wave が今後成功するかどうかはもちろん誰にも分かりません。しかし、少なくとも Google Wave によって「リアルタイム」という今後進むべき方向が示されたことは確かです。Google による挑戦は失敗に終わることもありえます。しかし現在のネット上でのコミュニケーションが最上のものとは言えない以上、それを根本から立てなおそうという試みは Google Wave が最後になるとは思えません。もしそのような試みが続けて行われるとしたら、Google Wave の同類はまた世界のどこかに現れるでしょう。

本書は Google Wave の入門を超えて、いつか来るリアルタイム Web の世界への入門の助けになることを願っています。

本書について

本書は大きく二つの部分から成ります。

第一部は二つの章を通じて Google Wave の概要をざっくりと理解してもらうことを目的としています。Google Wave について、名前以上のことを知らない人は一通り目を通しておくといいでしょう。ただし、あまり具体的な機能の話はないので、既にバリバリと Google Wave を利用している方には物足りないかもしれません。そう言った方は第一部

を飛ばして第二部に進んでいただいて構いません。

第二部は第一部とは反対にひたすら具体的な話が続きます。まずはじめに、第三章ではGoogle Wave の機能についてスクリーンショットを多用して紹介します。何を置いてもまずは Google Wave を使ってみたいという方は第一部と第三章に目を通してください。続く第四章では Google Wave の持つ API について説明します。ウェブ系の開発者の方で、Google Wave 上で動くツールやゲームを作ってみたい方はこの第四章を読みましょう。細かな API を全て紹介することはできませんでしたが、API を利用すればどういったことが可能なのかを一通り理解できるはずです。最後の第五章では Google Wave の仕組みとオープンソース Wave サーバーについて軽く説明しています。私の力不足によりまとまっているとは言いがたい内容ですが、Google Wave の内部に興味がある方は付録と合わせて調査の参考にしていただければと思います。

謝辞

高畠さん GClue の佐々木さんシーサー株式会社のみなさん

最後に、本書の完成を待つことなく逝った父、淳一へ。少しでも病床での励みになればと送った「本が完成したら送るけん読んでな」というメールに「楽しそうに仕事しよるな」と返事を貰ったのが結局最後のやりとりになりました。

目次

第 I 部

Google Wave とはなにか

第 1 章

Google Wave とはなにか

1.1 はじめに

1.1.1 Google Wave にできること

Google Wave とはそもそもなんなのでしょうか？ Google Wave の機能は膨大で、その人の目的や中心的に使う機能によって見方が全く変わってしまうため、これはなかなか難しい質問です。本来であれば一番最初に簡潔でキャッチーな Google Wave を表す一文を紹介したいところですが、残念ながら私自身単純な回答は持ち合わせていません。敢えて言えば、Google Wave は「Google が公開した拡張可能な多人数同時リアルタイムコミュニケーションのツールであり仕様」とでもなるでしょうか。これではどうにも分かりにくいですよね。ここは諦めて、まず Google 公式のヘルプドキュメントを見てみることにしましょう。

> Google Wave はオンラインでのコミュニケーション・コラボレーションのためのツールで、リアルタイムなやりとりをよりシームレスにしてくれます。リッチテキスト、画像、動画、地図などを使ったコミュニケーション・コラボレーションを一つの場所で行うことができます。
>
> wave は複数の参加者がいる会話の場です。そして参加者とはそのコンテンツについて議論したり協力して編集するために wave に追加された人たちです。参加者は wave の中でいつでもどこにでも内容に関する反応を書き込むことができ、またコンテンツ自体を書き換えたりさらに参加者を追加して wave を発展させることができます。wave はプレイバック機構によって以前の状態に巻き戻していつどのような編集が行われたかを確認できます。[*1]

初めに本節の目的からは少し外れますが、この引用文を理解するにあたって一つ注意しておくべき点があります。それは「"Google Wave"と"a wave"は別物である」と言うことです。もし上記を一読して混乱した方がいたらもう一度読み直していただきたいのですが、実は最初の段落と二つ目の段落は違うものについて説明しています。前者は Google の提供するサービスとしての Google Wave の機能について、後者はそのサービス内で wave と呼ばれている表示要素についての説明です。これらを混同すると理解の大きな妨

[*1] http://www.google.com/support/wave/bin/answer.py?hl=en&answer=162898 (2010/6/8 版を翻訳)

げになるため、本書でもそれら二つを意図的に使い分けています。注意して読み進めてください。なお他の資料では（大文字始まりの）Wave が Google Wave という意味で使われることもよくありますが、本書では混乱を避けるためサービスとしての Wave については「Google Wave」と、より明確な表記に統一することとしました。

それでは本題に戻りましょう。先のヘルプドキュメントから Google Wave は次のようなことが可能なツールであることが分かります。

- 利用者同士がコミュニケーション・コラボレーションできる
- リアルタイムにやりとりできる
- 各種マルチメディアを扱える
- 多人数が同時に参加できる
- 会話の内容は過去にさかのぼって確認できる

また直接は書かれていませんが、自明のこととして

- ブラウザ上で動作するツール

であることも大きなポイントでしょう。

1.1.2　Google Wave の 3 つの P

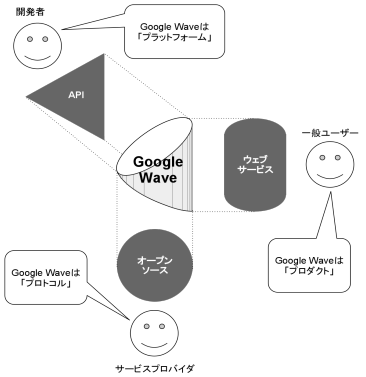
確かにこれら全てを実現するツールが Google Wave だとするととても便利そうです。ブラウザ上で動作させるにあたって非常に高度な技術が使用されていることも間違いないでしょう。とはいえ、これだけでは「ブラウザ上で動作する多機能チャット」にすぎません。Google による年に一度のテクノロジーカンファレンス、Google I/O のキーノートで大々的に発表され、居並ぶ Web 技術者から圧倒的な賞賛を受けた Google Wave が単なる「すごいチャット」程度のものということがあるでしょうか？もちろんそんなはずはありません。実は「すごいチャット」であることは Google Wave の一つの側面に過ぎないのです。Google I/O のキーノート冒頭の発言を引用してみましょう。

Stephanie Hannon 氏

> 「本日のプレゼンテーションは 3 つのパートからなります。初めにプロダクトのデモをお見せし、次に API について話します。みなさんは Wave を自分のサイトを拡張するために利用できますし、Wave 自身を拡張することもできます。三つ目に Wave がプロトコルでもあることを示します。我々はこれがコミュニケーションのオープンシステムになることを強く望んでいます。他の人々が wave サービスを構築して我々のサービスと相互運用してくれることが私たちの望みなのです。そうすればユーザーは自由に自分たちのプロバイダを選べるでしょう。我々はこれらを 3 つの P と呼んでいます。プロダクトとプラットフォームとプロトコルです」

Google Wave がすごいチャットであること、これは 3 つの P で言えばプロダクトとしての側面です。Wave はそれだけではなく、さらに二つの側面を持っていると Hannon 氏は言います。プラットフォームと言う側面を活用することで先に上げたような Google Wave の機能を自分で書いたプログラムから呼び出すことができ、さらにはプロトコルとして仕様（実装も！）が公開されていることから Google に一切頼ることなく先ほど上げ

たような機能を独自にユーザーに提供できるというのです。つまり Google Wave は一般ユーザーにとっては Google Maps のような Google が提供する Web サービスの一つに過ぎませんが、Web 系の開発者にとっては自分たちが開発したアプリケーションを動かすための基盤になり、さらにサービスプロバイダにとってはメールサーバーや Web サーバーと同等の、ユーザーに全く新しいサービスを提供する手段なのです。



1.1.3 E メールの再発明

どうでしょう、キーノートで聴衆が熱狂した理由がだんだんわかってきたのではないでしょうか？ Google I/O にはインフラ担当から Web サービスの企画・開発担当まで様々な Web 系技術者が集まっていましたが、Google Wave はその全ての層の技術者に等しく衝撃を与えられるサービスだったのです。それにしても Google はなぜこのようなサービスを開発し、しかもその仕様を残さず公開しようと思ったのでしょうか。その答えを知るためにキーノートのもう一人のトーカー、Rasmussen 氏の発言をみてみましょう。

Lars Rasmussen 氏

> 「それで、コミュニケーションについてなんだが、これまでのところ E メールがインターネットで僕らがコミュニケーションする一番ポピュラーな手段ということになってる。これは驚くべきことだ。だって E メールは 40 年以上も前に発明されたものだからね。インターネットよりも、ウェブよりも前だ。E メールは次のようなものの知識がない状態で開発されてる。SMS や IM、ブログ、Wiki、BBS、ディスカッショングループ、メディア共有サイト、共同編集ツール、要するに今ではみんなが当たり前と思ってるいろんな種類のコミュニケーションを知らないまま作られた。加えてコンピューターやネットワークもこの 40 年の間で劇的に進化してる。なので、一番初めに、もう 2 年以上前だけど、僕らは自問したんだ。もし今 E メールが発明されたとしたら、一体どういうものになるだろうか、とね。もちろんここにはたくさんの技術者がいてその問いに対する回答も星の数ほどあることはわかってる。今日みんなに披露する Google Wave は、その問いへの僕らなりの回答だ」

「Google Wave とはなにか」という問いに答えるため、これまで長々と文章を書き連ねてきましたが、どうやらこれがもっとも本質的な答えになりそうです。Google Wave とは「Google が再発明した現代版 E メール」だったのです。そうは言っても Google Wave は E メールと言うよりはむしろチャットではないかと思うかもしれません。しかし Google

が目指したのは E メールを改良すること**ではない**ということに注意して下さい。そうではなく逆に E メールのことは一度忘れてしまい、当時 E メールが生み出された目的、すなわちコンピューターのユーザー同士のコミュニケーションを円滑に行うという目的に一度立ち返り、相応しいツールをゼロから考え直した結果が Google Wave なのです。

それではこれから「Google Wave は現代版 E メールだ」という視点に立って、Google Wave と E メールを簡単に比較してみます。ただし細かい機能ベースの比較については後の節で行うこととして、ここで行うのはもう少し一般的な比較です。まず 2 つの大きな類似点としてその言葉の多義性があるでしょう。「E メール」という言葉は Outlook、Thunderbird などのメールクライアントを指すこともあれば、やり取りされる一通のメッセージを指す場合もあり、さらにはそのメッセージを交換するためのシステムそのものを呼ぶ場合もあります。同様に「Wave」という単語も Google Wave というプロダクトを指すこともあれば、サービス内での会話の単位として用いられることもあり、またその会話をリアルタイムに同期し続けるためのシステムを指すこともあります。この多義性について Hannon 氏は「3 つの P」と表現しましたが、同じ言い方をするなら E メールはプロダクトとプロトコル「2 つの P」からなるとも言えます。名前として「メール」と同様に非常に一般的な用語である「ウェーブ」と言う単語を選び、さらにその意味を一つに限定せずいろいろの用途に使っているのは、E メールからの影響を受けて意図的にやったことだと私は考えています。

さらに「現代版 E メール」という視点に立つと、Google がこれから Google Wave をどのように広めていきたいと考えているかもよく理解できます。E メールがこれほど使われている理由、それはツールとして非常に汎用性が高く便利だったと言うこともありますが、なによりも仕様が公開されていてオープンソースライセンスの元で公開されているサーバーソフトウェアがあるため、誰でもメールサーバーを公開して E メールプロバイダになれるという点が大きいでしょう。E メールを置き換えると言う野望を掲げる Wave ですから、その点はもちろんそのまま踏襲し、Wave の実装も時期は未定ですがオープンソースライセンスで公開される予定です。そうすればドメインを持つ人や企業が自由に Wave サーバーを立てることができるようになりますので、E メールのように会社ごとに社員が利用できる Wave サーバーを持つようになったり、Yahoo! Wave のようなものが登場してユーザーが自由に自分が利用するプロバイダを利用できるようになる可能性もあります。

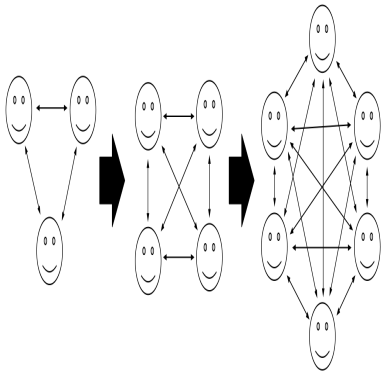
加えて、E メールとは違い Wave はクライアント側でも複雑な処理が必要ですので、クライアントの参照実装も公開される予定です。Google Wave の現在の UI についてはあまり洗練されていないという評価を受けることが多いように思います。私も正直同感ですが、これは将来ソースを公開することを見越してその読みやすさを優先し、現時点ではデザインや動作の過度な修飾を避けているからなのかもしれません。

1.1.4　Google Wave の新しさ

さて、E メールと Google Wave の全体的な類似点は分りました。それでは反対に両者が最も異なる点はなんなのでしょう。Google Wave の本質的な新しさはどこにあるのか、と言いかえてもいいかもしれません。もちろん立場によってさまざまな考えがあるでしょうが、私はそれを「ネット上でのコミュニケーションを実体のあるものとして定義したこと」だと思っています。E メールとどう違うんだ、と思われる方もいるかもしれませんが、E メールが表しているものはコミュニケーションではなく「メッセージ」です。E メール上でのコミュニケーションは「やりとりされた一連のメッセージを人間が恣意的に解釈した結果現れるもの」であって、E メールシステム内に存在するものではありません。コミュニケーションがシステムで定義されているものではないために、やりとりの過程で CC や BCC が追加されたり抜け落ちてしまったり引用が部分的であったりというシステム外の要因で、最終的に各人の結論の解釈が全く異なるものになってしまうことも珍しい話ではないでしょう。

もちろんそれでもそれなりにうまくいっていると考えることはできますし、なにも全く新しいオブジェクトを作り出さなくても関連するメッセージを集約するためのオブジェクトを追加する形でも問題は解決できるという考えもあるでしょう。実際、いくつかのツール・サービス[*2]ではそのような方法で問題を解決しようとしています。しかしコミュニケーションを個々のメッセージの集約と捉えてしまうと参加者が増えるに連れて必然的にその複雑さが増してしまうことが大きな問題です。たとえ 10 人ほどの人数でも各自がバラバラに関係者を選んでメールをすればその組み合わせの数は膨大ですし、常に全員を CC に含めることにしたとしてもそれぞれが引用で返事を書いていたのではあっという間に誰も議論の全体像がわからなくなってしまうでしょう。

[*2] 第 1.4.3 節（p.27）参照



Google Wave を用いたコミュニケーションではこのような問題は起こりません。なぜなら Google Wave においてコミュニケーションは人間が解釈するものではなく、実際にそこにあるもの（Wave）だからです。参加者が途中どれだけ変化しようともある話題に対する Wave は常にひとつだけで混乱のしようがないのです。

Wave をコミュニケーションの実体と捉えることで、いくつかの Wave の機能の裏にある考えも明らかになります。例えば Google Wave には会話を順番に再生するプレイバックと言う機能があります。これは参加者が文章の好きな場所を編集できる Google Wave では必須の機能で、必要に迫られて作られたものではありますが、同時に会話のコンテキストを表現しているとも言えます。同じ表現の発言であっても、だれがどのタイミングで言ったかによってその意味は変わってきます。E メールでは把握が難しかった会話の空気を再現できるようにしたのが Google Wave のプレイバック機能なのではないでしょうか。

また、Google Wave において Wave 作成者（New Wave した人）は自分自身が他の参加者から Remove されることがないということの他にはほとんど特別な権限を持ちません。作成者ができることは他の参加者ができることとほぼ同じです。ロボットやガジェットの追加・削除、参加者の追加、Wave 内のすべての文章の編集・削除などは Wave 参加者全てが実行可能です。ここに不満を持つ人も多いとは思いますが、Wave がコミュニケーションの実体だと考えるとある程度理解できます。例えばいま私があなたに話しかけて会話を開始したとして、その会話は果たして私のものでしょうか？もちろんそんなことはありません。敢えて誰のものかといえば会話をしている「あなたと私」のものでしょう。あなたはさらに第三者をその会話に誘うこともでき、そうすると会話は「あなたと私と彼」のものになります。私が会話に飽きてその場を離れたとしても、その会話はあなたと彼の間でそのまま続けられます。一般的にコミュニケーションは開始した人が何か特別な権限を持つようなものではないのです。このことを愚直に実装しているのが現在の Google Wave の権限モデルなのでしょう。

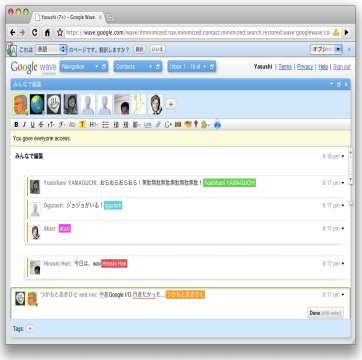
1.2　Google Wave でできること

Google Wave とは何か、その理解を難しくしている原因に「機能が豊富すぎること」があるのは事実です。「とてもいろいろなことができて、すごいのはわかる。でもこれがなんなのか、何に使うツールなのかが分からない」というのが初めて Google Wave の説明を受けた人の正直な感想ではないでしょうか。私自身、Google I/O のキーノートを見て

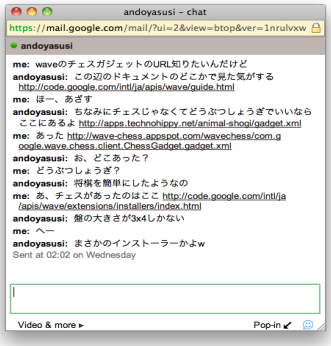
感じたことがまさにそうでした。メールに似たメッセージ送受信・IM のようなリアルタイムコミュニケーション・複数人同時チャット・ドラッグ&ドロップでの画像アップロード・画像スライドショー・外部ブログへの組み込みと双方向の編集・外部 SNS（Orkut）との同様な連携・Wiki のような共同編集・編集履歴の再生・多言語同時表示・リアルタイム検索・API を使った機能拡張・数独ガジェット・チェスガジェット・投票ロボット・Twitter との連携・code.google.com との連携・コンソールで動作する独自クライアント・入力のリアルタイム翻訳、などが Google I/O でのデモの例ですが、正直ここに書き出したものを見ても、これがたった一つのサービスのデモだとは信じられないくらいです。

そして事実、Google Wave の新しさはそれら個々の機能ではありません。Google Wave の（プロダクトとしての）新しさ、それは現在ウェブ上に存在する様々なツールの機能を「コミュニケーション」という視点で捉え直し、一つのツールとしてまとめ上げたことなのです。Goolge I/O でなされた多種多様なデモも、それぞれの機能を見せたかったわけではなく、Wave というコミュニケーションをモデル化したオブジェクトの可能性を参加者に感じてもらうことが本来の目的だったとすれば納得できます。本節では Google Wave のツールとしての機能的な特徴をいくつか紹介しますが、それらの多くは一つのモデルを複数の側面から見たものに過ぎないということを意識して読み進めて頂ければと思います。

1.2.1 リアルタイムコミュニケーション



Google Wave の特徴を上げろと言われたらほとんどの人がこの点を上げるのではないでしょうか。Google Wave のデモでユーザーが一文字打つたびに全参加者にその入力が共有されるということに驚いた人も多いでしょう。この特徴は現在のほとんどのチャットツールへの不満から生まれたそうです。Rasmussen 氏は「チャットをしている時間の半分は相手の入力を待っている時間だ」といいました。自分が何かを考えてメッセージを入力している間、チャットの相手はなにもできず返答が届くのを待っているだけです。自分のメッセージを送ると、今度は相手が何かを考えてメッセージを入力している間、自分が何もできずただ待つだけになります。相手の入力が遅いとここで待ちきれず、さらに次の話題を振ってしまい一つのチャットで複数の話題が交差してしまうのもよくあることでしょう。

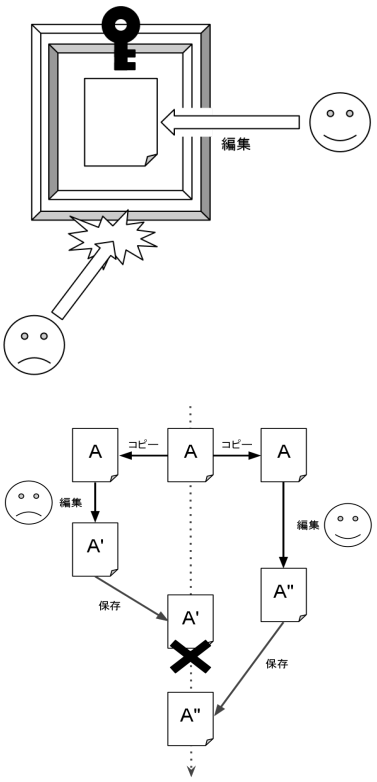


Google Wave を使えば、相手がタイプする一部始終を相手は見ることができるので、上記のような状況を避けられます。その結果、ただ待っているという時間がなくなり、場合によっては相手が打ち終わる前に返信も入力できるので、これまでのチャットより圧倒的にコミュニケーションがスムーズになります。また仮に相手が打ち終わるのを待つとしても、一気に入力した部分、迷いながら入力した部分、一度書きかけてから表現を改めた部分など、最終的な文章だけでは読み取ることが不可能な情報を理解して返事ができるのは大きな利点です。

さて、Google Wave のリアルタイムな入力の共有はなにも文字に限った話ではありません。画像を貼りつけると即座に他の参加者にも反映されますし、その画像のキャプションを変更しても同様です。検索結果までもリアルタイムに変更され、検索結果に出ていた Wave が編集され検索条件から外れれば即座に結果から消え、また逆に新しく検索条件に当てはまるよう Wave が編集されると編集結果にすぐに追加されます。さらに驚いたことには、Google Wave には「ガジェット」という外部の開発者が作成した特殊な機能を持つ表示要素を貼り付けることができますが、正しく開発されていればこのガジェット内での変更も参加者間でリアルタイムに同期されます。ガジェットについては第 4.2 節（p.92）で詳しく紹介しますのでここで詳細には立ち入りませんが、Google Wave ではリアルタイム性があらゆる面に渡って徹底されていると言ういい例でしょう。

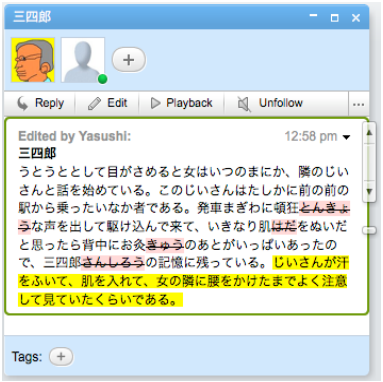
1.2.2　多人数での共同編集

Google Wave のリアルタイム性に継ぐ特徴は複数人で一つのドキュメントを編集できることでしょう。ドキュメントの共同編集はなかなか面倒な問題で、例えば共有の領域に置いたドキュメントを複数で編集する方式だと、通常はドキュメントを編集できるのはひとりだけになってしまい、ある人が編集している間は他の人はドキュメントを編集することができません。それを嫌って一旦ドキュメントを自分の領域にコピーして、それを編集した後で共有のドキュメントを置き換えるようなことも管理の行き届いていない組織ではまま起こりますが、それでは編集権の取り合いが発生しないかわりに以前の変更が上書きされ完全に削除されてしまうという危険性があります。進んだ組織であればコンカレントなバージョン管理システムを利用するでしょうが、それでも衝突を完全に避けることはできず、衝突が発生してしまうとその解消は手間のかかる作業です。



しかし Google Wave であれば何人で編集したとしても即座にその編集内容が全員で共有されるため、このような問題は発生しません。またそれによって利用シーン自体も一度に一人しか編集できない形態の共同編集とは大きく異なるものになり得ます。つまりこれまでのようなツールでは共同編集と言っても、一人がメインの著者になりそれ以外が内容チェックや校正をするか、または章ごとに担当を分け最終的に一つにまとめるような書き方しかできませんでしたが、Google Wave を使えば文字通り全員で一つの文章を仕上げることが可能になります。例えば会議などで Wave の画面をプロジェクタでスクリーンに投影し、発表資料として使いながら全員で直接編集、会議が終わったときには発表資料がそのまま議事録に変わっているといった使い方は Wave 以外のツールではまずできません。その上で、もちろん一人がメインの著者になり一通り終わったところでその Wave にレビュアーを追加して完成させるというこれまで通りの共同編集のやり方も Google Wave では可能です。

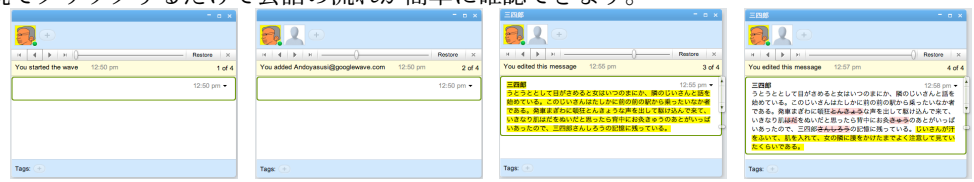
また、共同で編集するからといって常に全員が一緒に編集することができるとは限りません。共同作業中にいったん席を外して、時間をおいて再度 Google Wave にログインすることもあるでしょう。そうした場合であっても心配はいりません。Google Wave はログアウトした時点の Wave の内容を覚えていて、再度その Wave を開いたときに前回から変わった部分があれば文字の背景色を変えて知らせてくれることで、キャッチアップを助けてくれますし、次の項のようにプレイバックを利用することもできます。



1.2.3　プレイバック

Wave の参加者はいつでも誰でも追加することができますが、遅れて Wave に参加してきた人にとって、表示されている内容だけを頼りにそこで行われている会話を理解することは容易ではありません。Wave は上から下に会話が進むだけではなく、インラインで返信することもできるためドキュメント全体がツリー状の構造を持っています。しかも Wave 参加者は誰の発言でも自由に書き換えたり削除することができます。つまりどういう過程を経て現在の状態になっているのかが分からないと Wave の内容を本当に理解することはできないのです。そこで Google Wave にはプレイバックという機能が用意されました。

先程の共同編集の項で最後にログインしたときから変更があった部分に色がつくという説明をしました。プレイバックはこれをさらに推し進めた機能だと思ってください。機能の名前から予想がつくかもしれませんが、プレイバック機能を使うと Wave の編集履歴を遡ってその時々の内容を確認できます。前のバージョンから変更された部分は自動的にフォーカスがあたり強調表示されるので、プレイバックモードに入り「次へ」ボタンを連続でクリックするだけで会話の流れが簡単に確認できます。



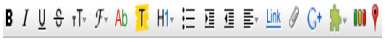
なお、これは余談ですが、実はこのプレイバックを可能にしている技術はリアルタイムの共同編集を可能にしている技術と同じです。一見全く異なって見える機能が技術的には根を同じくしていると言うところに Wave というモデルの可能性を感じてみてください。この技術の詳細については第 6.2 節（p.266）で説明しています。

1.2.4　リッチテキスト編集

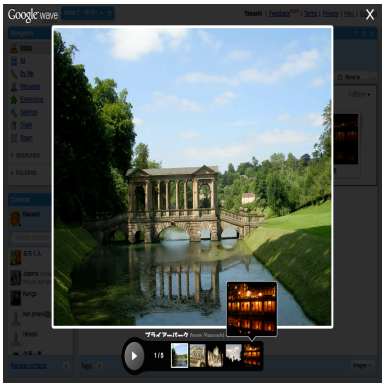
誰でも好きなところを編集できるドキュメントと聞くと Wiki[*3]のようだと思った人がいるかも知れません。確かに Wave の「誰でもどこでも編集可能」という思想は Wiki の

[*3] http://c2.com/cgi/wiki?WikiWikiWeb

影響も大きいと思われますが、Google Wave での編集は Wiki よりも少しだけ今風になっており、文章の編集にリッチテキストエディタが利用できます。もちろん参加者間でのリアルタイムな状態の共有はこれらの設定に対しても有効で、フォントサイズや色・インデントを変更すると即座に他の参加者に反映されます。



さらに Google Wave はマルチメディアの扱いにも長けており、Youtube[*4]に登録されている動画 URL を書けば自動的にその場に動画を挿入してくれますし、画像に関しては現在のところ[*5]Gears[*6]の助けが必要ですが、Wave 上に画像をドラッグ＆ドロップするだけで簡単に Wave に貼り付けられます。張り付けた画像はスライドショーと言う形式で画面いっぱいに表示することができ、進むボタンで次の画像を表示したり、再生ボタンで自動的に順番に画像を表示させることも可能です。Google I/O ではこのスライドショー機能を使って Wave に関するプレゼンテーションを Wave 自身で行っていました。



1.2.5 外部サイトとの連携

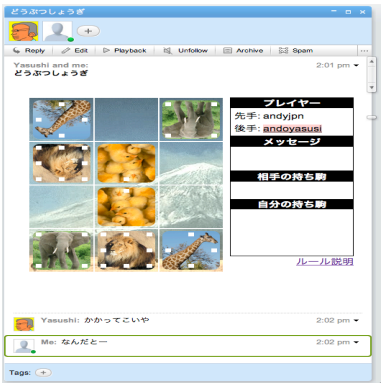
Google Wave はエンベッド API と呼ばれる JavaScript のオブジェクトと関数群を使い、Google Wave とは直接関係のない外部のサイトに簡単に組み込むことができます。この外部サイトとの連携機能については Google I/O では大きく取り上げられましたが、それ以降あまり目立った動きはありません。また、API としては単純で、できることも限定的であることから今のところ開発者の注目度もそれほど高くはないようです。しかし今後の Google Wave の発展の要と言う意味では、もしかするとこの外部サイトとの連携機能こそが、この節で取り上げる全ての特徴の中で最も重要なものかもしれないと私は考えています。

[*4] http://www.youtube.com
[*5] 将来的には HTML5 の機能で置き換えられる予定です
[*6] http://gears.google.com

エンベッド API を利用してできることは、要するに外部サイトへ指定した Wave を埋め込み、その Wave を参照したり編集することです。それだけのこと、と言ってしまえば確かにそれだけのことかもしれません。しかしそこに組み込まれているのはリアルタイムに多人数で共同編集できる正真正銘の Wave なのです。例えばブログのコメント欄として Wave を組み込めば、そのサイトを閲覧しているユーザーたちがそのブログのコメント欄でコンテンツについてリアルタイムにやりとりできるようになります。複数のブログを持っている人が全てのコメント欄を Wave にすれば、Google Wave にコメントを集約して Google Wave から全てのブログのコメントに対してリアルタイムに返答できます。さらに、全く無関係な二つのサイトに同じ Wave を組み込めば、それらのサイト間でもリアルタイムにそれぞれの読者同士がやりとりできるようになるでしょう。

つまりエンベッド API が多くのサイトで使われるようになれば、限定的ではありますが、Google Wave のリアルタイム性が Google Wave の中だけに留まらずウェブ全体に広がることになるのです。Google Wave はサーバー・クライアント共にいずれオープンソースになりますが、それらが実際に広く使われるようになるには長い時間が必要でしょう。しかしこのエンベッド API であれば普及を待つ必要はありません。簡単な JavaScript を貼りつけるだけで既に存在するあらゆるサイトに部分的なリアルタイム性を持たせることができるのです。

1.2.6　機能拡張



前節ではエンベッド API を説明をしました。Google Wave にはあと 2 つ API があります。それがガジェット API とロボット API です。こちらは Google Wave 自体の機能を拡張します。ガジェット API では特殊な機能を持つコンポーネントを Wave に追加で

き、ロボット API では Wave の参加者として振る舞うプログラムを作成できます。開発者はこれらを利用すれば、Google Wave をゲームプラットフォームや業務ツールに変えることさえ可能です。Google Wave は「進化した E メール」といえるという話を先程書きましたが、この API を利用した機能拡張については E メールには対応する機能が全く存在しません。それだけにこの API を利用した機能拡張は Google Wave の大きな特徴といっていいでしょう。

API に関して Rasmussen 氏は「Google Maps の経験からサービス普及に API がいかに重要かを学んだので、Google Wave でも API を提供することにした」と言っています。実は Google Wave の開発に中心的な役割を果たした Rasmussen 兄弟[*7]は Google Wave に携わる前は Google Maps 開発チームの中心人物でした。Google Maps は API を提供したことで多くのサイトに組み込まれ、Google Maps 自身にはない機能を持った地図サービスも多数現れました。そしてそれらのサービスが Google Maps 自体の知名度を高め、Google Maps への流入を高めると同時に、さらに API を利用したサービスを増やすという好循環を生みました。Google Wave はそんな Google Maps 以上に汎用性の高いサービスですから、これらの API を利用してどのようなサービスが今後生まれるかを考えると非常に楽しみです。

1.2.7 モバイル

最後に、Google Wave は Android や iPhone などのモバイルガジェットでも動作します。Google Wave の UI は Google Web Toolkit（GWT）を利用してほぼウェブ標準技術だけ[*8]を使用して作られているため、ウェブ標準に対応したブラウザが動作するガジェットであれば自動的に対応が完了していることになるのです。Google Wave のリアルタイム性は常に持ち歩くことができるモバイルガジェットでこそ、その力を発揮します。残念ながら今のモバイル版 Google Wave は（おそらく）GWT で生成されたものそのままで、安定性やパフォーマンスなどの点で満足できる出来とは言い難いのも事実です。しかしこれは単に開発リソースの問題に過ぎません。モバイル版の重要性は誰の目にも明らかですし、PC 版が安定すればモバイル版の改善にも本気で取り組んでくれることでしょう。もちろんクライアントサーバー間のプロトコルが公開されれば、Google の対応を待つま

[*7] Jens & Lars Rasmussen
[*8] ドラッグ&ドロップには Gears が用いられていますが、HTML5 のドラッグ&ドロップに関する規格が固まればそちらが利用される予定です

でもなく Google 以外の会社や個人がモバイルクライアントを作成することも可能です。今のところは、モバイルで動かすことが可能だと示してくれているだけですが、Google Wave がまだまだ開発中のサービスであることを考えれば十分ではないでしょうか。

1.3　既存のコミュニケーションツールとの比較

先に Rasmussen 氏の「Google Wave はさまざまなウェブ上のコミュニケーションツールの知見を元に再発明された E メールだ」という発言を紹介しました。この節では Google Wave の理解をさらに深めるため、E メールを含むさまざまなコミュニケーションツールとの比較を通して、それらの知見が Google Wave にどのように活かされているのか、つまりこれまでのツールの何を継承し、何を捨て、何を新しく付け加えたのかを探ってみます。

1.3.1　E メール

E メールと Google Wave との比較はこれまでに何度か書いてきましたが、ここでもう一度簡単にまとめておきます。Google Wave が E メールから引き継いだ点で目立っているのは

- アドレス帳・Inbox・詳細メッセージといった全体的な UI の構成
- 「ユーザー ID@ドメイン名」という構成のアドレス
- プロトコル・サーバー参照実装の公開

以上の 3 点でしょうか。前者の 2 点は E メールを置き換えるという目的からユーザーに違和感を抱かせないようにするために意図的に似せたものと思われます。E メールの影響として最も重要なのは 3 点目でしょう。初めからプロトコルや参照実装を公開するという決定は、同様にプロトコルが公開された E メールがここまで広まったという事実がなければありえなかったのではないでしょうか。これら 3 点以外にも E メールに似ている点を見つけることは可能かもしれませんが、先に書いたように Google Wave が E メールから影響を受けた最も大きな点は、機能でも実装でもなくツールとしての目的だということは忘れないでいただきたいと思います。

逆に E メールの反省のもとに Google Wave で改善された点は

- メッセージのやりとりでコミュニケーションを表すのではなく、コミュニケーションそのものを表す Wave というオブジェクトを導入した

ところでしょう。コミュニケーションを表すオブジェクトの導入を決定したことで、そのオブジェクトは「複数人で」「同時に」編集できることが求められ、Wave オブジェクトの基本的な特性が決定しました。さらにそういったオブジェクトを E メール同様に多くのサービスプロバイダで分散してホストするためにフェデレーションプロトコルが導入される必要性が生まれたと考えると、E メールの与えた影響は少し逆説的ですが、非常に大きいと言えるでしょう。

1.3.2 チャット

E メール以上に Google Wave に似ているツールといえばチャットサービスが挙げられます。似ている点は明らかでしょう。

- メッセージがリアルタイムに送受信される
- 複数人が同時にやりとりできる

これらはちょうど E メールには足りていなかった部分でもあります。よくチャットでは「ルーム」という単位で会話がまとめられますが、データモデルとしての wave を説明する場合にはこの「ルーム」を引き合いに出すのが最も簡単です。さらに、これは似ているという表現は適切ではありませんが、影響という面で非常に大きいのはプロトコルです。Wave がサービスプロバイダ間で使用するプロトコルは XMPP[*9] といいうチャット用のプロトコルを拡張したものです。そういう意味でも実装の面で最も大きな影響を受けているのはチャットだと言っていいでしょう。

逆に一般的なチャットと異なる部分は

- 一文字入力するごとに送信される
- 他の人の文章も編集できる

という点でしょう。前者は小さな違いに見えるかもしれませんが、先ほど説明したように、このことによって最終的な文章だけからは読み取れない情感を伝えることができるようになりました。後者については、この特徴のおかげで Wave は単なる「チャットルーム」以上のものでありえます。現在のチャットサービスは要するに表示方式を変えたターンアラウンドタイムの短い E メールに過ぎません。チャットルームと言う形式でコミュニケーションを定義するということは、コミュニケーションをメッセージのシーケンスと捉えていることに等しく、本質的には E メールと全く同じですから結局これまで書いてきたような E メールの持つ問題をほとんど解決できないのです。Google Wave では複数人リアルタイムコミュニケーションという特色はそのままに、メッセージの所有権をなくし、既存のメッセージを編集できるようにしたことで多くの問題を解決することに成功しました。

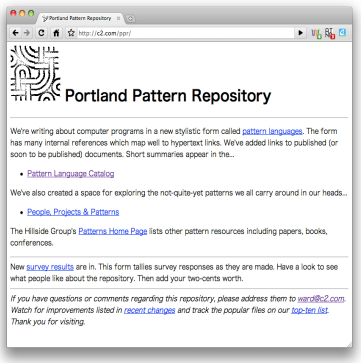
1.3.3 Wiki

Wiki はブラウザ上でドキュメントを共同編集できる非常にシンプルなツールです。残念ながら E メールやチャットほどには有名ではないため、少し説明が必要かもしれません。Wikipedia のウィキの項では次のように説明されています。

> ウィキ（Wiki）あるいはウィキウィキ（WikiWiki）とは、ウェブブラウザを利用して Web サーバ上のハイパーテキスト文書を書き換えるシステムの一種である。それらシステムに使われるソフトウェア自体や、システムを利用して作成された文書群全体を指してウィキと呼ぶこともある（「〜に関するウィキサイト」など。し

[*9] Extensible Messaging & Presence Protocol

ばしばウィキペディアが「ウィキ」または「Wiki」と略されるが、「ウィキ」は以上のようなシステムの一般名であるため、厳密には混同が生ずる)。... 中略... ウィキでは通常、誰でも、ネットワーク上のどこからでも、文書の書き換えができるようになっているので、共同作業で文書を作成するのに向いている。この特徴から、ウィキはコラボレーションツールやグループウェアであるとも評される。[*10]



Wiki はもともとはパタン・ランゲージを共同で記録するために Ward Cunningham 氏によって開発されたシステムで、そのオリジナルの C2 Wiki は未だに現役で動作しています。Wiki が何かということについては未だに明確な定義というものはありませんが、その C2 Wiki には Wiki Design Principles[*11] というページがあり、Cunningham 氏が初めての Wiki を設計した際の設計原則として 12 の項目が紹介されています。その中から WikiName[*12] や Wiki 記法[*13] に関する項目を除いたものが以下です。

- オープンであること: 不完全であったりうまく構成されていないと思えるページは、読んでいる人が正しいと思うよう自分で編集できる
- 自然発生的であること: サイトの構造やテキストの内容は変更が許されていて進化できる
- ユニバーサルであること: 編集と組織化のメカニズムは執筆のそれと等しいので、全ての執筆者は自動的に編集者であり組織者になる
- 観測可能であること: サイト内での行動は、他の全てのサイト訪問者によって観察されレビューされる

Wiki ではサイトの閲覧者誰もが自由にドキュメントを編集できますが、これを「Wave の参加者」と置き換えれば上記の原則のほぼ全てが Google Wave でも成り立っていることがわかるでしょう。Wiki のドキュメントは誰でも自由に編集できると言う点が非常に重要視されていて、その哲学についてはよく「間違いが起きないようにするのではなく、間違いを訂正しやくする」と表現されます。つまり、元々の Wiki はもし誰かが不適切な編集をしたとしても十分な数の参加者がいれば誰かが正しく訂正するだろう、といういわば性善説に基づいて管理されているのです。Google Wave の権限管理の緩さもけして手

[*10] http://ja.wikipedia.org/wiki/%E3%82%A6%E3%82%A3%E3%82%AD
[*11] http://c2.com/cgi/wiki?WikiDesignPrinciples
[*12] Wiki の特徴的な自動リンクのシステム。Wiki と Wave はページの管理方法が全く異なるので省略します
[*13] 非常に単純なマークアップ記法です。Wave はリッチテキストエディタを持つので省略します

を抜いているわけではなく、Wiki を参考に同様の信念で敢えてそのように作られているものだと私は思っています。

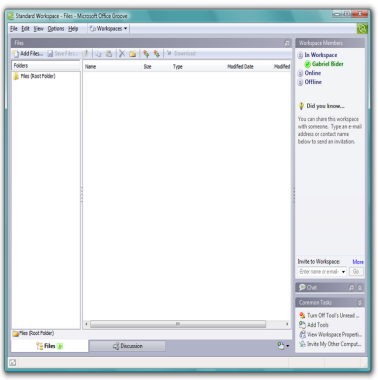
・・・

以上、Google Wave への影響が特に大きいと思われるコミュニケーションツールを 3 つ取り上げました。もちろん参考にしたツールはこれら以外にも SNS・メディア共有サイト・ディスカッショングループなど多くのものがあります。いずれにしても、これらとの比較でわかることは、Google Wave の機能を部分的に取り上げればほとんど全てが他のサービスによってすでに実現されているものだと言うことです。しかし、その事実は Google Wave の新しさを少しも損ねるものではありません。というのももともと Google Wave の新しさは個々の機能ではなく、それらの機能を破綻なく一つのツールにまとめ上げたことだからです。Google Wave の UI は一見しただけでは特に新しさを感じさせません。しかしこれだけの機能をもちながら UI として「普通」と感じさせることこそが Google Wave の凄さと言えるのではないでしょうか。

1.4 Wave と目的が近いツールとの比較

現在のコミュニケーションツールの乱立に不満を覚え、どうにかしたいと思っているのはもちろん Google だけではありません。今度は Google Wave と同じ問題に挑戦していると思われるツール・目指しているところが近いと思われるツールをいくつか取り上げて Google Wave と比較してみましょう。

1.4.1 Groove



Google Wave 発表当初からよく似ていると言われていたのが Microsoft Office スイートに含まれる Groove というコラボレーションツールです。元々はロータスノーツの開発者が立ち上げた会社の製品でしたが、2005 年に Microsoft に買収され、Office2007 から Office スイートの一員となりました。ワークスペースと呼ばれるコラボレーション用の領域を目的に応じて作成し、必要なメンバーを招待した上で、ファイル共有やチャット、掲示板などをコミュニケーションの目的や相手の接続状況に応じて使い分けるという基本的なコンセプトは、確かに Google Wave をビジネスに利用したとすればこう言った使い方になるだろうなという説得力があり、当初似ていると言われたのも納得ができます。

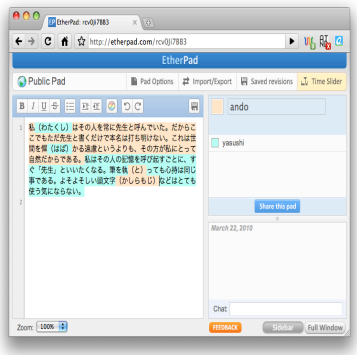
また Groove の P2P で動作するため中心となるサーバが不要な点、それぞれのメンバーのマシンにデータのコピーを持つので他のメンバーと接続されていなくても編集が可

能である点、それらのコピーの同期を取るレプリケーションが技術的な核と言われている点は、それぞれ Google Wave のサーバーサイドの、Federation プロトコルでサーバー間の同期をとり、Wave はそれぞれのサーバーがコピーを保持していて、それらを同期する Operation Transformation こそが技術的な核であるという特徴ときれいに一致しています。Groove は Wave サーバーとクライアントを一つにまとめ、P2P で繋いだものと言う見方もできるかもしれません。

ただし Groove はあくまでも Office スイートの一員で、ブラウザ上で動くわけはなくオープンソースでもないプロプライエタリな Windows アプリケーションに過ぎません。Groove の目的は組織内での共同作業を効率化することですが、Google Wave の目的はメールを代替しネット上でのコミュニケーションの質を変革することです。根本的に目的が異なるため両者は直接に衝突するようなものではなく、またそれらの相似も収斂進化の結果と捉えるのが妥当でしょう。

これは余談ですが、両者の比較については Bob MacNeal 氏が 2009 年 7 月に Wavesandbox 上で行った「Google Wave は Microsoft Groove を追い越すだろうか？ (Will Google Wave Overtake Microsoft Groove?)」というアンケートがあります。そもそもアンケートの場が Wavesandbox というところがすでに偏っていてどの程度信用できるものか疑問ではありますが、その結果は はい:73 いいえ:3 たぶん:16 で、圧倒的に Google Wave 優勢だったようです[*14]。

1.4.2　EtherPad



EtherPad は Google の元社員たちが集まって作った AppJet 社の提供する、ブラウザ上でリアルタイムにリッチテキストを共同編集できるサービスです。その見た目も機能も編集モードに入っている時の Wave とよく似ていて、URL を伝えるだけですぐに共同編集を始めることができたり参加者ごとに文字の背景色が色分けされ誰が編集した部分なのかがすぐにわかるという部分に関しては Google Wave よりも優れているかもしれません。また Time Slider という Google Wave でいうプレイバックと同等の機能もあり「共同で文書を編集する」というユースケースに特化したことで、用途によっては Google Wave よりもシンプルで使いやすいツールになっていると言えます。

しかし、あまりにも対象領域が重なりすぎたためか AppJet 社は 2009 年 12 月 4 日 Google に買収されました。AppJet 社の元メンバーは Google に出戻る形になり、今は

[*14] http://bobtuse.blogspot.com/2009/07/will-google-wave-overtake-microsoft.html

Wave チームと共に働いているそうです。EtherPad については 2010 年 3 月末をもってサービス終了をし、現在は Google Code[*15]でソースコードが公開されています。

1.4.3 Raindrop

Raindrop は Mozilla の考える、複雑化しすぎたネット上の数多くのコミュニケーション手段を集約する仕組みです。そういう意味で Raindrop と Google Wave の解決しようとしている問題は同じといっていいでしょう。ただし、その方法は両者で大きく異なります。Google は様々なコミュニケーション手段を含んだ全く新しいシステムを E メールの考え方を参考に生み出しました。Raindrop はそうではなく既存の Web 標準を使用して、様々なコミュニケーション手段をアグリゲートし、そこにインテリジェントな処理を加えた上でよりユーザーに適した形で提供するという手段をとります。つまり、Google Wave はネット上のコミュニケーション全てに関わるシステムを置き換えることを野望としましたが、Raindrop の場合はもう少し現実的にあくまでもコミュニケーションツールのハブになることを目指していると言えるでしょう。

実際には Raindrop はバックエンドに CouchDB を使用した小さな Web サーバーで、事前に E メールや Skype・Twitter などのアカウント、任意の RSS フィードを登録しておくと、その Web サーバーが提供するページ内でそれらの内容を同時に確認できます。もちろんそれらは通常の Web ページにすぎませんから、閲覧には自分が普段使用しているブラウザがそのまま利用でき、そこで表示されるものは取得した全メッセージの中から自分に関係が深いものを抽出したものになっています。さらにメッセージが twitter からのものであれば Raindrop 上から返信できますし、メッセージに Youtube などの URL が含まれていれば、その場で動画を再生することもできます。

また Raindrop の大きな特徴として、その拡張性の高さがあり、開発者は HTTP、JavaScript、CSS などの標準的な技術を使用して Raindrop の機能を拡張できます。対応するプロトコルも追加可能ですので、将来的には Wave プロトコルも取り込み、全てを飲み込んだコミュニケーションハブになる可能性もあります。

・・・

さて、いくつかの視点から Google Wave の比較対象になると思われるサービスを 3 つほど見てきました。これらのサービスとの比較により、Google Wave の他にはない特徴が浮かび上がってきます。まず何よりもオープンプロトコルというアプローチが Google

[*15] http://code.google.com/p/etherpad/

Wave と同じネット上でのコミュニケーション手段が発散しているという問題に取り組む他のサービスと比較しても、志の高さで群を抜いていると言えるでしょう。確かに短期的には Raindrop や Opera Unite[*16]のように様々なコミュニケーション手段を集約するというアプローチの方が素直で誰にでもわかりやすいものかもしれません。しかしその方法ではコミュニケーション手段が増大し続けるのを止めることはできず、どうしても場当たり的な対応にならざるをえません。Google Wave のようにすべてが可能なプロトコルを定義し、その参照実装を公開するという方法であれば、少なくともプロトコルのレベルではコミュニケーション手段を統一でき、問題を根本から解決できる可能性があります。

また、やはり Google Wave のほとんどキーストローク単位と言っていい同期は、他のサービスにはない Google Wave ならではの新しいユーザー体験を提供しています。チャットとして利用しているときは相手の打ち終わりを待つことなく返事を書き始められ、多人数で一つの文書を編集していればそのときどこが参加者の注目を集めているかをカーソル位置や編集の様子から一望でき、さらにそれらを組み合わせて利用できることによるターンアラウンドタイムの短縮は、手紙から E メールに置き換わったのと同種の変化をネット上でのコミュニケーションにもたらすかもしれません。

1.5　Google プロダクト内での位置づけ

Google は Google Wave 以外にも様々なウェブサービスを提供していますし、深くコミットしているウェブ標準やオープンソースも数多くあります。Google Wave とそれらの関係はどのようなものなのでしょうか。

1.5.1　機能的に Google Wave に含まれるサービス

- GMail
- GTalk
- Google Docs

これらは第 1.3 節で比較した E メール・チャット・Wiki にそれぞれ対応する Google のサービスです。特に GMail とは画面構成も非常によく似ています。これらのサービスは機能的には完全に Google Wave に含まれていると言っていいでしょう。将来的にこれらのサービスと Google Wave がどのように住み分けられるかはまだ分かりませんが、Google Wave の成功の度合いによって次のような可能性が考えられます。

1. Google Wave が非常な成功を収める。GMail、GTalk、Google Docs はすべて Google Wave に統合され、廃止される。
2. Google Wave がそこそこの成功を収める。GMail、GTalk、Google Docs はバックグランドで Wave プロトコルを利用するようになる。
3. Google Wave がそれほど普及しない。GMail、GTalk、Google Docs とは今まで通り直接の関係をもたない。

[*16] http://unite.opera.com

1.5.2 Google Wave から利用されるサービス

- YouTube
- Google Maps
- Google AppEngine
- Gears

Google Wave から利用される、といってもその利用のされ方はいくつかあります。前者の 2 サービスは Google Wave に貼り付けることができます。両者とも Google のサービスであるだけはあり、YouTube は動画 URL を本文中に書くだけで動画を挿入できますし、Google Maps は専用の挿入ボタンが編集ツールバーに用意されています。Google Wave 開発者が元 Google Maps チームだったこともあり、Google Maps ガジェットは Google Wave のリアルタイム性を非常にうまく生かしたものになっています。

Google AppEngine は Google Wave の拡張の一つ、ロボットをホストするのに使用されます。と言っても、当初は GAE 上でしかロボットを動かすことができませんでしたが、現在はロボットが利用するプロトコルは公開されているため、自由なサーバー上で好きな言語を使用してロボットを動作させることができます。ただし、今はまだ公式のライブラリは GAE 上で利用可能な Python 版と Java 版しかありませんし、Wave ロボットは対応するイベント次第でかなりの頻度で発生するリクエストを処理しなければいけません。今しばらくは GAE で Wave ロボットを動かすのが得策でしょう。

Gears は先に書いた通り、画像をドラッグ&ドロップで Wave に貼りつけるために使用されています。ただし、この依存は HTML5 のドラッグ&ドロップ API が普及するまでの一時的なものです。HTML5 を推進するために Gears 自体すでに開発終了が宣言されていいます。[*17]

1.5.3 Google Wave が準拠する新しい標準

- OpenSocial
- HTML5

OpenSocial と HTML5 はいずれも Google が先頭に立って推進している規格で、Google Wave ももちろん準拠します。OpenSocial は SNS のもつプロフィール情報やソーシャルグラフを利用して SNS を拡張するための仕様[*18]で、Google Wave のガジェット API は現在この OpenSocial に部分的に準拠したものになっています。もちろん将来的には完全準拠する予定です。

HTML5 は Google Wave のリッチな UI を実現する基盤になっています。HTML5 がなければ Google Wave は非常に限られたブラウザでしか動作しないウェブアプリケーションになっていた可能性があります。今後も HTML5 の仕様が固まるのに合わせて Google Wave の HTML5 対応もさらに進むでしょう。特に WebSocket API が多くのブラウザで実装されるようになると Google Wave のパフォーマンスやレスポンスが大幅に

[*17] メンテナンスは継続されます
[*18] 要するに mixi アプリや goo ホームガジェットを作るための仕様

改善されることが期待されます。

1.5.4　Google Wave を利用する環境

- Chrome
- Android

Google Wave は HTML5 対応したブラウザであれば動作しますが、Chrome は HTML5 の準拠率が高く JavaScript エンジンの V8 も高速なので Google Wave を利用するのにたいへん適しています。また PC に限らず Android 上のブラウザでも問題なく動作します。

1.5.5　Google Wave の開発ツール

- Google Wab Toolkit（GWT）

Google Wave の開発には GWT が全面的に採用されています。Rasmussen 氏は、GWT がクロスブラウザ対応をはじめとする JavaScript の本質的ではない部分を受け持ってくれたから Google Wave のような複雑なサービスを作ることができたという趣旨の発言をしています。

1.5.6　Google Wave に類似したサービス

- Google Buzz

2010 年 2 月 10 日に Google は Google Buzz というサービスをリリースしました。Google Buzz は最初から GMail に組み込まれていて、Twitter に似たインターフェースで様々なコミュニケーションサービスの情報を集約し、リアルタイムに情報が更新されていくことから、公開直後から Google Wave と比較されることの多いサービスでした。

しかしそれらのキーワードは似ているように見えても、詳細に検討すると両者の見ているものは全く異なります。Google Buzz はソーシャルウェブの実験場という意味合いの強いサービスですが、Google Wave はソーシャルな要素よりもリアルタイムウェブという未来を見据えています。また、Google Buzz は一つのウェブアプリケーションですが、Google Wave は Wave という仕様を満たした実装の一つに過ぎず、本体なのはオープンな仕様の方です。Twitter がどれほど流行しても E メールがなくなることはないように、Google Buzz と Google Wave が潰し合うようなこともおそらくないでしょう。

・・・

Google Wave プロジェクトが始まったのは 3 年ほど前ですが、実際にはそれ以前から公開されていた GMail、GTalk、Google Docs、Google Maps といったサービスの開発・運用経験があってこその Google Wave です。さらに Google Wave を世に出すには、HTML5、OpenSocial などの基礎になる規格のある程度以上の普及が不可欠でした。そういった意味で Google Wave は現在の Google のまさに集大成といっていいサービスでしょう。

第 2 章

Google Wave の利用例

これまで Google Wave とは何かを理解するために、設計思想の検討、他のツールとの比較などを行いました。次に Google Wave の具体的な利用シーンを考えてみることで、さらに理解を深めましょう。なお、Google Wave の実際の機能や使い方についてはまだ説明していないため、説明が少しわかりにく部分もあるかもしれません。後で読みなおす場合も考えて具体的な手順に触れる場合もありますが、重要なのはどういう用途に利用できるか、という部分です。細かい部分が理解できなかったとしても気にせず読み進めてください。

2.1 議事録

Google Wave は Google Apps のサービスとして特定のグループのメンバーだけに公開して利用することもでき、会議の議事録を作成するのに最適です。議事録に必要な議題・日時・参加者などの基本情報は毎回同じですからテンプレートを用意して使いまわすようにします[*1]。Google Wave にはミーティング用テンプレートも用意されていますが、執筆時点では英語だけなので構成を参考にするだけで、グループに合わせて自作した方がいいでしょう。会議の内容ごとに適切なタグを設定しておくと後で同じテーマの会議の議事録を確認しやすくなります。

テンプレートを元にアジェンダを作成したら、会議の参加者を wave に追加します。Google Wave はガジェットを利用して画像や Google Docs のスプレットシートを貼り付

[*1] テンプレートを元に wave を開始するには、wave の「Copy message to new wave...」メニューを使います

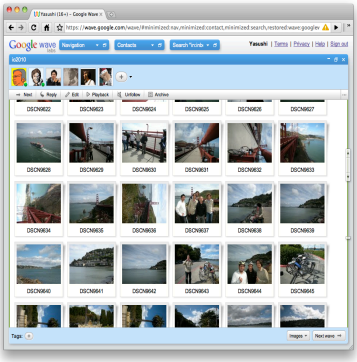
けることもできるので、この時点で参加者手持ちの資料を貼りつけておいてもらうのもいいかもしれません。

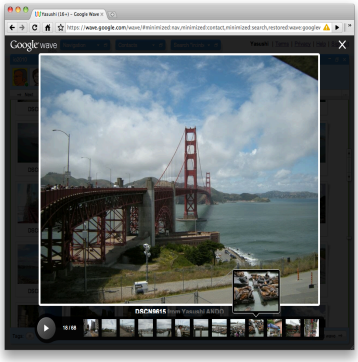
実際に会議が始まると議事録担当が主になって会議の進行に従って議論の内容を wave にまとめていくだけです。各参加者もノート PC で内容を確認しながら不備があれば気がついた人が直接その場で修正すればいいでしょう。会議中に提示された資料についてはできる限りその場で議事録 wave に貼りつけておきます。資料を貼り付けるための各種拡張は議事録担当の Google Wave に事前にインストールしておきましょう。wave から wave へのリンクについては検索ペインから wave 上に wave をドラッグ&ドロップするだけで済むので、資料自体を wave で用意してもらうようにすれば、よりスムーズに記録を進められます。

会議が終わるとその時点で議事録はほぼ完成しています。会議参加者以外で議事録を必要とする人がいるなら参加者に追加して完了です。紙での議事録の出力が求められるときには Ferry という拡張を使用して Google Docs に Export できますので、いったん Google Docs に書き出してから印刷するといいでしょう。

Google Wave を使用して議事録を作成する利点は、複数人で補い合いながら作成できることの他、遠隔地にいてもリアルタイムで wave を確認して内容を把握できることや、新しく追加されたメンバーが過去の会議の流れをプレイバックを使用して追えることによりキャッチアップの期間が短く済むことなど数多くあります。Google Wave を何に使っていいか分からないと感じている人は、まず議事録ツールとして使ってみてはどうでしょうか。

2.2 旅行の計画と記録





友人と旅行に行くときには計画の段階から一つ wave を立てておくと非常に便利です。まずはじめに wave にアンケートガジェットや Google Maps ガジェットを用意して参加者を誘えば、相談しながら旅程を決定できます。Google Wave 上から直接 Google 検索を実行して見つけたサイトのリンクを wave に貼り付けることもできるので、例えばホテルの連絡先なども簡単に wave に組み込めます。完成した旅程表を紙でも持っておきたいのであれば、先の節で紹介した Ferry という拡張を利用して Google Docs に変換してから印刷しましょう。

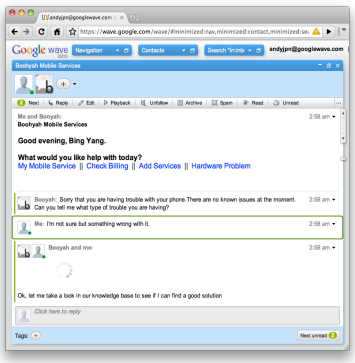
旅行が始まると流石に wave の出番はないかもしれませんが、旅行の間に撮った写真を都度 wave にアップロードしておくとなかなか便利です。Google Wave はドラッグ&ドロップで大量のファイルも簡単に添付でき、添付した画像のサムネイルも自動的に作成さ

れます。夜にホテルで友人とその日見たものについて話をしながら写真を整理するのも楽しい時間の過ごし方でしょう。

写真をその都度 wave にアップロードしておけば、旅行が終わったあとで共有のフォトアルバムを作って親しい人に公開するのも簡単です。Google Wave はある wave に添付されている全ての画像をコピーして新しい wave を作成することができます。そのようにして作られた旅行中に撮りためた画像だけを含む wave にタイトルを付け、写真を見せたい人たちを wave に招待すればフォトアルバムの完成です。よくあるフォトアルバムサイトのように添付画像をスライドショーとして順に見ることもできます。

プライベートで利用する場合は、このようにしてイベントと絡めて使用するのが一番 wave の便利さを実感できるでしょう。

2.3 CRM（顧客管理）



企業が wave を業務に組み込む場合、社員間のコミュニケーションのために直接 Google Wave を使うというケースを除けば、顧客管理に利用するケースが一番多くなりそうです。Google I/O 2010 の企業・コミュニティによる wave 事例紹介ブースでもオラクル社を含む数社が CRM に wave を利用する実験的なプロダクトを公開していました。本節ではその中でも特に早くから wave を CRM で利用するプロトタイプを公開していたセールスフォース・ドットコム社（以下セールスフォース）の事例について紹介します[*2]。

セールスフォースが公開しているのはユーザーサポートロボットです。セールスフォースの提供しているサービスに問題が発生するとユーザーは Google Wave を立ち上げ、サポートロボットを呼び出します。ロボットはセールスフォースの顧客情報にアクセスでき、必要であればユーザーに応じて返答の内容をカスタマイズすることも可能です。ロボットは Known Issue を提示したり、ユーザーの入力を読み取って登録されているヘルプ記事から関連のありそうなものを一覧して、まずは単独でユーザーの問題を解決しようとします。

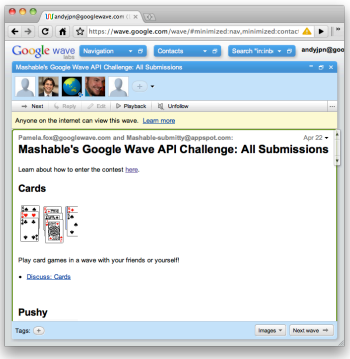
ロボットだけでは問題を解決できそうになければ、ユーザーの要望に従ってサポート担当者を wave の参加者に追加します。追加されたサポート担当者はそれまでのロボットとユーザーとのやり取りを読むことができますし、必要であればプレイバックを利用して流れを確認した上で、ロボットからサポートを引き継ぐことができます。

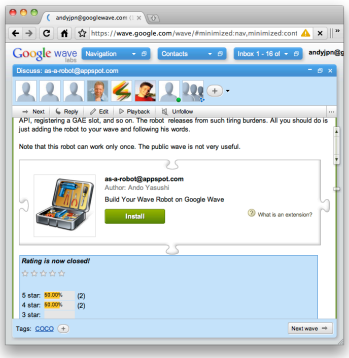
[*2] 本例はあくまでもプロトタイプで、実際に利用されているわけではありません。参考: http://www.youtube.com/watch?v=TQ0b1CVRZHs

サポート担当者の PC 上ではセールスフォースの管理ツールに wave が組み込まれていますが、wave 上でのユーザーと担当者との会話は特殊なところはありません。ファイルを添付したり、画像やガジェットを使用することで適切にユーザーの状況を把握しサポートすることができるでしょう。難しい状態に陥ったとしても、ユーザーには見えないプライベートリプライ機能を利用すれば、上司や技術担当者と相談しながらサポートを継続できます。

企業のサポートは適切な回答者に辿りつくまで様々な部署をたらい回しされ、その度ごとに同じ説明を繰り返す羽目になりがちですが、wave を利用すればそのようなことにはなりません。セールスフォースだけでなくオラクル社や SAP 社も Google Wave に注目していますので、気がつくと企業のサポートはほとんどが wave 経由になる未来もありえるのかもしれません。

2.4　プログラミングコンテスト





2010 年 4 月に Google と Mashable の協賛で Wave 拡張のコンテストが開催されました。ここではそのコンテストにおいてどのように Google Wave が活用されていたかを紹介します。

まず、コンテストへの応募はもちろん Google Wave からです。Mashable Submitty という拡張をインストールし、「New extension submission」ボタンをクリックすると応募用の wave が開きます。応募用の wave にはロボットが参加者として登録されていて、このロボットにより初めから応募フォームのテンプレートや応募のワークフローを管理するためのガジェットが wave に用意されています。

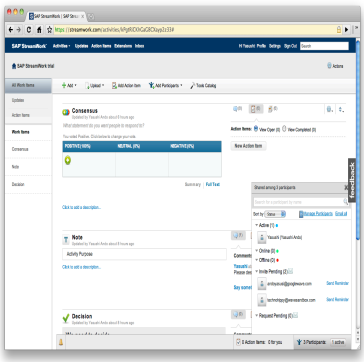
応募用 wave には拡張のインストーラー URL を入力する欄があり、この欄に URL を入力するとロボットがその内容を読み取って自動的に応募される拡張のインストーラーを wave に追加します。これはあとでレビューアーが拡張の動作を簡単に確認できるようにするものです。応募に必要な項目の記入が終わり、ワークフローガジェットの「Share with Reviewers」ボタンをクリックするとその wave の参加者としてレビューアーが追加されます。入力に不備があった場合は wave 上でレビューアーと直接やり取りをして内容を修正できます。

ワークフローガジェットは応募者とレビューアーで表示が異なり、レビューアーだけがレビュープロセスを完了できたようです。レビューが終了するとロボットによって自動的にその応募作品専用の wave が作られます。応募用の wave には先ほど入力した拡張の説明やインストーラーに加え、投票用のガジェット、twit ボタンや関連ページへのリンクな

ども追加されています。もちろん全応募作品をまとめた wave も用意されていて、さらにその全作品へのリンクを含む wave はエンベッド API を利用して Mashable のサイト上にも表示されていました。

このように Google Wave はロボットやガジェットなどの拡張機能を駆使することで、まるで全く新しいウェブサービスのように使うことも可能です。拡張機能は単体で利用するだけでなく、ここでの例を参考に組み合わせて使うことを考えると Google Wave の可能性がさらに大きく広がるでしょう。

2.5 SAP StreamWork



SAP 社はかなり早い段階から Google Wave に注目していて、Google Wave の発表後すぐに SAP Gravity という Google Wave 上で動くビジネスプロセスモデリングツールのプロトタイプを公開したりしていました[*3]。そんな SAP 社が 2010 年 3 月 30 日に公開したコラボレーティブな意思決定支援ツールが SAP StreamWork[*4]です。

StreamWork を一言で表せば SAP 社によるビジネス特化型の Wave です。はじめにミーティング・意思決定・プロジェクト計画などのテンプレートから目的に応じたものを選択してアクティビティを開始し、必要であれば組織外の人も含む参加者を追加して、多人数でリアルタイムに業務上のやり取りを進めていく様子はまさに Wave そのものですが、目的を絞ったことによって業務用ツールとしてはやはり Google Wave よりも洗練されているという印象を受けます。

その StreamWork の概念実証として、Wave の連合プロトコルを使用して Google Wave と共同するデモ動画が公開されています[*5]。デモ動画では StreamWork のアクティビティの参加者として Google Wave のユーザーを追加し、それぞれのツール間でリアルタイムにメッセージをやりとりする様子を見ることができます。また StreamWork では「Tool」という Google Wave でいうガジェットのような追加機能が利用できますが、驚いたことにこのツールは Google Wave 上でガジェットとして扱われ、それぞれの間でリアルタイムに操作や表示の内容が共有されていました。

Google Wave で実現される密度の高いコミュニケーションは、目的を同じくする集団でこそ有効です。そしてその目的を同じくする集団の筆頭が会社組織ではないでしょう

[*3] http://www.youtube.com/watch?v=FaNhXPSCQWo
[*4] http://sapstreamwork.com/
[*5] http://www.youtube.com/watch?v=3HuLnoxxz5U

か。Google Wave そのものが利用されなくても、StreamWork のように Wave の考え方を取り入れたビジネスツールはこれからもどんどん出てくるでしょう。それらのビジネスツールが Wave の連合プロトコルでお互いにやりとりできるようになれば、相手がどのようなツールを使っているかを全く気にせずにリアルタイムにやりとりできるようになります。利用しているグループウェアがいつのまにか wave プロトコル対応をしていて、気がつくと Wave ユーザーだった、ということがそう遠くない将来に起こるかもしれません。

・・・

以上、いくつかの使用例を紹介しました。ただしこれらはあくまでも Google Wave の利用シーンのほんの一例に過ぎず、ここで紹介したような使い方しかできないということではもちろんありません。Google Wave はその開発グループのメンバーでさえ何に使えるのかを掴みきれていないかのような発言をしています。ここに書いた使い方に囚われることなく、いろいろなことに Google Wave を利用して、なにか新しい適した使い方が見つかったときにはぜひ積極的に周囲に広めてもらえると嬉しく思います。

第 II 部

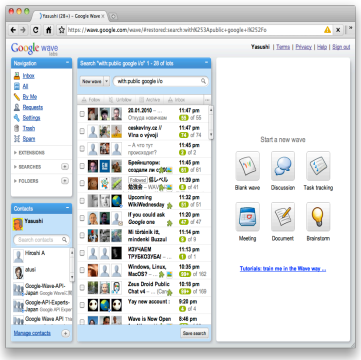
Google Wave の 3 つの P

第 3 章

プロダクト

3.1 はじめに

本章では Google Wave をプロダクトとして、つまり GMail・Google Docs などのような Google が提供するウェブサービスの一つを指すものとして、その画面構成や使い方について説明します。



上は Google Wave にログインした直後の図です。これを見れば分かるように、ウェブサービスとしての Google Wave は、一見 GMail と変わらないようにも見えます。しかし第一章で紹介した通り、その提供するサービスは、GMail を大きく超えています。Google の既存のサービスに例えるなら GMail（メール） + GTalk（チャット） + GoogleDocs（文書管理） + iGoogle（ガジェットプラットフォーム）といったところでしょうか。そのように複雑な機能を合わせ持つサービスを、それよりも遥かに機能の少ない GMail と同程度にシンプルなインターフェースにまとめる辺りは、さすが Google と言えるでしょう。

3.1.1 注意事項

3.1.1.1 Google アカウント

Google Wave を使用するには Goolge アカウントが必要です。この本を手に取る方でまだ Google アカウントを持っていない人はあまりいないと思いますが、もし持っていない場合は以下のサイトの右下にある「アカウント作成」リンクをクリックして、必要な内容を入力しアカウントを作成しておいてください。

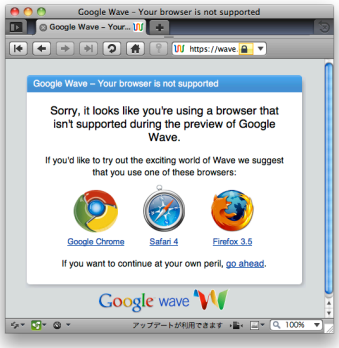
- https://www.google.com/accounts/Login

3.1.1.2　対応ブラウザ

Google Wave は新しい Web 標準である HTML5 の機能を大いに活用したウェブサービスです。古いブラウザや、Web 標準に則っていないブラウザでは正しく動作しない可能性があります。執筆時点で公式に対応しているのは以下の 3 つのブラウザです。

- Google Chrome
- Safari 4 以上
- Firefox 3.5 以上

上記以外のブラウザでアクセスすると次のような画面が表示されます。



上記の画面が表示されても画面下部の「go ahead」リンクをクリックすれば Google Wave は実行できます。ただしその場合は、正しく表示されなかったり、一部機能が動作しないことがありえます。なるべく推奨された 3 つのブラウザのいずれかを使用しましょう。

3.1.2　Google Wave にログイン

先の要件を満たすことができたら、さっそく Google Wave を使ってみましょう。Google Wave の URL は以下です。

- http://wave.google.com

3.2　Google Wave の画面各部の名称

Google Wave の使い方に進む前に、まずは大まかに各部の名称と役割について理解しておきましょう。

Google Wave は大きく次の要素で構成されています。

1. ヘッダー
2. ナビゲーションペイン
3. コンタクトペイン
4. 検索ペイン
5. wave

なお、ペイン（Pane）についてはパネル（Panel）と呼ばれることも多いようです。迷いましたが、本書ではペインで統一しています。

3.2.1 ヘッダー

ヘッダーにはロゴと、アカウント名、ヘッダーリンクがあります。ロゴは標準の Google Wave とサンドボックス版で異なります。拡張の開発のために両方を開いていて、どちらを見ているか混乱したときはロゴを確認するといいでしょう。

標準

サンドボックス

ヘッダーリンクは Terms、Privacy、Help、Sign out からなり、それぞれのリンク先は次のような内容です。Help のリンク先は実際には単なるヘルプを超えたチュートリアル的な内容を含む記事も多いので目を通しておくといいでしょう。

3.2.2 ナビゲーションペイン

ナビゲーションペインは wave のグループ化に関連する機能を持ちます。具体的には以下のようなことができます。

- Inbox、Spam、Trash（ゴミ箱）などデフォルトフォルダでの分類
- 新着・未読 wave の確認
- 検索条件の保存
- フォルダの作成とそれによる wave の管理

リンク	内容
Terms	Google Wave のさまざまな規約へのリンクです。法的な注意事項・プライバシポリシー・コンテンツポリシー・利用規約に興味が興味がある場合は読んでおきましょう
Privacy	プライバシポリシーへのリンクです。Google がどのような情報を収集して、それらがどのように使用されるかに興味がある場合は読んでおきましょう
Help	Google Wave に関する様々なリソースへのリンクがあります。もし余裕があれば一通り目を通しておくことをおすすめします
Sign out	サインアウトしてログイン画面に戻ります

表 3.1　ヘッダーリンク

- 拡張インストーラー一覧の確認

ナビゲーションペインで選択したグループに属する wave の一覧は検索ペインに表示されます。本ペインの詳細な使い方については 3.3.5 節で説明します。

3.2.3　コンタクトペイン

コンタクトペインはアドレス帳のような役割を持ち、以下のようなことができます。

- コンタクトカードの一言メッセージを編集
- 検索フィールドで wave アドレスを検索
- これまでやり取りをしたことのあるユーザーやロボットの一覧を表示
- 一覧や検索結果から参加者を選んで wave を開始
- wave アドレスの追加

上記以外のより細かなコンタクトリスト管理はこのペイン下部の `Manage contacts` リンクをクリックして別画面で行うことになります。
本ペインの詳細な使い方については 3.3.6 節で説明します。

3.2.4　検索ペイン

本ペインでは wave の参加者と本文の一部を更新時刻順に一覧できます。ログイン直後は Inbox の内容が表示されていますが、検索条件を変更したり、ナビゲーションペインでフォルダを選択することで様々な条件で wave を一覧できます。検索以外にも本ペインでは以下のようなことができます。

- 新規 wave を作成
- 複数 wave の状態をまとめて更新
- 詳細を表示する wave を選択

本ペインの詳細な使い方については 3.3.2 節で説明します。

3.2.5 wave

wave には大きく二つの意味があります。一つは W が大文字になる固有名詞としての Wave。これは今見ているサービスそのものであったり、そのサービスが使用しているプロトコルのことであったりします。そしてもう一つが本章で説明する「表示要素としての wave」で、これこそが Google Wave サービスの中核です。これまで挙げた 3 つのペインは全てこの wave やその参加者を管理するために使用されるものです。

それではそのサービスの中核たる wave とは何でしょうか。それは、一言でまとめるなら「複数人が共同で編集できるツリー構造を持ったドキュメント」となります。ただしここで言うドキュメントには文章だけではなく画像や動画、インタラクティブなゲームのようなものまでを含むことができるということに注意してください。

wave の詳細な使い方については 3.3.3 節と 3.3.4 節で説明します。

なお、混乱を防ぐため本書ではサービスとしての Wave は「Google Wave」に表記を統一しています。表題や行頭にあるため先頭が大文字になり Wave と表記されていたとしても、それは表示要素[*1]としての wave を指していると考えてください。

3.3 Google Wave の使い方

3.3.1 各ペイン・wave 共通の操作

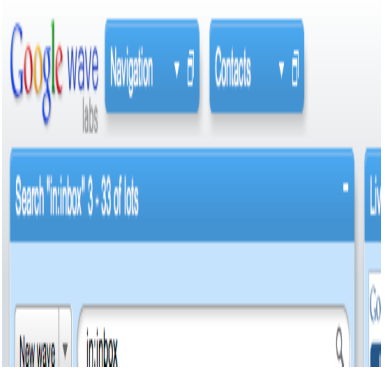
それぞれのペインの使い方に進む前に、全ペイン・wave で共通の操作について説明しておきます。なおこの節ではペインと wave の両方を合わせて「ペイン」と呼んでいます。

全ペインの共通部分を抜き出すと上図のようになります。青色のヘッダ部とその下のボディ部、大きく二つに分かれ、ヘッダ部にはペインタイトルとボタン、ボディ部にはその内容とペインサイズに応じてスクロールバーが表示されます。スクロールバーは少し変わっていて、ヘッダーから下の方にあるストッパーまでの間で動かすことができます。

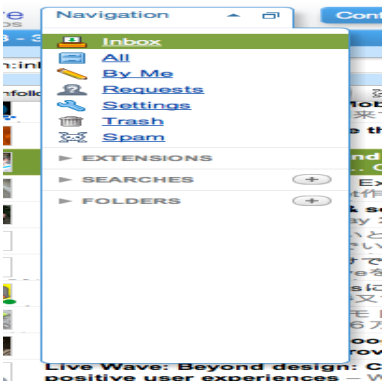
3.3.1.1 ペインを最小化する

ヘッダの右方にある最小化ボタン  をクリックするとペインは最小化され、ヘッダーのみになって画面上方にまとめられます。

[*1] またはデータモデル



上の図はナビゲーションペインとコンタクトペインを最小化したところです。二つのペインを最小化することによって検索ペイン（Inbox）が一回り大きく表示されました。最小化を解除するにはヘッダーの右側にある最小化解除ボタン  をクリックします。なお、最小化された状態でも  ボタンをクリックすることでそのペインの機能を利用できます。



3.3.1.2　ペインサイズを変更する

先ほどペインを最小化する方法を説明しましたが、もう少し細かくペインのサイズを調整したいこともあるでしょう。それにはペインとペインの隙間にマウスカーソルを合わせ、カーソルが  または  のような形[*2]になったところで、上下または左右にドラッグ&ドロップしてください。ペインの境界を移動してサイズを変更できます。

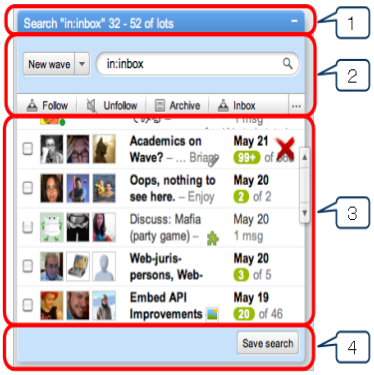
[*2] ブラウザによって多少異なります

実際にこれほど極端にサイズを変更することはまずありませんが、例としてペインの比率を大幅に変更してみました。Google Wave ではペインの配置は固定されているのでナビゲーションペインやコンタクトペインの縦幅を調節することはよくあります。

それでは、ここからは個々のペインの説明に移ります。

3.3.2 メッセージ管理（検索ペイン）

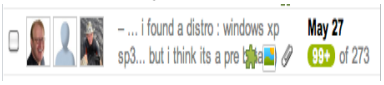
特定の条件でメッセージを一覧したり、複数メッセージをまとめて管理するには検索ペインを使用します。



1. **タイトル**　検索条件とリストの範囲が表示されます
2. **ツールバー**　新規 wave 作成や検索、wave の属性変更などのツールが利用できます
3. **ダイジェストリスト**　wave の属性やダイジェストが確認できます
4. **フッター**　現在ダイジェストリストの表示に使用している検索条件を保存できます

3.3.2.1 wave の情報を読み取る

検索ペイン中央にあるダイジェストリストでは多数の wave の情報をまとめて確認することができます。



一つの wave に対応するリストアイテムの典型的な見た目は上のようになっています。表示されている内容は左から順に

チェックボックス　複数 wave の属性をまとめて変更するときに使用します

参加者のプロフィール画像　wave の参加者が最大 3 人まで表示されます。プロフィール画像には以下のようにいくつか特殊なものもあります

タイトルと本文のダイジェスト　wave の内容に応じてダイジェストの前後にアイコンが表示される場合があります。アイコンの意味するところはそれぞれ以下のようになります

wave **の最終更新時間と内部に含まれる未読** wavelet **数と全** wavelet **数**　未読数は緑背景に白抜きの数字で表示されますが、未読が 100 件以上になると正確な数値ではなく「99+」と表記されます

	プロフィール画像の右下に緑ドットが表示されている場合は、そのユーザーが Google Wave にログイン中であることを示します
	コンタクト管理画面でプロフィール画像を設定していない場合に使用される、デフォルトアイコン
	参加者がグループである場合に使用されるアイコン
	公開用ユーザーのアイコン。このユーザーが参加している wave は誰でも読み書きできます
	Google Wave 招待状送付中ユーザーのアイコン。GMail または wave アドレス以外のメールアドレスの参加者を追加すると wave の招待状が送付され、登録が完了するまではプレースホルダーとしてこのアイコンが表示されます

表 3.2　検索ペインプロフィールアイコン

	ガジェットアイコン。wave 内でガジェットが使用されていることを示します。ガジェットについては 4.2 節を参照してください
	画像アイコン。wave に画像が添付されていることを示します。
	添付ファイルアイコン。wave に画像以外のファイルが添付されていることを示します。
	Unfollow マーク。Unfollow された wave であることを示します。Unfollow された wave は Inbox には表示されなくなり、更新があっても通知されることはありません。ただしフォルダ内には存在していますし、検索も可能です
	フォルダ名。色つきのフォルダに属している場合はダイジェストの前にフォルダ名が表示されます

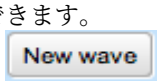
表 3.3　オプションアイコン

3.3.2.2　wave の内容を表示する

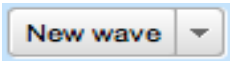
ダイジェストリストを確認して気になる wave があったときには、そのリストアイテムをクリックしてください。検索ペインの右に wave が開き、内容を閲覧・編集することができます。wave の詳しい使い方に付いては 3.3.3 節と 3.3.4 節を参照してください。

3.3.2.3 新規 wave を作成する

検索ペイン左上の「New Wave」ボタンをクリックすると新規 wave が作成されます。作成直後の wave の参加者は自分一人だけです。また拡張をインストールすると「New Wave」ボタンの横にカスタム wave 作成用のボタンが追加されることがあり、そのボタンを使用すれば特定のロボットやガジェットが予め組み込まれた wave を作成することもできます。

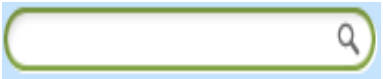


拡張インストール前



拡張インストール後

3.3.2.4 wave を検索する



検索ペイン上部の検索ボックスに検索条件を入力し、return キーを押すか、検索ボックス右端の検索ボタンをクリックすると wave を検索できます。検索条件はいろいろと便利な指定方法がありますので、次の表を参照してください。なお、プライベートリプライが検索条件に合致した場合は、そのリプライを含む wave が検索結果に現れます。

キーワード指定による検索

about:[キーワード]	タイトルや本文に [キーワード] が存在する wave を検索します。about:を付けずに直接 [キーワード] を指定した場合と同じです
title:[キーワード]	wave のタイトルに [キーワード] が存在する wave を検索します
caption:[キーワード]	添付したファイルのキャプションに [キーワード] が存在する wave を検索します

ステータスによる検索

is:read	プライベートリプライを含む全てのメッセージを読み終わった wave を検索します
is:unread	少なくとも一件の未読メッセージがある wave を検索します
is:filed	フォルダに格納されている wave を検索します
is:unfiled	フォルダに格納されていない wave を検索します
is:followed	follow している wave を検索します
is:unfollowed	follow していない wave を検索します

参加者による検索

creator:[名前]	指定された [名前] のユーザーが作成した wave を検索します
participant:[名前]	指定された [名前] が参加している wave を検索します。名前はグループ名でも構いません
contributor:[名前]	指定された [名前] のユーザーが少なくとも一つのメッセージを編集している wave を検索します
to:[名前]	指定された [名前] のユーザーが参加者であって、作成者ではない wave を検索します
onlyto:[名前]	指定された [名前] のユーザーが参加者であって、作成者ではなく、参加者が 2 人だけの wave を検索します
onlywith:[名前]	指定された [名前] のユーザーだけが参加者の wave を検索します
dfrom:[名前]	指定された [名前] のユーザーからの ping またはそのユーザーが作成者で、参加者が自身と二人きりである wave を検索します
dto:[名前]	指定された [名前] のユーザーへの ping またはそのユーザーがコントリビューターで、参加者が自身と二人きりである wave を検索します
from:[名前]	指定された [名前] のユーザーが少なくとも一つはメッセージを作成している wave を検索します
to:[名前]	指定された [名前] のユーザーが参加者で、作成者ではない wave を検索します
with:[名前]	指定された [名前] のユーザーが直接の参加者である wave を検索します
by:[名前]	指定された [名前] のユーザーがコントリビューターである wave を検索します
is:note	自身だけが参加者である wave を検索します

発言日による検索

past:[期間]	検索当日を含む、[期間] の間に発言のあった wave を検索します。past:1d は今日と昨日の wave を検索します
previous:[期間]	検索当日を含まず、[期間] の間に発言のあった wave を検索します。previous:1d は昨日と一昨日の wave を検索します
before:[期間]	指定された [期間] よりも前に発言のあった wave を検索します
after:[期間]	指定された [期間] よりも後に発言のあった wave を検索します

発言日の指定には day, week, month, year が使用可能です。つまり past:1day、previous:1week などと書けます。複数形も利用可能で after:3weeks とも書けます。また省略形として d, w, m, y も使用でき、past:1y などと書くこともできます。

フォルダによる検索

in:[フォルダ名]	指定された [フォルダ名] のフォルダ内の wave を検索します。デフォルトの検索条件である in:inbox もこの形式です
in:[検索名]	指定された [検索名] で保存された検索条件で検索します
is:unfiled	ユーザーが作成したフォルダに移動されていない wave を検索します
is:filed	ユーザーが作成したフォルダ内の wave を検索します

添付ファイルによる検索

has:attachment	添付ファイルを持つ wave を検索します
has:document	ドキュメントが添付された wave を検索します（画像だけが添付された wave は含まれません）
has:image	画像が添付された wave を検索します
caption:[キーワード]	添付ファイルのキャプションに [キーワード] が含まれる wave を検索します
filename:[キーワード]	添付ファイルのファイル名に [キーワード] が含まれる wave を検索します
mimetype:[キーワード]	添付ファイルの MIME タイプに [キーワード] が含まれる wave を検索します

タグによる検索

has:tag	タグを持つ wave を検索します
tag:[タグ名]	指定された [タグ名] のタグを持つ wave を検索します

ガジェットによる検索

has:gadget	ガジェットを持つ wave を検索します。ガジェットについては 4.2 節を参照して下さい
gadgeturl:[キーワード]	指定された [キーワード] を含む URL のガジェットを持つ wave を検索します
gadgettitle:[キーワード]	指定された [キーワード] を含むタイトルのガジェットを持つ wave を検索します

その他の検索

id:[Wave ID]	指定された [Wave ID] を持つ wave を検索します
title:[キーワード]	タイトルに [キーワード] を含む wave を検索します
has:link	リンクを持つ wave を検索します
link:[URL]	指定された [URL] へのリンクを持つ wave を検索します

3.3.2.5　複数の条件を組み合わせて検索する

複数の検索条件を組み合わせるために検索ボックス内では以下のような演算子が利用できます。

[条件 1] & [条件 2]	[条件 1] と [条件 2] の両方が成り立つ wave を検索します。AND とも書けます。複数条件文を書いたときのデフォルトなので、単に空白で区切った場合もこの動作になります
[条件 1] |[条件 2]	[条件 1] と [条件 2] のいずれかが成り立つ wave を検索します。OR とも書けます
-[条件]	[条件] が**なりたたない**wave を検索します
"条件 条件"	先に書いた通り空白区切りで複数の条件文を書くと AND 条件とみなされます。空白を含むキーワードで検索したいときは""（ダブルクォーテーション）で囲ってください。

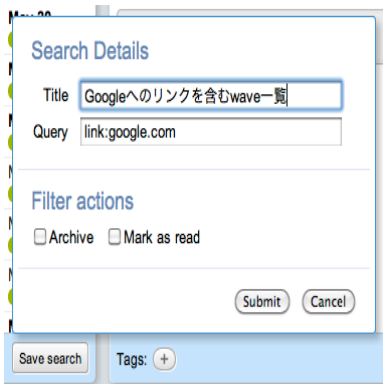
上記は組み合わせて使うこともできて、例えば

[条件 1] & ([条件 2] |-[条件 3])

と言う条件は「条件 1 が成り立ちかつ、条件 2 が成り立つが条件 3 は成り立たない」wave を検索します。

3.3.2.6　検索条件を保存する

検索ペインの一番下にある「Save search」ボタンをクリックすると現在の検索条件に名前をつけて保存することができます。保存した検索条件はナビゲーションペインの「SEARCHERS」に追加されます。



チェックボックスをチェックすると自動的にアーカイブに保存したり、既読にすることもできます。

3.3.2.7 ライブサーチ

Google Wave はライブサーチという面白い仕組みを採用しています。ライブサーチは Google Wave の基本機能で、ユーザーが特別になにかをするまでもなく有効ですが、せっかくですのでここで少し説明しておきます。

検索を実行すると検索結果がペイン中央に表示されますが、この結果は静的なものではありません。検索条件に合う wave が新規に作成された場合は検索を再度実行しなくても即座に検索結果にその wave が追加されますし、表示されていた wave が編集され検索条件から外れた場合は即座にリストから削除されます。例えば「with:public」という条件で検索してそのまま眺めていると、全世界で公開 wave が次々に作成されている様子が見て取れ、まるでかつての twitter の public timeline を見ているような気分に浸れます。

3.3.2.8 wave の属性を変更する

ダイジェストリストをクリックして wave を選択すると、リストの上にあるツールバーが利用可能になります。ツールバーボタンは大きくステータスの変更とフォルダの移動という 2 つの機能を持ち、それぞれ以下のような意味を持ちます。

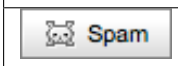
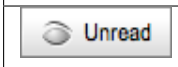
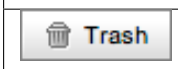
	wave を follow して、変更があったときに Inbox に表示されるようにします
	wave を unfollow して、変更があっても Inbox に表示されないようにします
	wave をアーカイブします。All には表示されますが、変更があるまで Inbox には表示されなくなります
	wave を Inbox に移動します
	wave を Spam フォルダに移動します
	wave を既読にします
	wave を未読にします
	wave を Inbox から取り除きます。更新があっても Inbox には現れません
	クリックするとフォルダ一覧が開き、選択したフォルダに wave を移動します

表 3.4 ツールバーボタン

ダイジェストリスト左端のチェックボックスをチェックするか、Shift ボタンを押しながらリストアイテムをクリックすることで複数 wave を選択することもでき、その状態でステータスボタンをクリックするとまとめてステータスを設定できます。

3.3.3　メッセージ閲覧（wave：ビューモード）

wave には閲覧専用のビューモードと編集可能なエディットモードがあり、既存の wave をオープンした直後はビューモードに設定されています。本項ではそのビューモードの使い方について説明します。

1. **タイトル**　wave のタイトルが表示されます
2. **ツールバー**　wave の参加者リストが表示され、プレイバックやモード・属性の変更などのツールが利用できます
3. **メッセージスレッド**　wave の本文が表示されます
4. **フッター**　タグの表示・設定や、次の未読へのナビゲーションが利用できます

3.3.3.1　wave を読む

wave の大部分を占めるメッセージスレッドには参加者の発言がツリー上に並んでいます。メッセージは時系列に並んでいるとは限りませんし、インデントのルールも少しだけ複雑です。詳細についてはメッセージ編集の項（3.3.4 節）で説明しますが、おおまかに次のルールに従っていると考えてください。

- 一番上の発言が wave を開始した発言（ルートメッセージ）で、それ以外は全ていずれかの発言への返信
- ルートメッセージ以外の発言は全てそれに先立った発言（親メッセージ）を持っている
  - インデントが同じグループの一番上のメッセージは、インデントが一つ浅いグループで、一つ上のメッセージに対する返信
  - インデントが同じグループの 2 番目以降のメッセージは、インデントが同じで一つ上のメッセージに対する返信

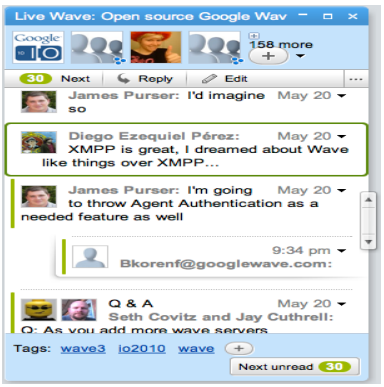
上図では返信メッセージから返信元のメッセージへ矢印が伸びています。

また、メッセージを読んでいる最中に次のように他の参加者の名前と共に発言が書き加わって行くことがあります。

これは今まさにその人がネットワークの向こうで Google Wave を開き、このドキュメントを編集しているということです。発言の内容にも寄りますが、Google Wave の特徴であるリアルタイムコミュニケーションのチャンスだと思って、思い切って返信してみてもいいかもしれません。

3.3.3.2 未読メッセージを確認する

既存の wave を開くと一番古い未読メッセージにフォーカスが当たります。また、それ以外の未読メッセージはメッセージの左枠が緑色になります。

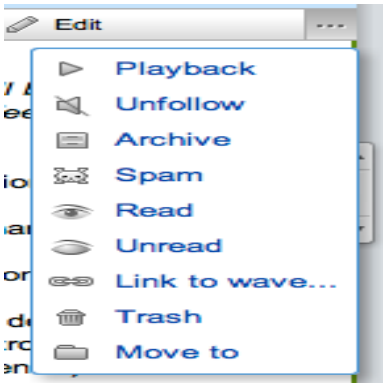


検索ペインでは未読のメッセージがある wave が上位に表示されますが、メッセージは発言順に並んでいるわけではないので、wave を開いても新しく編集された wave はいろいろな場所に分散しています。左端のマークを参考に効率よく未読メッセージを確認しましょう。未読メッセージにフォーカスが当たるとそのメッセージは既読になります。

3.3.3.3　wave の属性を変更する

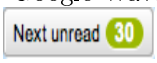
ビューモード時のツールバーには表 3.4 と同じボタンが揃っています。それらをクリックすることで閲覧中の wave のステータスを変更したり、フォルダに移動したりできます。

また、もし wave の横幅が狭くツールバーアイテムが全て表示されていないときは右端のボタンをクリックしてください。表示されていなかったボタンが次のように表示されます。



3.3.3.4　次のメッセージを表示する

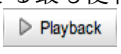
wave を複数人が編集するとメッセージは上から下へ並ぶことはなく、あちらこちらがばらばらに編集されることになります。そのようなドキュメントを編集された順番に読むためのボタン（とキーボードショートカット）が Google Wave には用意されています。ツールバーの 

ボタンとフッターの 

ボタンがそれです（緑の数字は未読メッセージ数です）。これらのボタンを使用するとメッセージにを編集された順に確認できます。さらに、表示中の wave の未読を全て読み終えた状態でそれらのボタンをクリックすると、Inbox 内の次の未読のある wave を、未読メッセージにフォーカスが設定された状態で開きます。つまりこのボタンを連続してクリックするだけで Inbox 内の全ての未読を順番に確認できます。

また、wave がビューモードのときにスペースキーを押下することでも、同様の機能が利用できます。

3.3.3.5　メッセージをプレイバックする

先の項の機能を使用することで未読を順番に確認できますが、それだけでは十分ではないかもしれません。また、既読の文章がどのような順番で編集されたか確認したいこともあるでしょう。そのような場合に利用できる最も便利で強力な機能がプレイバックです。wave のツールバーのプレイバックボタン 

をクリックしてみましょう。ツール

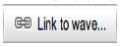
バーの内容が変わり、ルートメッセージだけが表示されたはずです。

これがプレイバックモードで、新しく表示されたツールバーのスライドバーやボタンを利用して、メッセージを発言順に再生できます。

	最初のメッセージだけを表示します
	wave を一コマ巻き戻します
	wave を一コマ進めます
	最新の状態の wave を表示します
	左から右に移動するに連れて新しい状態の wave を表示します
	現在表示されている状態まで wave の内容を巻き戻します。ただし、履歴が削除されるわけではありません
	プレイバックモードを終了します

現在全何コマの中の何番目を表示しているかはツールバーの下に表示されています。先の例では全 11 バージョン中最初の状態を表示していることが分かります。

3.3.3.6 wave のリンクを取得する

Google Wave は wave ごとに一意な URL を持っています。つまり誰かに特定の wave を開いてもらいたいときには、その wave を開いている時の URL を伝えるだけで構いません。ただ、ブラウザのアドレスバーに表示されている URL は wave の最大化の状態や複数 wave の配置など、単なるに wave の URL 以上の情報を含んでいるため、非常に長大になりがちです。必要最低限の URL が知りたい場合にはツールバーの 

ボタンをクリックしてください。次のようなダイアログが開き、wave の URL をコピーできます。

さらにダイアログには URL だけではなく、wave の URL の下には wave をエンベッドする（4.6 節を参照）ためのタグも表示されていて、外部サイトに表示中の wave を簡単に貼りつけることもできます。

3.3.3.7　wave を最大化する

wave のヘッダ部にある最大化ボタン  をクリックすると、他のペインを全て最小化してその wave だけを大きく表示できます。一つの wave をじっくりと読みたい場合は最大化してから読むといいでしょう。また、友人に wave の URL を教えるときに、最大化した wave の URL を伝えると、その友人は初めから最大化した状態で wave を読むことができます。なお、実際のところ、この最大化ボタンの動作は他の全てのペインの最小化ボタンをクリックすることと同じです。

3.3.3.8　wave を閉じる

wave のヘッダ部にあるクローズボタン  をクリックすると wave を閉じることができます。もちろんこのボタンを押しても wave が削除されるわけではありません。検索ペインで該当するダイジェストアイテムをクリックすればまた wave をオープンできます。

3.3.3.9　参加者の情報を見る

ツールバーの上にはその wave の参加者リストがあります。そのリスト内のプロフィール画像をクリックすると参加者のプロフィールを確認できます。

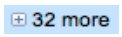
また、上図から分かるようにこのダイアログでは参加者のプロフィールを確認するだけではなく、次のようなことも実行できます。

- そのユーザーと自分だけが参加する新しい wave を作成する
- そのユーザーに ping[*3]を送る
- そのユーザーを wave の参加者から除く
- そのユーザーの wave の編集権限を設定する[*4]

参加者アイコンにはいくつか特殊なものがありますが、アイコンのサイズを除き、全て検索ペインで説明した内容と同じです。

	プロフィール画像の右下に緑ドットが表示されている場合は、そのユーザーが Google Wave にログイン中であることを示します
	コンタクト管理画面でプロフィール画像を設定していない場合に使用される、デフォルトアイコン
	参加者がグループである場合に使用されるアイコン
	公開用ユーザーのアイコン。このユーザーが参加している wave は誰でも読み書きできます
	Google Wave 招待状送付中ユーザーのアイコン。GMail または wave アドレス以外のメールアドレスの参加者を追加すると wave の招待状が送付され、登録が完了するまではプレースホルダーとしてこのアイコンが表示されます

表 3.5 wave プロフィールアイコン

参加者が多いときはツールバーの上の参加者リストには表示しきれないことがあります。全参加者を確認するには参加者リストの右端の 

と言う部分をクリック

[*3] 自分とそのユーザーだけが参加する wave を作成し、さらにそのユーザーが wave にログイン中の場合はポップアップして会話の開始を促します
[*4] 現在のところ Full access と Read only の二種類だけです

してください。参加者リストが拡大して全参加者を確認できます。表示を元に戻すには Fewer をクリックしてください。

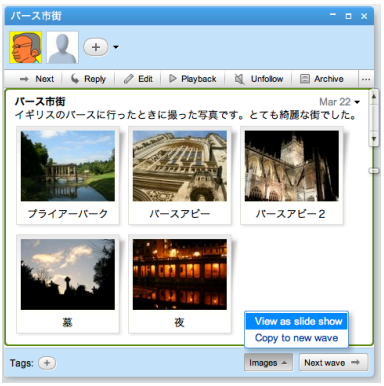
3.3.3.10　wave を複数開く

wave は複数開くことができます。ただし単純に検索ペインで wave を選択すると、そのとき開いている wave が置き換えられてしまいます。現在開いている wave を閉じずに、新しく wave を開くには、検索ペインで Ctrl キーまたは Command キーを押しながらダイジェストアイテムをクリックしましょう。

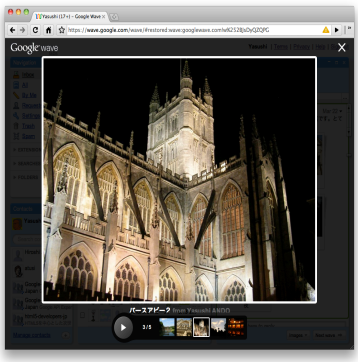
wave はいくつでも開くことができるようですが、あまりたくさん同時に開いても本文が見づらくなるだけです。検索ペインを最小化した状態で 3-4 個を同時に開くのが現実的なところではないでしょうか。なお、複数 wave が開いているときに検索ペインで wave を選択したり、「New Wave」ボタンをクリックすると右下の wave が置き換えられます。

3.3.3.11　添付された画像のスライドショーを見る

画像が添付された wave では、添付された画像をスライドショーとしてみることができます。スライドショーが利用可能な wave はフッターに Images ボタンが表示されています。クリックして「View as slide show」を選択しましょう。



wave 上ではアイコンサイズだった画像が大きく表示されます。



画面をクリックしたり、下のサムネイルを選択すると表示する画像を切り替えられる他、サムネイルの左側の再生ボタンをクリックすることで一定時間ごとに自動的に全ての画像を切り替えて表示することもできます。

3.3.3.12 iPhone や Android 携帯で見る

Google Wave は iPhone や Android などの HTML5 が利用可能なスマートフォンにも辛うじて対応しています。ブラウザで PC 版と同じ URL にアクセスしてみてください。

非対応ブラウザであることを示す画面が開きますが、気にせずに右下の「go ahead.」リンクをクリックします。

するとスマートフォンでも Google Wave を利用することができます。ただし、残念なことに画面の読み込みに非常に長い時間がかかり、実用に足るとは言い難いのが実際です。

3.3.3.13　ショートカットを利用して効率よくメッセージを閲覧する

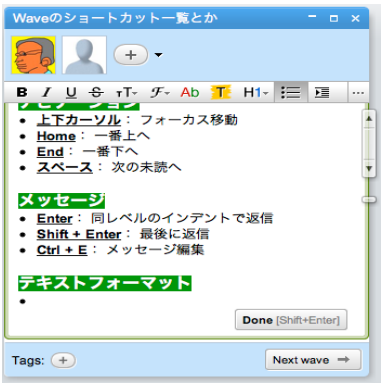
Google Wave では数多くのショートカットキーが使用できます。その中でビューモードの wave で利用できるショートカットとしては次のようなものがあります。

上下カーソルキー	上または下のメッセージにフォーカスを移動する
Home キー	wave 内の最初のメッセージにフォーカスを移動する
End キー	wave 内の最後のメッセージにフォーカスを移動する
スペースキー	wave 内の次の未読メッセージにフォーカスを移動する
左右カーソルキー	検索ペインと wave の間でフォーカスを移動する
Page Up/Donw キー	wave をスクロールアップ／ダウンする
Ctrl キー＋スペースキー	wave 内の全てのメッセージを既読にする

表 3.6　ビューモード wave ショートカットキー

3.3.4　メッセージ編集（wave：エディットモード）

前項で wave の基本的な使い方とコンテンツの閲覧について説明しました。本項ではさらに wave の編集について説明します。エディットモードの wave はショートカットやツールバーの内容がビューモードのときとは変化しています。



3.3.4.1 新しく wave を作成する

新しく wave を作成するには次のような方法があります。

1. 検索ペインのツールバーの「New Wave」ボタンをクリックする（3.3.2.3 節を参照）
2. wave の参加者を選択して、表示されるダイアログの「New Wave」ボタンをクリックする（3.3.3.9 節を参照）
3. wave ツールバーの「New wave with everyone here」メニューアイテムをクリックする（3.3.4.2 節を参照）
4. wave が一つも開いていない状態で、画面右側の wave テンプレートをクリックする（3.3.4.3 節を参照）
5. コンタクトペインのユーザーを選択して、表示されるダイアログの「New Wave」ボタンをクリックする

なお、新規作成直後に編集できるメッセージはルート blip と呼ばれ、それ以外のメッセージとは以下のような点で異なります。

- メッセージの一行目が wave のタイトルになります
- ルートメッセージは削除できません（wave そのものを削除したいときは、検索ペインの Move to ツールバーボタンを使用して wave を Trash に移動してください）

3.3.4.2 参加者を引き継いで wave を作成する

wave にあまりにも多数のメッセージが書き込まれると全体の見通しが悪くなり、同じ参加者で次の新しい wave を始めたくなるでしょう。しかし参加者が多いときに一人一人新しい wave に追加しなおすのは大変です。そのようなときには wave ツールバーの参加者追加ボタンの横にある小さな  ボタンをクリックし、「New wave with everyone here」ボタンを選択しましょう。表示している wave と同じ参加者で新しい wave を開始することができます。

3.3.4.3 テンプレートを使用して wave を作成する

Google Wave には wave の主な利用目的に合わせた便利なテンプレートが用意されています。テンプレートを使用して wave を開始するには、wave を一つも表示していないときに画面右側に表示されるボタンを利用します。

 Blank wave	テンプレートを使用せずに新規 wave を開きます
 Discussion	何か共通の議題について話しあうための wave を開きます。投票ガジェットが初めから組み込まれています
 Task tracking	複数人でタスク管理するための wave を開きます
 Meeting	議事録用の wave を開きます
 Document	ドキュメント用の wave を開きます。ドキュメントのステータスを表すためのガジェットが初めから組み込まれています
 Brainstorm	ブレインストーミング用の wave を開きます。ホワイトボードガジェットが初めから組み込まれています

表 3.7　wave テンプレートボタン

3.3.4.4　既存のメッセージを編集する

既存のメッセージを編集するには右上のメニューから「Edit」を選択します。

メッセージは他の人が作成したものであっても編集できますし、閲覧の時と同じく他の人と同時に一つのメッセージを編集することも可能です。編集を終了するには入力エリア下の Done [Shift+Enter] ボタンをクリックするか、編集中ではない他のメッセージをクリックしてください。

3.3.4.5 リッチテキストエディタを使う

エディットモードになるとツールバーの内容がビューモードから変化します。その中でテキストのフォーマットに関わるボタンは以下です。

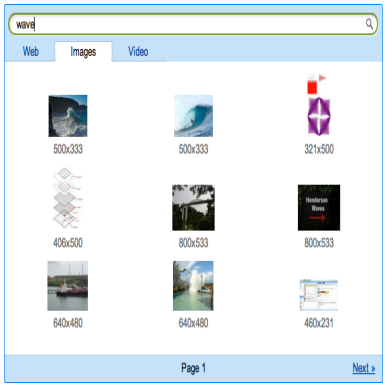
B	選択したテキストを太字にします
I	選択したテキストを斜体にします
U	選択したテキストに下線を付けます
S	選択したテキストに取り消し線を付けます
TT	選択したテキストのフォントサイズを設定します
F	選択したテキストのフォントファミリーを設定します
Ab	選択したテキストのテキストカラーを設定します箇条書きにします
T	選択したテキストの背景色を設定します
Tx	選択したテキストのフォーマットを全てクリアします
H1	選択したテキストをヘッダーにします
☰	リストを作成します
⇥	インデントを一段深くします
⇤	インデントを一段浅くします
≡	右寄せ・中寄せ・左寄せに設定します
Link	選択したテキストを外部サイトにリンクします

表 3.8 ビューモード wave ツールバー

3.3.4.6 wave に画像やファイルを添付する

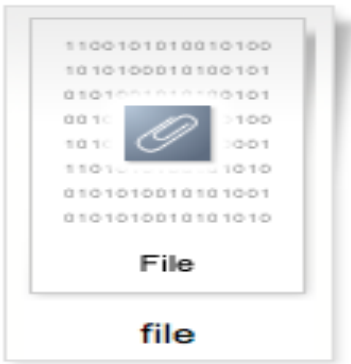
ローカルにあるファイルを wave に貼付けるにはツールバーのファイル添付ボタン をクリックするか、ファイルを wave にドラッグ＆ドロップします。ネット上にある画像を張り付けるには、Google の画像検索が利用できます。Google 検索ボタン をクリックしてダイアログを開き、検索した画像を選択してください[*5]。

[*5] Google 検索ダイアログでは画像のほか、Web サイトや動画も検索できます



添付されたものが画像であればその画像のサムネイルが表示され、クリックするとスライドショーが始まります。

画像ではないファイルを添付した場合は以下のようなサムネイルが表示され、クリックするとそのファイルがダウンロードされます。



特殊な拡張をインストールしていると添付ファイルの右上にメニューが追加されることがあります。その場合、メニューアイテムをクリックした際の挙動は拡張次第です。

3.3.4.7　wave にガジェットを追加する

ガジェットとは、簡単に言えば wave に組み込めるブログパーツのようなものです。ガジェットを利用すると単なるテキスト編集を超えたリッチなツールやゲームのような高度な機能を wave に埋め込むことができます。

ガジェットを追加する方法はインストーラーを使用する方法と、ガジェット XML の URL を自分で指定する方法の 2 つありますが、ここでは後者を説明します。まずツールバーのガジェットボタン  をクリックしてください。ダイアログが開くので、ガジェット XML の URL を入力すると、そのガジェットが wave に追加されます。

wave ガジェットは次のサイトにたくさんあります。キーワードで検索もできますので、いろいろと試してみるといいでしょう。

- http://wave-samples-gallery.appspot.com/

なお、いくつかのよく使うガジェットについては Google Wave に初めから専用ボタンが用意されています。

	Yes-No-Maybe ガジェット
	Google Maps ガジェット

表 3.9 ガジェットボタン

特に Google Maps ガジェットは、Google Wave チームの Lars Rasmussen 氏が元々担当していたプロダクトということもあってか、非常によくできています。座標だけではなく、プレースマークやラインまで共有され大変便利なので活用するといいでしょう。

3.3.4.8　メッセージに返信する

前節で見たように wave ではメッセージにリプライをつけることができますが、リプライは返信の対象やその参加者によってやり方が少しずつ違います。まずはその種類を確認しておきましょう。

リプライの種類	返信の対象	リプライを読めるユーザー
（ノーマル）リプライ	メッセージ	wave の参加者全員
インラインリプライ	メッセージの一部	wave の参加者全員
プライベートリプライ	メッセージ	リプライ作成者が選択

それでは以下にそれぞれのリプライを実行する方法を説明します。

メッセージに通常のリプライをする

通常のリプライを実行する方法は次のように数多くありますが、作成されるリプライ自体に違いはありません。好きな方法でリプライしてください。

ツールバーメニュー　ツールバーの 

ボタンをクリックすると、フォーカスが当たっているメッセージに返信します

メッセージのメニュー　メッセージの右上にある ▼ ボタンをクリックして表示されるメニューにはリプライ関連のアイテムが多くあり、それぞれ次のようなリプライを作成します。

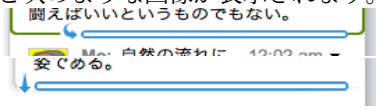
Reply　一番下のメッセージの場合は同じインデントレベルで、そうでないときには一つインデントレベルの深いリプライを作成

Indented reply　一つインデントレベルの深いリプライを作成

Continue thread　同じインデントレベルにある一連のメッセージの一番下にリプライを作成

Insert above　同じインデントレベルで一つ上にリプライを作成

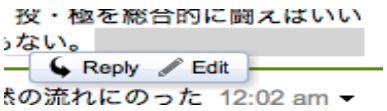
マウスオーバーで表示される返信エリア　メッセージの下端にマウスカーソルを合わせると次のような画像が表示されます。



前者をクリックすると一つ深いインデントレベルで、後者をクリックすると同じインデントレベルで、リプライが作成されます

wave **最下部のリプライボタン**　wave の一番下にある次のボタンをクリックするとインデントされていないリプライが一番下に追加されます

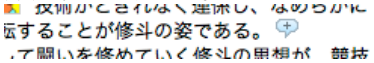
ダブルクリックで表示されるミニメニュー　メッセージの文章ではない部分、つまり下の図で四角く塗りつぶされている部分をクリックすると小さなメニューが表示され、その中の「Reply」ボタンをクリックするとメッセージにリプライできます



メッセージの一部にリプライをする

インラインリプライはメッセージの中のある行に対して残すことができるリプライです。時にはメッセージ全体ではなくその一部に注釈のような形でコメントを残したいこともあるでしょう。そのような場合にインラインリプライを使用します。メッセージの中の注釈を付けたい部分をダブルクリックすると小さなメニューが開くので[*6]「Reply」ボタンをクリックしてください。

インラインリプライがある行にはインラインリプライの表示非表示を設定するアイコンが表示されます。クローズボタン  をクリックするとインラインリプライは次のように非表示になります。



オープンボタン  をクリックするとまたインラインリプライが表示されます。また、一つのメッセージ内に数多くのインラインリプライがあり、まとめて開閉したいときにはメッセージの右上にある ▼ ボタンをクリックして表示されるメニューにある次の 2 つのアイテムを使用することもできます。

メッセージにプライベートなリプライをする

あるメッセージに対して、wave 参加者全員ではなく一部の人にだけ見えるようにリプライしたい場合や、他の人には見える必要のない個人的なメモを残したい場合はプライベートリプライを使用します。メッセージの右上にある ▼ ボタンをクリックしてメニューを開き、「Private reply」をクリックしてください。

[*6] 文章ではない部分をクリックすると、先ほど説明したようにインラインリプライではなく通常のリプライになります

プライベートリプライ作成直後は参加者が自分一人です。個人的なメモでないのなら、参加者リストにある追加ボタン をクリックして他の参加者を追加する必要があります。

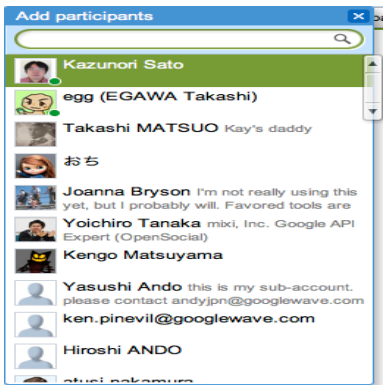
プライベートリプライは wave の構成要素の中でも非常に特殊です。プライベートリプライはむしろ新しい wave を作成して、ある wave に挿入したものと考えた方がいいかもしれません。プライベートリプライの参加者は元メッセージを含む wave とは完全に独立していて wave に参加していない人を追加することさえできます。また、wave を完全に削除することができないように、プライベートリプライもいったん作成すると削除できません。

3.3.4.9　メッセージを削除する

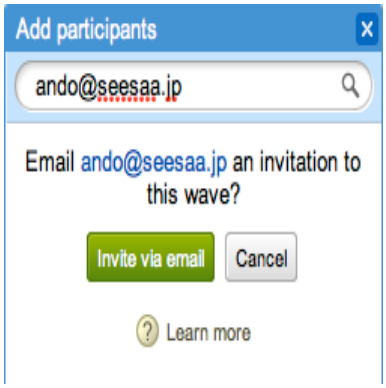
メッセージを削除するには、メッセージの右上にある ボタンをクリックしてメニューを開き、「Delete message」をクリックしてください。ただし、ルートメッセージ（wave 作成と同時に作成されたメッセージ）とプライベートリプライは削除できません。

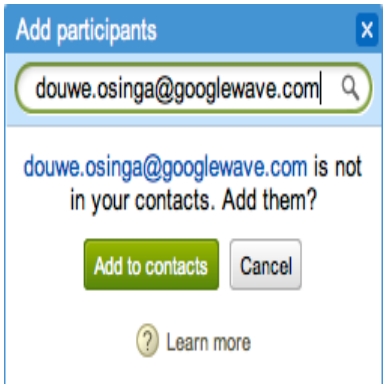
3.3.4.10　参加者を追加する

wave に参加者を追加するには、参加者リストにある追加ボタン をクリックして参加者追加ダイアログを開き、メールアドレスを入力して確定するか、コンタクトリストのリアルタイム検索結果が下に表示されるのでその中から選択します。



なお、アドレス入力したユーザーが wave を利用していない場合はその場で招待状の送付ができますし、wave を利用していても自分のコンタクトリストに登録されていない場合はその場で登録を促されます。

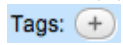




3.3.4.11 ロボットを追加する

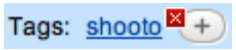
ロボットは wave の参加者のように振舞うことのできるサーバーサイドプログラムのことで、人工無能ロボットや日英翻訳ロボットなどいろいろなロボットが存在します。ただし、wave から見ると普通の参加者もロボットもまったく同じですので、追加する方法は前節の参加者の追加と同じです。ロボットについての詳細は 4.3 節を参照してください。

3.3.4.12 wave にタグを付ける

タグは全員で共有される wave の属性のようなものです。フッターにあるタグ追加ボタン 

をクリックしてください。

ダイアログが開き wave をタグ付けすることができます。設定したタグはリンクになっていて、クリックすると検索ペインに同じタグを持つ wave の一覧が表示されます。設定したタグを削除するには、タグ上にマウスを移動したときにタグの右上に表示される削除ボタンをクリックしてください。



3.3.4.13 誰でもアクセス可能な wave を作成する

先に見たように wave には好きなユーザーを参加者として追加できます。しかし一人一人明示的に参加者として追加するのではなく、誰でもアクセスできる掲示板のような wave が必要な場合もあるでしょう。そのようなときには通常の参加者を追加する手順で「public@a.gwave.com」を追加します。

ツールバーの下に"gave everyone access."と表示されていることが分かるでしょう。これで誰からでもアクセスできるようになっています。

パブリックな wave は自分から何か発言するか、明示的に follow しない限り Inbox には現れませんが、中には非常に有益な情報を持つものもたくさんあります。wave について詳しく知りたいことがあれば、次のようにしてパブリック wave を検索するといいでしょう。

with:public [気になるワード]

3.3.4.14　ショートカットを利用して効率よくメッセージを編集する

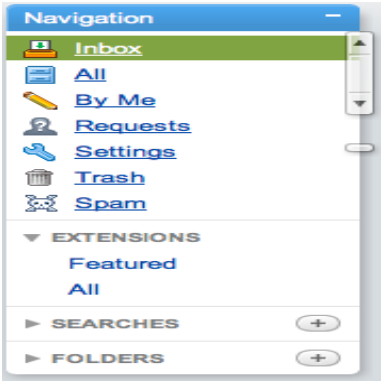
エディットモードでは次のようなショートカットが利用できます（一部ビューモードで使用するものも含まれます）。

（ビューモードで）Enter キー	リプライを追加する。テキストを選択してた場合は
（ビューモードで）Shift キー＋ Enter キー	同じインデントレベルのスレッドの一番最後にリプ
（ビューモードで）Crtl キー＋ E	メッセージを編集する
Shift キー＋ Enter キー	編集を完了する
Ctrl キー＋ B	選択した文字を太字にする／太字を解除する
Ctrl キー＋ I	選択した文字を斜体にする／斜体を解除する
Ctrl キー＋ K	リンクを追加する
Ctrl キー＋数字キー	数字キーが 1 から 4 のときは、その行をヘッダーに
Ctrl キー＋ 5	リストにする
Ctrl キー＋ 6	ヘッダ・リストなどのフォーマットを解除する
Ctrl キー＋ C	選択したテキストをコピーする
Ctrl キー＋ X	選択したテキストを切り取る
Ctrl キー＋ V	コピーまたは切り取ったテキストを貼り付ける

表 3.10　エディットモード wave ショートカットキー

3.3.5　フォルダ管理（ナビゲーションペイン）

wave をフォルダで管理するにはナビゲーションペインを使用します。



3.3.5.1 デフォルトのフォルダを理解する

ナビゲーションペインはデフォルトフォルダとして次のものを持ちます。

フォルダ名	説明	検索条件
Inbox	自分宛の wave が最初に入るフォルダ	in:Inbox
All	自分宛の全 wave が入るフォルダ	-
By Me	自分が発言した wave が入るフォルダ	by:me
Requests	Google Wave 以外の Wave ユーザーからのフレンドリクエストや、メッセージ送信許可のリクエストが入るフォルダ。ただし、現在のところ Google Wave 以外の Wave サーバーは実質存在しないので使用されません	in:Requests
Settings	Google Wave の設定に使用できる wave があるフォルダ	with:settie
Trash	ゴミ箱	in:Trash
Spam	スパム用のフォルダ	in:Spam

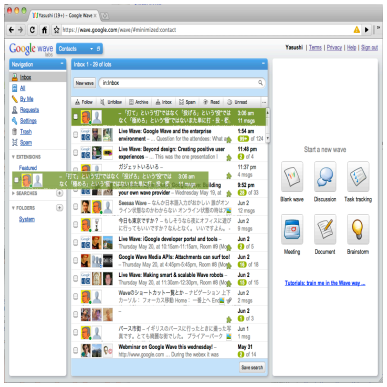
フォルダの意味はフォルダ名から予想できると思いますが、ここでの注目は一番右のカラムです。

実は wave のフォルダは All フォルダを除く全てが特定の検索条件とひも付いています。ナビゲーションペインでフォルダを選択すると、検索ペインにそのフォルダに属する wave の一覧が表示されると共に、検索ボックスに表に書いた検索条件が表示されているでしょう。従って正確に書けば、上記のうち検索条件が「in:フォルダ名」で表されているものだけがフォルダで、By Me や Settings などはある検索条件へのショートカットになっています。

3.3.5.2 wave をフォルダに保存する

wave をフォルダに移動するには検索ペインや wave のツールバーにある Move to ボタン 

が利用できるほか、検索ペインから wave をドラッグしてナビゲーションペインのフォルダにドロップすることでも可能です。



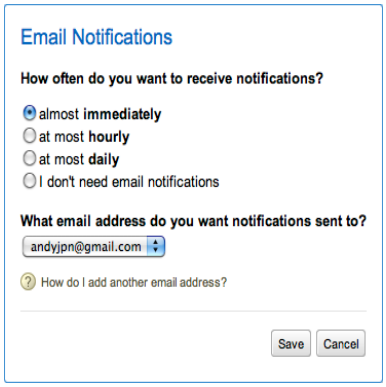
なお、このとき wave をドロップできるのはフォルダだけ、つまり前節で言えば検索条件が「in:フォルダ名」になっているものだけで、「By Me」や「Settings」にドロップしても無視されます。

3.3.5.3　フォルダの変更を通知する

現時点では Google Wave は E メールを置き換えるには至っておらず、Google Wave ユーザーであってもやはり Wave クライアントよりもメールクライアントを立ち上げている時間の方が長いと言うのが正直なところでしょう。Google Wave では Inbox フォルダに限り、内容に変更があったときに E メールで通知を受けることができます。

フォルダをクリックすると右側にメニューボタン  が現れますので、Inbox をクリックして「Notifications」というアイテムを選択してください。

すると次のようなダイアログが開きます。



通知を受ける頻度と、メールアドレスを選択し、「Save」ボタンをクリックすると Inbox 内の wave が変更されたときに指定された頻度でメール通知が届きます。

3.3.5.4 フォルダを作成する

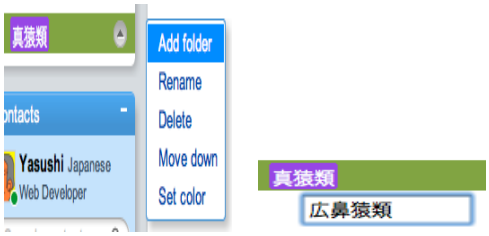
フォルダを作成するにはナビゲーションペインの FOLDERS ラベルの横にある Add ボタン  をクリックします。テキストエリアが表示されるので、好きなフォルダ名を入力して Enter キーを押してください。フォルダが作成されます。

デフォルトのフォルダと異なり、ユーザーが作成したフォルダにはランダムな背景色が設定されます。さっそく作成したフォルダに wave を追加してみましょう。自分で作成したフォルダに wave を追加する方法はデフォルトフォルダと全く同じです。

このように、ユーザーが作成したフォルダに属する wave が検索ペインに表示されるときにはフォルダと同じ色のラベルが設定されます。

3.3.5.5 サブフォルダを作成する

フォルダは wave だけではなくフォルダを持つこともできます。フォルダメニューを表示して「Add Folder」を選択しましょう。



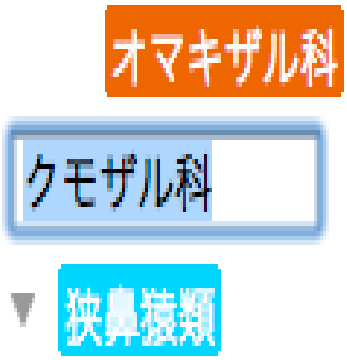
テキストフィールドが親フォルダの下に挿入され、フォルダ名を入力して Enter キーを押せばサブフォルダの完成です。もちろんフォルダの階層はいくらでも深くできます。

ただし、サブフォルダに属する wave はサブフォルダだけに所属していて、親フォルダとは何の関係もありません。つまり親フォルダ名をクリックして検索ペインに表示される wave は直接そのフォルダに所属しているものだけで、子フォルダ内の wave は表示されません。フォルダの親子関係は単にフォルダの間の関係で、所属する wave には関係がないことに注意してください。

3.3.5.6　フォルダをリネームする

フォルダメニューの「Rename」を選択すると、既存のフォルダの名前を変更することができます。

フォルダ名がテキストフィールドに変わります。名前を変更したら Enter キーを押して確定してください。



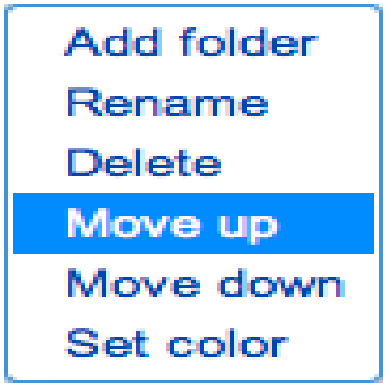
3.3.5.7 フォルダを削除する

フォルダを削除するにはフォルダメニューから「Delete」を選択します。確認画面などは特に表示されず、クリックすると即座にフォルダが削除されてしまうので注意してください。なお、消したフォルダに入っていた wave は消されるわけではありませんが、All フォルダでしか見つけられなくなります。

なお、「Delete」メニューアイテムはデフォルトフォルダや、サブフォルダを持つフォルダには表示されません。

3.3.5.8 フォルダを並び替える

フォルダが並び替え可能な場所にある時はフォルダメニューに「Move up」や「Move down」が表示されます。少し面倒ですがこれらのメニューアイテムを利用してフォルダを並び替えることができます。

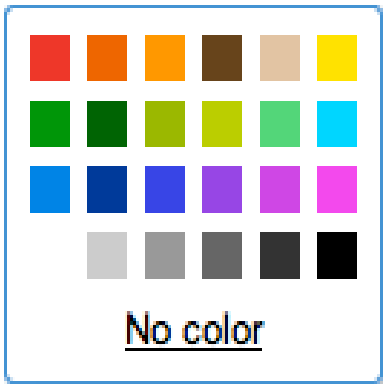


なお、フォルダの並び替えが可能なのは同一階層の間だけで、親子関係を変更することはできないようです。

3.3.5.9　フォルダの背景色を変える

階層構造を作って管理できるとは言え、フォルダの数が多くなるとそれだけでは一覧性に難があります。Google Wave ではフォルダ名に背景色を設定することで区別をつけやすくできます。フォルダメニューの「Set color」を選択してください。

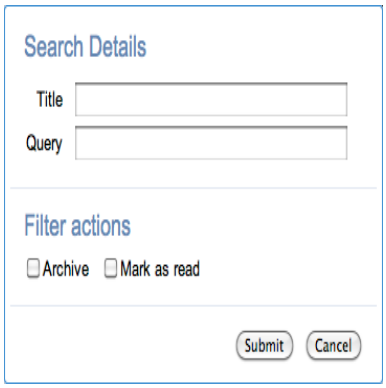
次のようなダイアログが開きます。



好きな色を選択すると、選択した色がフォルダの背景色に使用されます。なお、背景色として「No color」を選択すると、フォルダの背景色がなくなるだけではなく、検索ペインの wave に表示されていたフォルダ名も表示されなくなります。逆に、デフォルトフォルダに背景色を設定するとデフォルトフォルダ名が検索ペインの wave に表示されます。

3.3.5.10 検索条件を保存する

ナビゲーションペインではフォルダの他にも検索条件に従って wave を分類することができます。SEARCHES ラベルの横にある Add ボタン  をクリックしてください。次の様なダイアログが表示されます。



表示されるダイアログは検索ペインで検索条件を保存したとき（50 ページ参照）と同じですが、今回は検索条件（Query）が空白になっています。ここでの手順で検索条件を保存するときには、残念ながら実際に動作させるまで結果が確認できません。そのため検索条件を保存するには検索ペインのフッターにある「Save search」ボタンを使用した方がいいでしょう。

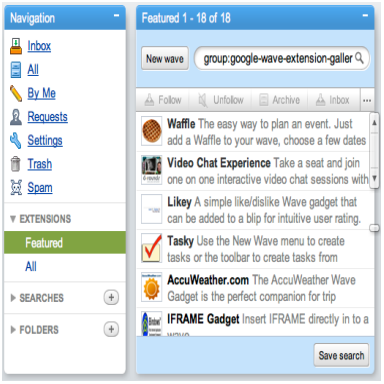
3.3.5.11 検索条件を変更する

登録した検索条件の結果が期待と異なったときには、検索メニューからで Edit を選択して検索条件を修正することができます。

なお、上記の図の通り検索条件もフォルダと同じくリネーム・削除・並び替え・背景色設定が可能です。実際の手順についてはフォルダの場合と全く同じなので省略します。

3.3.5.12 便利なガジェットやロボットを捜す

ナビゲーションペインの EXTENSIONS には Google が選んだ便利な拡張が揃っています。



Featured にある拡張が Google おすすめのもの、All にあるものが Featured にあるものも含む Google が受け付けた全ての拡張です。Featured または All フォルダを開き、検索ペインのダイジェストを読んで拡張を選ぶと、次のような拡張インストーラーが開きます。

自分の Google Wave でいつでも使えるようにインストールしたいと思ったら「Install」ボタンを、試しに一度使ってみたいと思ったら「Try It Now!」リンク[*7]をクリックしましょう。インストーラーの詳細については 4.4 節を参照してください。

ここに登録されている拡張の多くはユーザーによって登録されたもので[*8]、これからも少しずつ増えていきます。また、拡張はここにあるものが全てではなく、Wave API のサンプルギャラリー[*9]にもたくさんの拡張が登録されていますし、Google Code などで控えめに公開している人もいるでしょう。wave になにか機能が足りないと感じたときには積極的に使える拡張を探しに行くクセをつけておくと wave がどんどん便利なものになるに違いありません。

以下に、執筆時点で Featured に登録されている拡張を簡単に紹介しておきます。

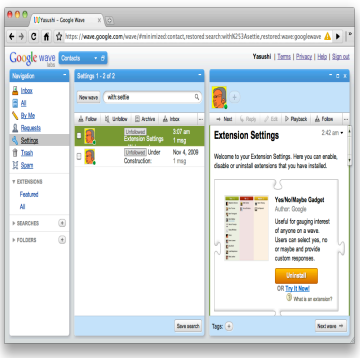
3.3.5.13　インストールした拡張を管理する

一度インストールした拡張は「Settings」フォルダの中にある「Extension Settings」というタイトルの wave で管理することができます。

[*7] 「Try It Now!」リンクが用意されていない拡張もあります
[*8] 登録方法については 4.4.8 節を参照してください
[*9] http://wave-samples-gallery.appspot.com/

AccuWeather	場所と日時を選ぶと天気を教えてくれるガジェット
Colcrop Game	二人で遊べる陣地取りゲームガジェット
ConceptDraw MindWave	参加者で同時にマインドマップを編集できる拡張
Ferry	wave の内容を Google Docs に書き出すことができる拡張
Likey Gadget	簡単に Live/Dislike を投票できるガジェット
Iframe Gadget	iframe を wave に貼り付けることができるガジェット。例えば Google Docs を簡単に wave に貼り付けられる
Image Gadget	Web 上の画像を自由にリサイズして張り付けアノテーションを設定できるガジェット
Map Gadget	Google Map ガジェット
Mind Map Gadget	参加者で同時にマインドマップを編集できる拡張
Napkin Gadget	複数人で自由に描画できるキャンバスガジェット
Phone Conference	電話会議ガジェット
Pollo Gadget	投票ガジェット
Take-Out Gadget	みんなで出前などの注文を取るときに便利そうなガジェット
Tasky	Wave 上でタスク管理するための拡張
Video Chat Experience	ビデオチャット
Waffle (Date-Picker Gadget)	イベントの日取りを決めるのに使用できるガジェット
Wave Sudoku	多人数で同時に数独を楽しめるガジェット。デモ向き
Yes/No/Maybe Gadget	参加者が Yes・No・Maybe から一つを選び、それを一覧できるガジェット

表 3.11 フィーチャーされた拡張一覧



なお、「Settings」フォルダは Google Wave の設定に関する wave をまとめるフォルダのようですが、執筆時点では拡張の再インストール／アンインストールしかできないようです。

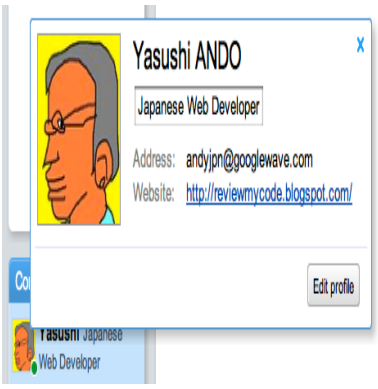
3.3.6　コンタクト管理（コンタクトペイン）

コンタクトペインは友人の wave アドレスや自分のプロフィールを管理するためのペインです。

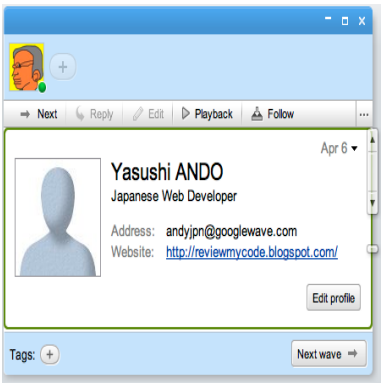
1. **タイトル**　Contacts
2. **ツールバー**　ログインユーザーのプロフィールや検索フィールドがあります
3. **コンタクトリスト**　全コンタクトリストまたは検索されたコンタクトリストが表示されます
4. **フッター**　コンタクト管理画面へのリンクとコンタクト追加ボタンがあります

3.3.6.1　プロフィールを変更する

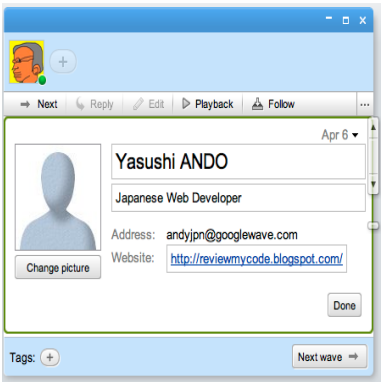
Google Wave に参加した人がまずやりたいことはプロフィール画像をオリジナルなものに変更することではないでしょうか。それにはまず、コンタクトペイン上部の自分のプロフィールをクリックしてプロフィールダイアログを開きます



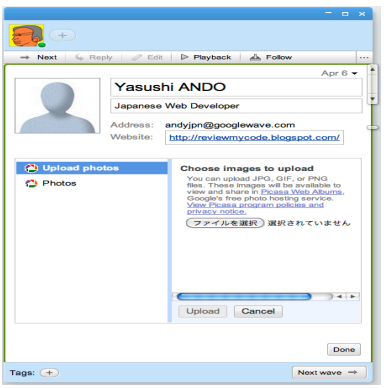
コンタクトペインで名前の直後に表示されていたステータスはこのダイアログで直接編集できます。このステータステキストは友人のコンタクトペインにも表示されるので、自分の状況に応じてときどき書き換えてみるのもおもしろいでしょう。プロフィールを変更するには、ここからさらに「Edit profile」ボタン Edit profile をクリックしましょう。プロフィール設定用の wave が開きます。



さらに「Edit profile」ボタンをクリックすると名前・ステータス・ウェブサイトが編集可能になります。



プロフィール画像を変更するにはさらに「Change picture」をクリックして、Picasa Web Album からプロフィール画像を選択してください。



3.3.6.2 友人と連絡を取る

友人と連絡を取るにはツールバーの検索フィールドを使ってアカウントを検索し、コンタクトリストから友人を選択してください。検索フィールドは名前または wave アドレスを対象に前方一致検索します。

ダイアログのボタンはそれぞれ次のような意味を持ちます。

New wave　そのユーザーと自分だけが参加する新しい wave を作成します
Ping　そのユーザーに ping[*10] を送ります
Recent wave　検索ペインにそのユーザーと自分を含む wave 一覧を表示します

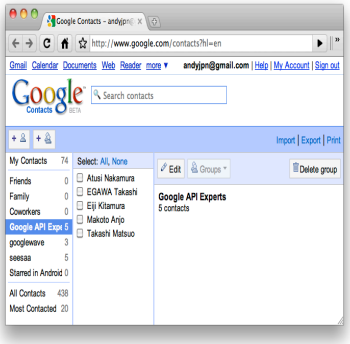
3.3.6.3　コンタクトを追加する

コンタクトペインからコンタクトを追加するにはフッターのコンタクト追加ボタン  をクリックして、wave アドレスを追加してください。

3.3.6.4　コンタクトを管理する

フッターの「Manage contacts」リンクをクリックすると Google Contacts が開きます。

[*10] 自分とそのユーザーだけが参加する wave を作成し、さらにそのユーザーが wave にログイン中の場合はポップアップして会話の開始を促します



Google Contacts ではコンタクトのグルーピングや削除が可能です。

3.4 Google Wave を Google Apps で利用する

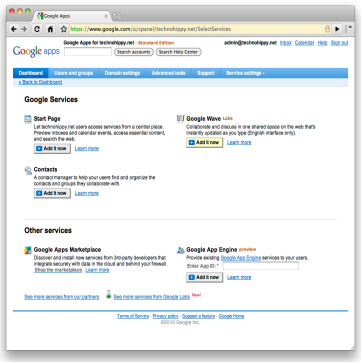
Google Apps は簡単に言えば Google 製のグループウェアで、Google Docs や GMail などのプロダクトを特定のグループ内で使用することができるようにするものです。Google Apps を使用していれば、ここで説明する手順で Google Wave サービスを有効にすることで、メンバーに Google Wave の機能を提供することができます[*11]。

Google Wave は現在のところ拡張版の Google Apps だけで有効にできます。まず Google Apps のダッシュボードから「ドメインの設定」を開き次のように設定しましょう。

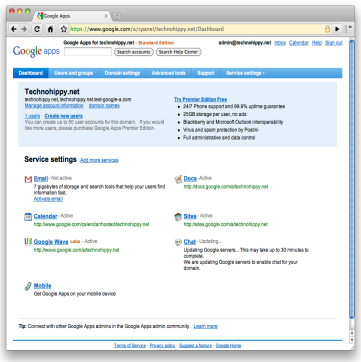
言語 「Englishii（US）」
新しいサービスとベータ版の機能 「ベータ版の機能を有効にする」にチェック
コントロールパネル 「拡張版（アメリカ英語のみ）」

上記のように設定し「変更を保存」ボタンをクリックすると Google Apps の表示が英語になります。再び Dashboard に戻り、Service Settings ヘッダの横の「Add more services」リンクをクリックしましょう。

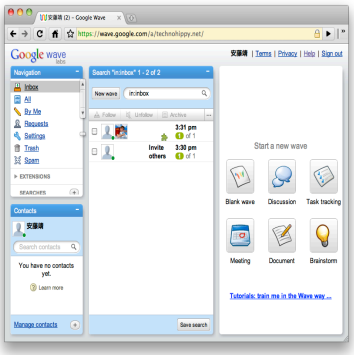
[*11] 設定次第では Apps の Google Wave ユーザーと通常の Google Wave ユーザーの間でやり取りすることも可能です



追加できるサービスの一覧に「Google Wave」があるはずです。「Add it now」ボタンをクリックすると確認画面に遷移します。「Yes, enable Google Wave」ボタンをクリックしてください。



Dashboard に戻ると利用しているサービスとして Google Wave が追加されていることが分かるでしょう。最後に、Apps のコンソールが英語版のままだと少し使い勝手が悪いので「Domain settings」で Language を日本語に戻し、Control panel も Current version にしておきます。設定を元に戻しても Google Wave は有効なままなので安心してください。



ダッシュボードから Google Wave を開くと、Apps サービスとして Google Wave が動

作していることが分かります。Google Wave を使った密度の高いコミュニケーションは、Apps のようにある程度限られたグループで行う方が相応しいかもしれません。Google Apps 管理者の方は Google Wave を有効にしてメンバー同士の一歩進んだコミュニケーションを試してみてはどうでしょう。また管理権限はないけれど、会社で Google Apps を使用している方は管理者の方に Google Wave を有効にするよう頼んでみるのもいいかもしれません。

第 4 章

プラットフォーム

4.1 はじめに

ここでは Wave を構成する 3 つの P の 2 つ目、プラットフォームとしての Wave について説明します。なお、ここで言う「プラットフォーム」とは「ある種のソフトウェアが動作する基盤」のことです。つまり「Wave はプラットフォームである」と言えば、Wave という基盤の上でなにかアプリケーションを動作させることができるという意味になります。端的に言えば、Wave は API が公開されていてその機能が拡張可能であるということです。なお、Wave には Wave を拡張するためではなく単に Wave の機能を外部サイトから利用するための API もあります。それらについても本章で紹介します。

それぞれの API の詳細に移る前に本節ではまず各 API の概要とそれらの関係について説明します。

4.1.1 API 概要

Wave の提供する API は大きく分けると次の 4 種類です。

- エクステンション
  - ガジェット API
  - ロボット API
- データ API
- エンベッド API

ガジェット API とロボット API は共に Wave を拡張（Extend）するための API ということで、あわせてエクステンション（Extension）と呼ばれます。Wave をプラットフォームと呼ぶときの対象は厳密に言えばこの 2 つの API だけかもしれません。残る 2 つの API については Wave を拡張するわけではなく Wave を外部のサイトから利用するための API です。まずはそれぞれの API と Wave との関係を理解するために次の図を見てください。

- **ガジェット API** を使用すると、特殊な機能を持った表示要素を Wave に貼り付けることができます。ガジェットで扱うことができるデータは基本的に Wave からは独立していて、ユーザーの発言などをガジェットで書き換えることはできません。ただし、ガジェットが使用しているデータも Wave の一部ですから、参加者の間でリアルタイムに同期されます。ガジェットの利用例としては Google Maps を Wave に貼り付けるための Maps ガジェットや、参加者同士でチェスゲームを楽しむためのチェスガジェットなどがあります。
- **ロボット API** を使用すると、Wave の中でまるで普通の参加者のように振舞うプログラムを作成できます。もう少し具体的書くと「参加者が追加された」「Wave の内容が書き換えられた」などの Wave 内で発生するイベントに反応して、挨拶をしたり、スペルミスをチェックするプログラムを作成できます。利用例としては例えば参加者と会話する人工無能ロボットや、参加者の発言をすべて関西弁に書き換えてしまう関西弁ロボットなどいろいろなものが考えられるでしょう。さらに、ロボットは Wave にガジェットを貼りつけたり、自分が貼り付けたガジェットのデータにアクセスすることもできるので、ガジェットとロボットの両方を使用したエクステンションを作ることも可能です。
- **データ API** を使用すると、ユーザーの代理として Wave のデータを読み書きするプログラムを作成できます。ユーザーの権限は OAuth を使用してプログラムに委譲されます。
- **エンベッド API** を使用すると、特定の Wave を自分のブログに貼りつけることができます。ただし、自分のサイトと Wave との間の情報のやりとりは大幅に制限されています。

4.1.2　API 比較

Wave を使った面白いプログラムのアイデアが浮かんでも、慣れないうちはガジェット API とロボット API のどちらを使えばいいのか戸惑うことがあるかもしれません。いずれの API を使うべきなのか、または組み合わせて使用するべきなのか、判断の参考になることを期待して、それぞれの API の対比を表 4.1 にまとめました。

これらを踏まえて、やりたい事に対してどの API を使えばいいかを簡単にまとめると次のようになります。

	ロボット API	ガジェット API	データ API	エンベッド API
拡張するもの	Wave の機能を拡張	Wave の機能を拡張	Wave の機能を外部サービスで利用	Wave の機能を外部サービスで利用
動作する場所	サーバーサイド	クライアントサイド	サーバーサイド（Wave 外）	クライアントサイド（Wave 外）
できること	参加者とほぼ同じ	参加者情報取得・ガジェット固有の共有状態を操作	Wave を検索・読取	Wave を表示
利用可能な言語	任意（公式ライブラリは Java・Python のみ）	JavaScript・Flash	任意	JavaScript

表 4.1 Wave API 比較

- ユーザーの発言を読み取って返答したり、ユーザーの発言を書き換えるなど、Wave 内のコンテンツに対してなにか処理をするには ロボット API を使用
- Wave に特殊なユーザーインターフェースを提供するには ガジェット API を使用
- Wave に特殊なユーザーインターフェースを提供しつつ、Wave 内のコンテンツにたいして処理をするには、ロボット API とガジェット API を組み合わせて使用
- ユーザーの代わりに Wave のデータにアクセスするウェブサービスを作成するには データ API を使用
- 特定の Wave を他のサイトに埋め込みむには エンベッド API を使用

本格的な Wave エクステンションを作成するには、ほとんどの場合でロボット API とガジェット API を組み合わせて使用することになるでしょう。

4.1.3 Wave らしいエクステンション

[*1]

Wave は参加者のあらゆる操作が逐一共有され、その極端なリアルタイム性でこれまでにない使用感をユーザーに与えます。ユーザーの無用な混乱を招かないようにするため、あなたの作るエクステンションの UI も Wave らしさを保ったものにしておきましょう。以下に現在「Wave らしい」といわれるエクステンションの特徴について述べます。ただし、Wave はまだ始まったばかりのサービスです。「Wave らしさ」の定義もこれから変わっていくかもしれません。あまり杓子定規には捉えず、好奇心をもって自分でいろいろなエクステンションを試しながら自分なりの「Wave らしさ」をエクステンションで表現すればよいのではないかと思います。

4.1.3.1 Wave らしいガジェット

Wave らしいガジェットとはユーザーの操作を共有状態を利用してリアルタイムに共有していくものです。ガジェットの中身だけは明示的にリロードボタンを押さなければ更新されない、などという UI が好ましいものではないことは明らかでしょう。また、ガ

[*1] 本節の内容は次のドキュメントによっています: http://code.google.com/intl/ja/apis/wave/extensions/designprinciples.html

ジェットは多人数で編集されるものでもありますから、Wave を見ている人ごとに表示を変えることも重要です。wave の文章を複数で編集しているときには自分以外の参加者のカーソル位置に参加者の名前が表示されるようになっているでしょう。同様にガジェットの中でも現在のビューアーが編集しようとしている場所とその他の参加者が作業中の場所の区別が付くようにしておくことが推奨されます。

4.1.3.2　Wave らしいロボット

Wave のロボットの重要な点は、参加者のようにふるまうことを期待されていると言うことです。実際、wave 内で通常の参加者とロボットとを一見して区別する方法はありません。現時点では Wave アドレスを表示してドメインが appspot.com または googlewaverobots.com であればロボットだということが分かりますが、将来に渡って同じ方法で確認できる保証があるわけではありません。

人間のようにふるまうと言う理想から言えば、例えばロボットに何か処理を実行させるための特別な文法を用意して、会話の中でその特殊なコマンドが出てきたら反応する、というようなロボットはあまり好ましくありません。例えパースが多少煩雑になったとしても普通の文章をうまく解釈して動作するのがりそうです。例えば map:[address] というキーワードに反応して地図を挿入するロボットよりは、参加者の発言を解析して住所と解釈できる部分を発言内に見つけたときに自動的に地図を挿入するロボットの方が Wave らしいと言えます。

また、ロボットは他のロボットと同じ Wave に参加した時でも正しく動作しなければいけません。「ロボット A はロボット B と一緒に使わないで下さい」という注意が必要なロボットはよいロボットとは言えないでしょう。むしろ他のロボットと組み合わせて使うとより便利になるロボットはいいロボットと言えます。例えば、プログラムのソースコードをハイライトする Syntaxy というロボットと、Wave に書かれたプログラムを実行する Monty というロボットは、組み合わせて使うことでお互いに一層便利になります。

Wave サンドボックス

API の説明に入る前に、API のデバッグに便利な Wave サンドボックスにを紹介します。Wave サンドボックスは開発者向けのもうひとつの Google Wave で、まだ通常の Google Wave には入っていない実験的な機能を試すことができたり、エクステンションを作成するときに役に立つデバッグメニューがあったりと、開発者には欠かせないものになっています。エクステンションを作成を始める前に次の手順に従ってサンドボックスのアカウントを手に入れておくといいでしょう。

Wave サンドボックスアカウントを取得する

Wave サンドボックスのアカウントは、フォームに必要事項を入力して Google にリクエストを送ることで取得できます。API リファレンスページ[*2]の左サイドメニューの一番上にある「Request Sandbox Access」をクリックして、フォームに希望するユーザー

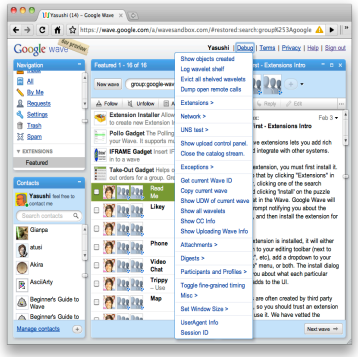
[*2] http://code.google.com/intl/ja/apis/wave/guide.html

名などを入力しましょう。

30 分以内にメールが届くというメッセージが表示されますが、実際には数分もかからず「WaveSandbox.com Account」というタイトルのメールが届くはずです。メールに書かれているログイン URL にアクセスし、これもメールに書かれているアカウント名とパスワードで Wave サンドボックスにログインしてみましょう。左上のロゴが次のようになっていればサンドボックスにログインできています。

デバッグメニュー

サンドボックスにログインすると右上に見慣れない「Debug」というリンクが追加されていることに気づくでしょう。このデバッグメニューがサンドボックスのサンドボックスたる所以です。メニューは Google Wave の開発自体にも使用されているため、エクステンションの開発者にはあまり関係がないと思われるものも多くあります。デバッグメニューでは特に次の項目がエクステンションのデバッグの役に立つはずです。

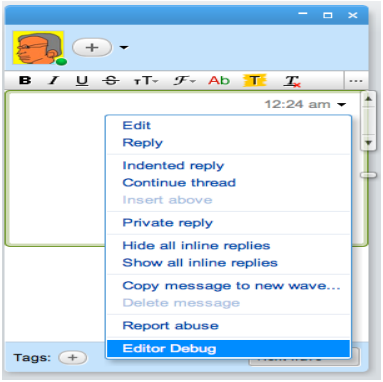


Extensions > サンプルガジェットや任意のガジェットを追加したり、エクステンションインストーラを追加できます

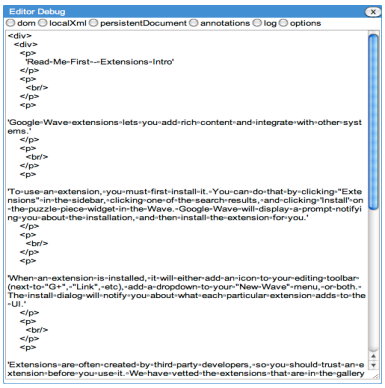
Get current Wave ID 現在表示している Wave の ID を表示します

Show UDW of current Wave Wave の未読・既読、登録されているフォルダなど特定のユーザーに関連する情報を表示します

また、サンドボックスでは画面上部のメニューだけではなく、wave のメニューに「Editor Debug」という項目が追加されています。



このメニューをクリックすると次のようなダイアログが開き、XML ドキュメントやアノテーションなど、現在の wave エディタの状態がリアルタイムに確認できます。エクステンションの開発では画面上部の Debug メニュー以上に有用ですので積極的に活用するといいでしょう。



4.2　ガジェット API

4.2.1　はじめに

ガジェットとは誤解を恐れずにいうと、Wave に埋め込めるブログパーツのようなものです。どのようなものなのかは実際に Wave でガジェット使用している例を見るのが一番わかりやすいでしょう。

wave の中にチェス盤が埋め込まれている様子が見て取れます。このチェス盤がガジェットによって表示されているものです。もちろんこのチェス盤はただ画像として表示されているのではなく、実際に参加者が交互にコマを動かしあってゲームを楽しむことができます。このように、ガジェットを利用すれば単なるテキストや画像のやりとりを超えた、より複雑なコミュニケーションを wave 上で行うことができます。

一般的なガジェットの用途としては次のようなものが考えられるでしょう。

- ボードゲーム（チェス・3 目並べ）
- 特殊な情報共有（投票・ホワイトボード）
- 外部サイトの利用（RSS 取得・Twitter 投稿）

なお、ガジェットは iframe 内で Wave とは別のドメインで実行されるため、wave のコンテンツへのアクセスが大幅に制限されます。具体的には参加者情報を読み取ることは可能ですが、wave 内の文章にアクセスすることは許されていません。ガジェットで必要な情報は基本的には共有状態（102 ページ参照）を利用してガジェットの中で管理し、wave のコンテンツ情報を必要とするガジェットは単体では実現できないのでロボットと組み合わせて実現する（162 ページ参照）ことになります。

4.2.2 はじめてのガジェット

wave ガジェットを作成するのに特殊な知識は必要ありません。ガジェットは HTML や CSS、JavaScript など Web 系の開発者であればおなじみの要素が埋め込まれた XML ファイルに過ぎないのです。例えば、wave 内に「Hello, Wave Gadget!」と表示するだけの単純なガジェットのソースは次のようになります。

http://wave-book.appspot.com/assets/gadget1.xml

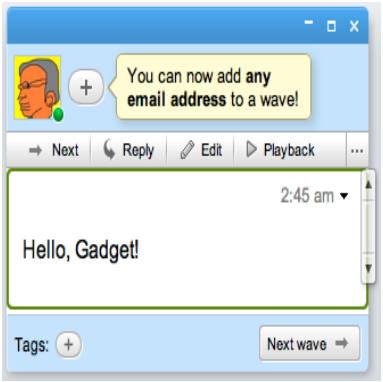
```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <Module>
3    <ModulePrefs title="My First Gadget" />
4    <Content>
5      <![CDATA[
6        Hello, Gadget!
7      ]]>
8    </Content>
```

```
9  </Module>
```

ご覧の通りルート要素として<Module>要素を持ち、さらにその子要素として<ModulePrefs>要素と<Content>要素がある非常にシンプルな XML です。<Content>要素は実際に表示される内容"Hello, Gadget!"を保持しています。なお、<ModulePrefs>要素には title 属性として"My First Gadget"が設定されていますが、これは wave ガジェットでは使用されません。ただしガジェットとしては ModulePref 要素・title 属性ともに設定必須のようです。

このガジェットを使用するには、XML ファイルを Wave からアクセス可能な場所に格納する必要があります。サーバーが Google AppEngine 限定されているロボットとは違い、ガジェット XML はグローバルな URL を持ちさえすれば、どのようなサーバー上においても構いません。アップロードしたら第 3.3.4.7 節を参考に URL を指定して wave にガジェットを貼りつけてみましょう。

上記のガジェットを実際に wave に埋め込んだ様子が以下です。



ガジェットに設定した通り、wave に「Hello, Gadget!」と表示されています。これは wave のコンテンツではないので、編集モードでも文章を変更することはできません。固定された文字列を表示する他にはなんの機能も持ちませんが、指定した URL を元に先ほど作成した XML ファイルを読み込んで、Wave がレンダリングしたれっきとしたガジェットです。wave ガジェットを作るのになにか特別な知識が必要なわけではないことが、わかっていただけたのではないでしょうか。

4.2.3　Gadgets API

実は先ほどの Hello World ガジェットは Wave 以外のサービス、例えば iGoogle や OpenSocial コンテナ（mixi, goo ホーム他）など様々な場所で動作します。というのも Wave Gadgets API は Gadgets API の拡張になっていて、Gadgets API 自体は Wave に限らず多くのサービスで採用されているものだからです。wave ガジェットについて説明する前に、この節では Wave ガジェットの API の基礎となっている Gadgets API について簡単に説明します。

2010 年 6 月現在の公式ドキュメント[*3]によると Gadgets API が利用可能なサイトには次のようなものがあります。

[*3] http://code.google.com/intl/en/apis/gadgets/docs/overview.html

Google 製プロダクト iGoogle, Google Apps start pages, Google Toolbar, Orkut, Blogger, Google Calendar, GMail, Google Sites

サードパーティ製プロダクトまたはサイト MyAOL, IBM websphere portal, RedHat JBoss portal, SUN portal, BEA weblogic portal

その他 Any webpage

また、Wave と同様に拡張されたガジェットが利用できるサービスとして、OpenSocial、Friend Connect、Maps、Finance API (Labs)、Calendar (Labs)、Spreadsheets (Labs) などがあります。なお OpenSocial ガジェット API については将来的には Wave でも利用可能になる予定です。

4.2.4 Gadgets API の基本

ここから先ほどのガジェットの XML ファイルの内容を詳細に見ていきましょう。なお、先に書いたように Wave ガジェットは Gadgets API の拡張ですから、ここでの説明はもちろん Wave ガジェットでも有効です。

4.2.4.1 Gadgets XML

Hello World ガジェットのソースをもう一度見てください。

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <Module>
3    <ModulePrefs title="My First Gadget" />
4    <Content>
5      <![CDATA[
6        Hello, Gadget!
7      ]]>
8    </Content>
9  </Module>
```

この XML はガジェットとして最小構成のものです。以下でそれぞれの要素の詳細を説明します。ここに出て来ていない要素については公式ドキュメント[*4]を参照してください。

<Module>タグ

ガジェット XML のルート要素です。ガジェットに関する情報はすべてこの子要素内に記述されます。

<ModulePrefs>タグ

タイトル、説明、作者など、ガジェットのメタ情報を記述します。ガジェットの仕様ではメタ情報を指定するための属性はすべて省略可能なはずですが、wave ガジェットでは

[*4] http://code.google.com/intl/en/apis/gadgets/

title 属性だけが必須になっているようです[*5]。<ModulePrefs>タグに設定できる属性には次のようなものがあります。

title　ガジェットのタイトルです。wave ではガジェットは本文の一部として扱われるためタイトルは表示されません

description　ガジェットの説明です

author　ガジェットの作者です

height　ガジェットの高さをピクセル単位で指定します。必須項目ではありませんが、無指定だと無駄な空白ができたりガジェットの一部が表示されなかったりする場合があります。事前にガジェットの高さが確定している場合は面倒がらずに記述しておきましょう。事前に高さが決定できない場合は dynamic-height フィーチャー（114 ページ参照）が利用できます

＜Content＞タグ

ガジェットの本体に当たる部分を記述します。本文そのものを書くこともできますし、type 属性を url と指定することで外部サイトのコンテンツを組み込むこともできます。<Content>タグに設定できる属性には次のようなものがあります。

type　ガジェットのコンテンツのタイプで、html または url が指定できます。html を指定した場合は<Content>タグの子要素の CDATA セクションで直接 HTML, JS, CSS などを記述します。url を指定した場合は href 属性で取り込む外部サイトの URL を指定します。url を指定すると<Content>タグの子要素は無視されます。また、省略した場合は html が指定されたものとみなされます。

href　type 属性の値として url を指定した場合に必須です。ガジェットとして取り込む URL を指定します。

CDATA セクション

Content タグの type 属性の値として html を指定した（もしくは何も指定しなかった）場合に自由に HTML や JavaScirpt、CSS などを記述します。詳細については次節以降を参照してください。なお XML の規約上 CDATA セクション内に]]>という文字列を書くことはできません。

4.2.4.2　ガジェット Core API

ガジェットにできることは単純な HTML の挿入だけではありません。JavaScript API を使用することで様々な機能を実現できます。次に API の機能を利用した、もう少し実用的なガジェットをみてみましょう。

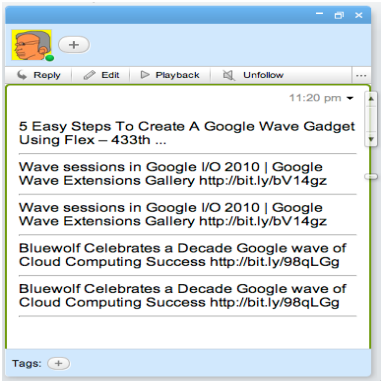
http://wave-book.appspot.com/assets/gadget2.xml

```
1  <?xml version="1.0" encoding="UTF-8" ?>
```

[*5] 2010 年 7 月現在

```
<Module>
<ModulePrefs title="My Second Gadget" height="500" />
<Content>
  <![CDATA[
    <div id="results" />
    <script type="text/javascript">
      var SEARCH_URL = 'http://search.twitter.com/search.json?';
      var SEARCH_KEYWORD = 'google wave';
      var RESULTS_DIV = document.getElementById('results');

      // tweet 検索結果を表示する
      function showResults(response) {
        var json = gadgets.json.parse(response.data);
        RESULTS_DIV.innerHTML = '';
        for (var i = 0; i < json.results.length; i++) {
          RESULTS_DIV.innerHTML += json.results[i].text + '<hr
/>';
        }

        // 5 分後に searchTwitter 関数を実行して検索結果を更新する
        window.setTimeout(searchTwitter, 5 * 60 * 1000);
      }

      // twitter API を利用して'google wave' を含む tweet を 5 件検索する
      function searchTwitter() {
        var params = {q:SEARCH_KEYWORD, rpp:5};
        var url = SEARCH_URL + gadgets.io.encodeValues(params);

        // リクエストを送信して、レスポンスを showResults 関数で処理する
        gadgets.io.makeRequest(url, showResults);
      }

      // ガジェットが読み込まれたときに searchTwitter 関数を実行する
      gadgets.util.registerOnLoadHandler(searchTwitter);
    </script>
  ]]>
</Content>
</Module>
```

上記のガジェットをネット上で公開し、URL を指定して wave に貼りつけると、Twitter API を利用して「google wave」という語句を含む tweet を検索して最新の 5 件を Wave 上に表示します。さらに、一度表示した後も 5 分ごとに検索結果をリフレッシュします。



それではソースコードを簡単に追ってみましょう。代入文と定義文を除けば、初めに実行されるのは<script>要素内の最後にある gadgets.util.registerOnLoadHandler(searchTwitt 関数呼び出しです。これによって、ガジェットの読み込みが完了したときに searchTwitter 関数を実行して twit を検索するように設定しています。ガジェット内では window.onload 関数は動作しないので注意してください。その searchTwitter 関数では gadgets.io.makeRequest 関数を使用して Twitter API を実行し、検索結果を showResults 関数に渡します。最後に showResults 関数で検索結果を画面に表示し、さらに window.setTimeout 関数で 5 分後にもう一度 searchTwitter 関数を実行するように設定しています。では、このガジェットで使用した 4 つの API を説明しましょう。

gadgets.json.parse(string)　JSON 文字列をパースしてオブジェクトを返します。この例では Twitter の search API の返り値をパースするのに使用しています。

gadgets.io.encodeValues(object)　引数として受け取ったオブジェクトのプロパティからエンコードされたリクエストパラメータを生成します。search API に検索文字列と結果項目数を渡すために使用しています。

gadgets.io.makeRequest(url, callback, options)　非同期に URL をリクエストして、レスポンスをコールバック関数に渡します。AJAX 通信は通常同一ドメイン内でしか動作しませんが、この API はサーバー側で通信を肩代わりすることで擬似的にクロスドメイン AJAX を実現しています。search API を実行するために使用しています。

gadgets.util.registerOnLoadHandler(callback)　ガジェットがロードされた時に実行されるハンドラ関数を登録します。ガジェットでは window.onLoad 関数が使用できないことに注意してください。ここではガジェットが読み込み終わってから Twitter 検索を開始するのに使用しています。

wave ガジェットで利用できるコア API は、上記の通り大きく分けて gadgets.util, gadgets.io, gadgets.json の 3 種類です[*6]。それぞれのオブジェクトには今回使用していない関数もたくさん定義されています。詳細については 4.2.11 節を参照してください。

[*6] ガジェット API としてはさらにもう一つ gadgets.Prefs というものがありますが、wave ガジェットには UserPrefs がないため使用できません

4.2.5 Wave ガジェット

ここまでの 2 つのガジェットは Wave に制限されない、さまざまなコンテナ上で動作する汎用的なものでした。ここからは Wave 専用のガジェットについて説明します。まず Wave ガジェット独自の機能にはどういうものがあるのか見ておきましょう。

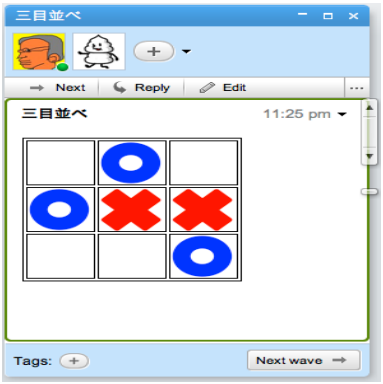
wave 参加者間でのリアルタイムな状態共有 wave ガジェットを使用することで、wave の最大の特徴であるリアルタイムコミュニケーションを、よりリッチに拡張できます。例えば地図ガジェットであれば表示している座標や拡大率がリアルタイムに共有されますし、チェスガジェットであればコマの位置がプレイヤー同士はもちろん観戦者も含めてリアルタイムに共有されます。

状態操作の履歴を使ったプレイバック ガジェットで使用する共有状態の変更履歴も wave サーバーによって管理されているため、ガジェットをプレイバック機構に対応させるために開発者が特別に何かをする必要はありません。例えばチェスガジェットであれば、開発者はただプレイヤーが指定した順番に駒を動かすだけで、wave が盤面の巻き戻しに対応してくれます。

wave の参加者情報取得 Wave ガジェットは wave のコンテンツにアクセスすることはできませんが、wave の参加者情報にはアクセスできます。なお、wave 以外のガジェットではガジェットが張り付けられたページがどのユーザーに所属するかが明確に決まっていて、ガジェットの所有者はそのユーザーになります。ところが、wave は特定のユーザーに所属すると言うことはないため、wave ガジェットも特定のユーザーではなく wave 自身に所属しています。そのためユーザー情報の扱いが wave ガジェット以外のガジェットとは少し異なります。

4.2.6 三目並べガジェット

最初に取り上げる wave ガジェットは、リアルタイムな状態共有機能を利用して参加者同士が三目並べを楽しむことができるガジェットです。



ソース全文は次のようになります。

http://wave-book.appspot.com/assets/gadget4.xml

```
1  <?xml version="1.0" encoding="UTF-8" ?>
```

```
<Module>
  <ModulePrefs title="三目並べガジェット" height="200">
    <Require feature="wave" />
  </ModulePrefs>
  <Content type="html">
    <![CDATA[
    <style>
      table { border:1px solid black; }
      td { width:50px; height:50px; background-image:none;
           border:1px solid black; background-image:none; }
      td.o { background-image:url(http://...appspot.com/o.png);
             background-repeat:no-repeat; }
      td.x { background-image:url(http://...appspot.com/x.png);
             background-repeat:no-repeat; }
    </style>

    <table>
      <tr>
        <td id="c00" onclick="changeState(0, 0)" />
        <td id="c01" onclick="changeState(0, 1)" />
        <td id="c02" onclick="changeState(0, 2)" />
      </tr>
      <tr>
        <td id="c10" onclick="changeState(1, 0)" />
        <td id="c11" onclick="changeState(1, 1)" />
        <td id="c12" onclick="changeState(1, 2)" />
      </tr>
      <tr>
        <td id="c20" onclick="changeState(2, 0)" />
        <td id="c21" onclick="changeState(2, 1)" />
        <td id="c22" onclick="changeState(2, 2)" />
      </tr>
    </table>

    <script type="text/javascript">
      // {クリック前の値:クリック後の値}
      var CLASS_TABLE = {'':'o', 'o':'x', 'x':''};

      // クリックされたときにセルの色を変える関数
      function changeState(r, c) {
        var state = wave.getState();
        if (state) {
```

```
44            var cell = document.getElementById('c' + r + c);
45            cell.className = CLASS_TABLE[" + cell.className];
46            state.submitValue(cell.id, cell.className);
47          }
48        }
49
50        // 他の参加者がクリックしたときに呼び出されてセルの色を変える関数
51        function stateChanged() {
52          var state = wave.getState();
53          var keys = state.getKeys();
54          for (var i = 0; i < keys.length; i++) {
55            var key = keys[i];
56            var elm = document.getElementById(key);
57            if (elm) elm.className = state.get(key, '');
58          }
59        }
60
61        // 初期化関数
62        // ここではコールバックの設定のみ
63        function init() {
64          if (wave && wave.isInWaveContainer()) {
65            wave.setStateCallback(stateChanged);
66          }
67        }
68
69        // ガジェットのロードが完了したときに実行される初期化関数を登録
70        //  (window.onLoad は使用できないので注意してください)
71        gadgets.util.registerOnLoadHandler(init);
72      </script>
73      ]]>
74    </Content>
75  </Module>
```

全体的な流れとしては、あるユーザーがテーブルのセルをクリックするとガジェットはコマの表示を変更すると同時に、全参加者によって共有されているゲーム盤の状態も変更します。そしてクリックしたユーザーとは別のユーザーのガジェットはそのゲーム盤の状態が変更されたことを察知して画面の表示を描き換えます。非常にシンプルな三目並べガジェットで、手番の管理も終了判定もなくクリックしたコマの表示が順に変わるだけです。しかし先に書いたように盤上のコマの状態は全参加者間で共有されているので、紳士協定に基づけば十分にゲームを楽しむことができるでしょう。

それでは、いくつかのパーツに区切って最初から順にソースを見ていきましょう。

4.2.6.1　Wave ガジェット XML

Wave 独自の機能をガジェットで使用するには一つだけやらなければいけないことがあります。それは<Require>タグでガジェットコンテナ（Wave サーバー）に対して wave フィーチャーをリクエストしておくことです。wave フィーチャーをリクエストしていなければ、<Content>タグ内で wave 独自の関数を呼び出してもコンテナはそれを解釈できません。従って wave の機能を利用するガジェット XML の雛形は次のようになります。

```
 1  <?xml version="1.0" encoding="UTF-8" ?>
 2  <Module>
 3    <ModulePrefs title="Wave Gadget Title">
 4      <Require feature="wave" />
 5    </ModulePrefs>
 6    <Content type="html">
 7      <![CDATA[
 8        ここに HTML, JS, CSS などを記述
 9      ]]>
10    </Content>
11  </Module>
```

4.2.6.2　共有状態（Shared State）オブジェクト

共有状態オブジェクトとはその名の通りガジェットの参加者全員が共有する状態を管理しているオブジェクトで、三目並べガジェトでも駒の配置を共有するために使用しています。どのように共有データを保持しているかを理解しておくことはガジェットのソースコードを読む上で必須ですから、まず初めに詳細に見ておきましょう。

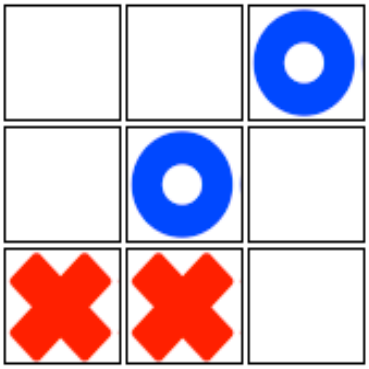
共有状態オブジェクトの中身は単純な連想配列、つまりキーと値の集合です。ただし、ただ単純にキーと値を保持しているだけではなく、事前にコールバック関数を設定しておいて、状態が変更されたときにその関数を呼び出すことができます。このコールバックという機能によって、ある参加者が共有状態を変更したときに、それ以外のすべての参加者はコールバック関数呼び出しを通じて即座に共有状態の変更を察知できるのです。共有状態オブジェクトこそが開発者が wave の特徴であるリアルタイム編集を最も直接的に利用するための手段で、wave ガジェットの中で最重要のオブジェクトと言っても過言ではありません。

しかし、残念ながら共有状態オブジェクトが値として受け取ることができるのは文字列だけです。例えば今回のような三目並べではゲーム盤を直接的に表現すると 2 次元配列になるかと思いますが、それをそのまま共有状態オブジェクトに保持させることはできません。といっても難しく考える必要はありません。複雑な構造を持つオブジェクトを保持させたいのであれば、JSON を使えばいいのです。

4.2.6.3　共有状態の方針決定

共有状態オブジェクトを利用する場合に最も重要なのはどのようなキーと値の組で状態を管理するかを決定することです。三目並べのようなシンプルなゲームでも、局面を表すための方法は一種類ではありません。例えば次のようなゲーム盤の状態を共有状態オブ

ジェクトに設定するために



JSON 形式にした 2 次元配列を使用して、'board' という単一のキーに対して盤面全体を値として与えることもできれば

```
1 {
2   'board': "[ \
3     [ '',  '', 'o'], \
4     [ '', 'o',  ''], \
5     ['x', 'x',  ''] \
6   ]"
7 }
```

次のようにセルの一つ一つの状態をばらばらに管理することもできます。

```
1 {
2   'c02': 'o',
3   'c11': 'o',
4   'c20': 'x',
5   'c21': 'x'
6 }
```

今回の三目並べでは後者の方法を選びました。これには理由があります。実は共有状態ではキーと値の組が変更の最小単位になり、値同士の差分などは考慮されないのです。つまり前者のような状態の持ち方をすると盤全体を一括で変更することしかできず、後者に比べて非常に衝突が発生しやすくなります。例えば先の状態から A さんが左上のセルに、B さんが右下のセルにほぼ同時にコマ"o"をおく場合を考えます（三目並べではありえないことですが・・・）。前者のように盤全体を一つの値として管理していた場合、A さん・B さんはお互い希望の状態に盤面を更新するためにそれぞれ次のような値を共有状態オブジェクトに設定します。

A さん

```
1 {
2   'board': "[ \
3     ['o',  '', 'o'], \
4     [ '', 'o',  ''], \
5     ['x', 'x',  ''] \
```

```
6    ]"
7  }
```

B さん

```
1  {
2    'board': "[ \
3      [ '',  '', 'o'], \
4      [ '', 'o',  ''], \
5      ['x', 'x', 'o'] \
6    ]"
7  }
```

当然ですがどちらもキー'board' に対する値の変更になっています。これらのリクエストがほぼ同時に発生すると、最終的にどちらが優先されるかはわかりませんが、いずれにしても先に反映された変更は上書きされて、なかったことになってしまいます。一方セル単位で状態を管理していた場合には、A さんはキー'c00' に対する値の変更、B さんはキー'c22' に対する値の変更をすることになり、キーが異なるためそもそも衝突が発生せず、どちらの変更も有効になります。

A さん

```
1  {'c00': 'o'}
```

B さん

```
1  {'c22': 'o'}
```

つまり共有状態オブジェクトにどのような形式で状態を保持するかを考えるときには、独立して変更できる値は異なるキーで管理して衝突をかぎる限り避けることが重要です。もちろん常に全てをばらばらに管理するのが最善というわけではなく、関連のあるデータ群であれば一つのキーの値としてまとめた方がいい場合もあるでしょう。しかし、いずれにしても 一旦決定した共有状態の持ち方を後で変更するのは大きなコストが掛かります。安易に決定してしまわず、事前によく考えて実装を開始するようにしましょう。

4.2.6.4　CSS と HTML

次に CSS と HTML を見てみましょう。

```
1  <style>
2    table { border:1px solid black; }
3    td { width:50px; height:50px; ... }
4    td.o { background-image:url(http://... }
5    td.x { background-image:url(http://... }
6  </style>
7
8  <table>
9    <tr>
10     <td id="c00" onclick="changeState(0, 0)" />
11     <td id="c01" onclick="changeState(0, 1)" />
12     <td id="c02" onclick="changeState(0, 2)" />
13   </tr>
```

```
14    <tr>
15      <td id="c10" onclick="changeState(1, 0)" />
16      <td id="c11" onclick="changeState(1, 1)" />
17      ...
18  </table>
```

コア API の例（4.2.4.2 節）とは違い、今回は CSS を使用しました。といっても CDATA 要素に<style>タグを追加するだけなので特に問題はないでしょう。ゲーム盤は 3 行 3 列<table>タグで表し、<td>タグにはアクセスを容易にするため「c + 行番号 + 列番号」と言う形式の ID を設定しています。なお、この ID は共有状態のキーとしても使用します。

4.2.6.5 ガジェットの初期化

それではここから<script>タグの中に入っていきます。まずはガジェットの初期化について。wave というオブジェクトに属している関数が wave ガジェット用の API です。

```
1  // 初期化関数
2  // ここではコールバックの設定のみ
3  function init() {
4    if (wave && wave.isInWaveContainer()) {
5      wave.setStateCallback(stateChanged);
6    }
7  }
8
9  // ガジェットのロードが完了したときに実行される初期化関数を登録
10 //  (window.onLoad は使用できないので注意してください)
11 gadgets.util.registerOnLoadHandler(init);
12
```

先ほど説明したようにガジェットでは window.onLoad 関数は利用できません。その代わり、ガジェットのロード完了時になにか処理をしたい場合には gadget.util.registerOnLoadHandler 関数で初期化関数を登録します。wave ガジェットの初期化関数ではほとんどの場合、ガジェットに必要なコールバック関数が登録されます。今回は wave.setStateCallback 関数で他の参加者が共有状態を変更したときにコールバックされる関数（stateChanged）を登録しました。共有状態の変更以外に設定できるコールバックもいくつかありますが、それらについてはこの次の例で紹介します。

なお、wave ガジェットは標準的なガジェット API の拡張になっていますので、wave ではなく iGoogle などに張り付けるユーザーもいるかもしれませんが、その場合はもちろん wave コンテナの機能は使用できません。そのようなときでもエラーが発生しないように、コールバック関数登録前に wave.isInWaveContainer 関数を呼び出してガジェットが実際に wave に張り付けられていることを確認しています。

4.2.6.6 状態の変更とその共有

三目並べガジェットはテーブルのセルがクリックされるとその表示を o -> x -> 空白 -> o ... と順番に変更し、さらにその変更はすべての wave 参加者に共有されます。実際

に処理を担当している部分を抜き出してみましょう。

```
1    <td id="c00" onclick="changeState(0, 0)" />
2
3  // {クリック前の値:クリック後の値}
4  var CLASS_TABLE = {'':'o', 'o':'x', 'x':''};
5
6  // クリックされたときにセルの色を変える関数
7  function changeState(r, c) {
8    var state = wave.getState();
9    if (state) {
10     var cell = document.getElementById('c' + r + c);
11     cell.className = CLASS_TABLE['' + cell.className];
12     state.submitValue(cell.id, cell.className);
13   }
14 }
```

表示の変更は<td>要素のクラスの変更と CSS で実現されています。さらにその変更された要素の ID と変更後のクラス名が共有状態オブジェクトを通じて参加者に共有されます。具体的には wave.getState 関数で取得した共有状態オブジェクト（state）の State#submitValue 関数を使って他のユーザーと変更された状態を共有します。

4.2.6.7　他の参加者が共有状態を変更したときのコールバック関数

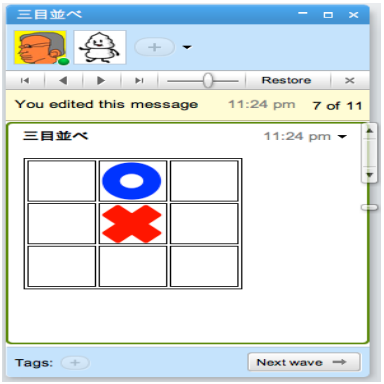
最後に先ほどの初期化関数内で登録したコールバック関数を見てみます。あるユーザーがセルをクリックして State#submitValue 関数が共有状態を変更すると、その他のユーザーのガジェットではこの関数が呼び出されることになります。

```
1  // 他の参加者がクリックしたときに呼び出されてセルの色を変える関数
2  function stateChanged() {
3    var state = wave.getState();
4    var keys = state.getKeys();
5    for (var i = 0; i < keys.length; i++) {
6      var key = keys[i];
7      var elm = document.getElementById(key);
8      if (elm) elm.className = state.get(key, '');
9    }
10 }
```

wave.getState 関数で共有状態オブジェクトを取得するところは前項と変わりません。先ほどはその共有状態オブジェクトを使用して状態の変更を共有しましたが、今回はそこから共有されている状態を取り出して<td>要素のクラス名を設定しています。詳細に書くと、State#getKeys 関数で状態のキー（要素の ID）をすべて取り出し、一つ一つのキーについて state.get(key, '') で値（要素のクラス名）を取り出しています。なお State#get 関数の第二引数はキーに対応する値が存在しなかった場合のデフォルト値です。

4.2.6.8 プレイバック

チェスガジェットを触ったことがあれば分かるかと思いますが、お互いの手番を遡って確認できるプレイバックはボードゲームと非常に相性のいい機能です。この三目並べでもプレイバックを利用するにはどうすればいいでしょう？実は、何もしなくて構いません。共有状態オブジェクトはデフォルトでプレイバックをサポートしていて、そのままで加えられた変更の履歴を再生できます。三目並べガジェットを wave に貼り付けて、ひとしきりコマを置いたり剥がしたりしたあとで wave をプレイバックモードにしてみましょう。ちゃんと盤面の変化が順に再生できることがわかります。



共有状態オブジェクトを使用した三目並べガジェットの説明は以上です。常に特定の参加者しかコマを置けないようにターン制を導入したり、終了条件をチェックしたりと、このガジェットを拡張できる点はいくらでもあります。実際に動かしてみた後はぜひ自分なりの機能を追加してみてください。

共有状態オブジェクトにはここでは使っていない関数がいくつもあります。詳細については 4.2.11 節を参照してください。

4.2.7 スケジュール調整ガジェット

Wave ガジェット API を使って利用できるのは共有状態だけではありません。API を利用すると wave の参加者や表示モード（編集・閲覧・プレイバック）などの情報にもアクセスできます。本節では先ほどの例で使用した共有状態以外の API も利用した、もう少し複雑なガジェットを作ってみることにします。

4.2.7.1 ガジェット概要

本節で作成するガジェットはスケジュール調整ガジェットです。ガジェットを貼り付けてイベントの候補日をカンマ区切りで複数日入力したあとで、参加者を Wave に誘って参加可能な日を選んでもらいます。もちろん参加者が選んだ内容はすぐに全員に共有され、それぞれの候補日について wave 内で議論を交わすこともできます。

4.2.7.2　ガジェット XML

それでは早速ガジェットのソースを見てみましょう。

```
 1  <?xml version="1.0" encoding="UTF-8" ?>
 2  <Module>
 3    <ModulePrefs title="Attendance Book Gadget" height="100">
 4      <Require feature="wave" />
 5      <Require feature="dynamic-height" />
 6    </ModulePrefs>
 7    <Content type="html">
 8      <![CDATA[
 9
10      <div id="viewer">
11        <select id="color">
12          <option value="black" selected>黒</option>
13          <option value="blue">青</option>
14          <option value="red">赤</option>
15        </select>
16        <button id="color-button" onclick="setColor()">色　を　変　更
</button>
17        <div id="calendar"></div>
18      </div>
19
20      <div id="editor">
21        ","区切りで候補日を入力して「日付を設定」をクリックして下さい。<br
/>
22        <input id="dates" />
23        <button id="dates-button" onclick="setDates()">日 付 を 設 定
</button>
24      </div>
25      <link rel="stylesheet" href="http://.../attendancebook.css"
26        type="text/css" />
27      <script src="http://.../attendancebook.js"></script>
28      ]]>
```

```
29    </Content>
30  </Module>
```

今回の例では JavaScript が複雑になるので、XML と JavaScript と CSS をそれぞれ別のファイルに記述しました。それらを別ファイルに分けるには、例のように JavaScript の場合は<script>タグを、スタイルシートタグの場合は<link>タグを使用します。JavaScript やスタイルシートが大きくなるとガジェット全体の見通しが悪くなるため、テスト用の簡単なガジェットでなければ、初めからこのように外部ファイルを読み込むようにしておいた方がいいでしょう。

さらに、<ModulePrefs>の中で wave フィーチャーに加えて、dynamic-height フィーチャーを Require しています。これはその名の通りガジェットの高さを動的に変更するためのフィーチャーになります。通常であればガジェットの高さは<ModulePrefs>タグの height 属性で指定したものに固定されてしまいますが、今回のガジェットは wave 参加者の人数によって高さを動的に変更する必要があります。そのためにこのフィーチャーを利用しました。実際にどのようにしてガジェットサイズを動的に変更するかについては後で説明します。

CDATA セクションには<link>タグと<script>タグの他に 2 つの DIV 要素があります。これらはそれぞれ、id が"viewer"の DIV は閲覧モードの時に表示される要素、id が"editor"の DIV は編集モードの時に表示される要素です。

4.2.7.3 ソース全文

ガジェット XML から読み込まれる CSS ファイルと JS ファイルの全文は次のようになります。

attendancebook.css

```
 1  div.corner, div.date, div.participant, div.choice {
 2    border:1px solid black;
 3    float:left;
 4    font-size:small;
 5    height:20px;
 6    margin:3px;
 7    overflow:hidden;
 8    padding:3px;
 9  }
10  div.choice { width:70px; }
11  div.corner { width:150px; }
12  div.black div.date { background-color:black; }
13  div.blue div.date { background-color:blue; }
14  div.red div.date { background-color:red; }
15  div.date { color:white; }
16  div.date { width:70px; }
17  div.black div.participant { background-color:black; }
```

```
18  div.blue div.participant { background-color:blue; }
19  div.red div.participant { background-color:red; }
20  div.participant { color:white; }
21  div.participant { width:150px; }
```

attendancebook.js

```
 1  // 要素を保持しておく
 2  var datesText = document.getElementById('dates');
 3  var editorDiv = document.getElementById('editor');
 4  var viewerDiv = document.getElementById('viewer');
 5  var colorSel = document.getElementById('color');
 6
 7  // モードに応じた画面を描画する関数
 8  function render() {
 9    var state = wave.getState();
10    if (!state) return;
11
12    if (wave.getMode() == wave.Mode.EDIT) {
13      renderEdit(state);
14    }
15    else {
16      renderView(state);
17    }
18    gadgets.window.adjustHeight();
19  }
20
21  // エディットモードの画面を描画する関数
22  function renderEdit(state) {
23    viewerDiv.style.display = 'none';
24    editorDiv.style.display = 'block';
25    datesText.value = state.get('dates', '');
26    datesText.editable = wave.getHost() == wave.getViewer();
27  }
28
29  // ビューモードの画面を描画する関数
30  function renderView(state) {
31    var dates = state.get('dates', '').split(',');
32    var participants = wave.getParticipants();
33    viewerDiv.style.display = 'block';
34    editorDiv.style.display = 'none';
35
36    var html = '<div class="corner"></div>';
```

```
37    for (var i = 0; i < dates.length; i++) {
38      html += '<div class="date">' + dates[i] + '</div>';
39    }
40    html += '<br style="clear:both;" />';
41    for (var i = 0; i < participants.length; i++) {
42      var participant = participants[i];
43      html += '<div class="participant">' +
44        '<img class="thumb" src="' + participant.getThumbnailUrl() +
45        '" width="20" height="20" />' +
46        participant.getDisplayName().substring(0, 15) + '</div>';
47      for (var j = 0; j < dates.length; j++) {
48        var key = participant.getId() + '-' + dates[j];
49        html += '<div id="' + key + '" class="choice" ' +
50          'onclick="javascript:updateState(\'' + participant.getId()
+
51          '\', \'' + dates[j] + '\')">' +
52          state.get(key, '') + '</div>';
53      }
54      html += '<br style="clear:both;" />';
55    }
56
57    document.getElementById('calendar').innerHTML = html;
58  }
59
60  // 参加可・不可の表示を切り替える関数
61  var STATE_TABLE = {'':' 参加可能', ' 参加可能':' 参加不能', ' 参加不
能':''};
62  function updateState(pid, date) {
63    var key = pid + '-' + date;
64    var state = wave.getState();
65    if (state && wave.getViewer().getId() == pid) {
66      var currentState = document.getElementById(key).innerHTML;
67      state.submitValue(key, STATE_TABLE[currentState]);
68    }
69  }
70
71  // 候補日を設定する関数
72  function setDates() {
73    var state = wave.getState();
74    if (state && datesText.value) {
75      state.submitValue('dates', datesText.value);
76    }
```

```
77  }
78
79  // カレンダーの色を変更する関数
80  function setColor() {
81    var color = colorSel.options[colorSel.selectedIndex].value;
82    viewerDiv.className = color;
83    var privateState = wave.getPrivateState();
84    if (privateState) {
85      privateState.submitValue('color', color);
86    }
87  }
88
89  // カレンダーの色を読み込んで設定する関数
90  function loadColor() {
91    var color = wave.getPrivateState().get('color', 'black');
92    viewerDiv.className = color;
93    var index = {'black':0, 'blue':1, 'red':2}[color];
94    colorSel.options[index].selected = true;
95  }
96
97  // 初期化関数
98  function init() {
99    if (wave && wave.isInWaveContainer()) {
100     wave.setModeCallback(render);
101     wave.setParticipantCallback(render);
102     wave.setStateCallback(render);
103     wave.setPrivateStateCallback(loadColor);
104
105     wave.ui.loadCss();
106     wave.ui.makeButton(document.getElementById('color-button'));
107     wave.ui.makeButton(document.getElementById('dates-button'));
108   }
109 }
110
111 // ガジェットのロードが完了したときに実行される初期化関数を登録
112 gadgets.util.registerOnLoadHandler(init);
```

4.2.7.4　ガジェットの初期化

ここからは JS ファイルの中を見ていきましょう。まずはガジェットの初期化です。

```
1  // 初期化関数
2  function init() {
3    if (wave && wave.isInWaveContainer()) {
4      wave.setModeCallback(render);
```

```
5       wave.setParticipantCallback(render);
6       wave.setStateCallback(render);
7       wave.setPrivateStateCallback(loadColor);
8
9       wave.ui.loadCss();
10      wave.ui.makeButton(document.getElementById('color-button'));
11      wave.ui.makeButton(document.getElementById('dates-button'));
12    }
13  }
14
15  // ガジェットのロードが完了したときに実行される初期化関数を登録
16  gadgets.util.registerOnLoadHandler(init);
```

gadgets.util.registerOnLoadHandler 関数で初期化関数を登録し、初期化関数内で各種ハンドラを登録するのは三目並べと同じです。今回は wave.setStateCallback 関数に加えて、wave.setModeCallback 関数・wave.setParticipantCallback 関数・wave.setPrivateStateCallback 関数も使用しました。関数名から読み取れるとは思いますが、それぞれ、表示モード変更時、参加メンバー変更時、プライベート状態変更時、に呼び出されます。wave ガジェットで利用できるコールバックは現在のところこの 4 つだけです。wave オブジェクトの詳細については 4.2.11 節を参照してください。

4.2.7.5 ガジェットの Look & Feel

ガジェットの初期化関数ではコールバック関数を登録する他に、wave.ui オブジェクトを利用して、ガジェットで使用しているボタンを wave でよく使われる見た目に変更しています。例から分かるように wave.ui の利用は非常に簡単です。ボタンの見た目を変更するのであれば、wave.ui.loadCss 関数呼び出しを実行したあとで、wave.ui.makeButton 関数にボタン要素を渡してください。これによってボタンの見た目が右から左のように変更されます（下図）。

wave.ui は必須ではありませんが、wave 内でガジェットがちぐはぐな印象を与えないように、なるべく使用するようにした方がいいでしょう。wave.ui に定義されている makeButton 以外の関数については 4.2.11 節を参照してください。

4.2.7.6 render 関数

それでは次にコールバック関数としてほとんどのイベントに紐づいている render 関数の実装を見てみましょう。

```
1  // モードに応じた画面を描画する関数
2  function render() {
3    var state = wave.getState();
4    if (!state) return;
5
6    if (wave.getMode() == wave.Mode.EDIT) {
```

```
 7       renderEdit(state);
 8     }
 9     else {
10       renderView(state);
11     }
12     gadgets.window.adjustHeight();
13   }
```

render 関数はその名の通り、画面を描画するための関数です。本来であればモードの変更・参加者の変更などイベントごとに必要な部分だけを書き換える関数を用意して、setCallback すればいいのかもしれませんが、今回はあまり複雑な画面でもないので毎回全てを書き換えるようにしました。実際の描画は表示モードに応じて renderEdit 関数か renderView 関数のどちらがかが実行します。表示モードはサンプルの通り wave.getMode 関数で取得でき、その返り値は次のいずれかです。

wave.Mode オブジェクト

DIFF_ON_OPEN　ガジェットを含む blip をユーザーが最後に開いた後で変更がありました

EDIT　ガジェットを含む blip が編集中です

PLAYBACK　ガジェットを含む blip がプレイバック中です

UNKNOWN　ガジェットを含む blip のモードが不明です

VIEW　ガジェットを含む blip が閲覧中です

今回は編集モードの場合だけ候補日入力用の画面を表示し、それ以外の場合はすべて候補日と参加者の表を表示することにしました。

また render 関数の最後で gadgets.window.adjustHeight 関数を呼び出しています。これは<ModulePrefs>で"dynamic-height"フィーチャーを Require することで利用可能になった関数です。この関数を実行することで表示モードや参加者の数に応じて変化する表示領域に合わせてガジェット自体の高さを調整しています。

4.2.7.7　renderEdit 関数

```
1  // エディットモードの画面を描画する関数
2  function renderEdit(state) {
3    viewerDiv.style.display = 'none';
4    editorDiv.style.display = 'block';
5    datesText.value = state.get('dates', '');
6    datesText.editable = wave.getHost() == wave.getViewer();
7  }
```

renderEdit 関数では閲覧用の要素を非表示にした後、編集用の要素を表示して、共有状態オブジェクトから候補日を取り出してテキストフィールドに設定します。さらに、閲覧中のユーザー（wave.getViewer()）がそのガジェットを追加したユーザー（wave.getHost()）なら、テキストフィールドを編集可能に、そうでなければ編集不能

にします。

```
1  <div id="editor">
2    ","区切りで候補日を入力して「日付を設定」をクリックして下さい。<br />
3    <input id="dates" />
4    <button id="dates-button" onclick="setDates()">日付を設定</button>
5  </div>
```

編集用の要素はカンマ区切りで日付のリストを入力するためのテキストフィールドと、その入力を確定するためのボタンからなっています。ボタンをクリックすると setDates 関数が実行され、テキストフィールドに入力された候補日のリストが共有状態オブジェクトに保存されます。setDates 関数の定義は次のようになっています。

```
1  // 候補日を設定する関数
2  function setDates() {
3    var state = wave.getState();
4    if (state && datesText.value) {
5      state.submitValue('dates', datesText.value);
6    }
7  }
```

テキストフィールド（datesElm）の値を、"dates"というキーで共有状態オブジェクトに登録します。

4.2.7.8 renderView 関数

renderEdit 関数はすでに用意されている要素を表示するだけの非常にシンプルな関数でした。スケジュール表を表示する renderView 関数はすでにある要素を表示するのではなく、設定されている候補日や参加者に合わせた要素を関数内で組み立てて表示するため少し複雑です。いきなり renderView 関数の説明に移るより、先に組み立てられる要素がどのようなものなのか見ておいた方が理解しやすいでしょう。例えばイベントの候補日が「4 月 1 日」と「4 月 2 日」、wave の参加者が「Yasushi ANDO」さんと「佐々木 陽」さんのときに組み立てられる要素は次のようなものになります。（適宜改行・空白を加えています）


```
1  <div class="corner"/>
2  <div class="date">4 月 1 日</div>
```

```
 3  <div class="date">4 月 2 日</div>
 4  <br style="clear: both;"/>
 5  <div class="participant">
 6    <img height="20" width="20" src="..." class="thumb"/>
 7    Yasushi ANDO
 8  </div>
 9  <div class="choice" id="foo@googlewave.com-4 月 1 日"
10    onclick="updateState('foo@googlewave.com', '4 月 1 日')" />
11  <div class="choice" id="foo@googlewave.com-4 月 2 日"
12    onclick="updateState('foo@googlewave.com', '4 月 2 日')" />
13  <br style="clear: both;"/>
14  <div class="participant">
15    <img height="20" width="20" src="..." class="thumb"/>
16    佐々木 陽
17  </div>
18  <div class="choice" id="bar@googlewave.com-4 月 1 日"
19    onclick="updateState('bar@googlewave.com', '4 月 1 日')" />
20  <div class="choice" id="bar@googlewave.com-4 月 2 日"
21    onclick="updateState('bar@googlewave.com', '4 月 2 日')" />
22  <br style="clear: both;"/>
```

タグを、横軸が候補日・縦軸が参加者のテーブル状に並べています。クラスが"choice"の タグをクリックすると、updateState 関数が呼ばれ、セルの表示内容が「参加可能」->「参加不能」-> 空白 -> 「参加可能」 -> ... と順番に切り替わり、さらにその内容が共有状態を通じて参加者間で共有されます。なお、この updateState 関数については、三目並べガジェットの changeState 関数とほぼ同じですので、今回は説明を省略しています。

それでは renderView 関数の実装を見てみましょう。

```
 1  // ビューモードの画面を描画する関数
 2  function renderView(state) {
 3    var dates = state.get('dates', '').split(',');
 4    var participants = wave.getParticipants();
 5    viewerDiv.style.display = 'block';
 6    editorDiv.style.display = 'none';
 7
 8    var html = '<div class="corner"></div>';
 9    for (var i = 0; i < dates.length; i++) {
10      html += '<div class="date">' + dates[i] + '</div>';
11    }
12    html += '<br style="clear:both;" />';
13    for (var i = 0; i < participants.length; i++) {
14      var participant = participants[i];
15      html += '<div class="participant">' +
```

```
16        '<img class="thumb" src="' + participant.getThumbnailUrl() +
17        '" width="20" height="20" />' +
18        participant.getDisplayName().substring(0, 15) + '</div>';
19      for (var j = 0; j < dates.length; j++) {
20        var key = participant.getId() + '-' + dates[j];
21        html += '<div id="' + key + '" class="choice" ' +
22          'onclick="javascript:updateState(\'' + participant.getId()
+
23          '\', \'' + dates[j] + '\')">' +
24          state.get(key, '') + '</div>';
25      }
26      html += '<br style="clear:both;" />';
27    }
28
29    document.getElementById('calendar').innerHTML = html;
30  }
```

一見複雑そうに見えますが、要するに候補日リストと参加者リストから先ほど見た要素を組み立てて ID が calendar の DIV 要素の innerHTML に設定しているだけです。参加者一覧は `wave.getParticipants` 関数で `Participant` オブジェクトの配列として取得できます。さらに、候補日一覧は `renderEdit` 関数と同じく共有状態オブジェクトから取得できますので、それらを二重にループして`<div>`要素をテーブル状に並べています。参加者のアイコンや名前は `Participant` オブジェクトのプロパティから取得できます。`Participant` オブジェクトの詳細については 4.2.11 節を参照してください。

4.2.7.9 プライベート状態

4.2.6.2 節で紹介した共有状態はガジェットの参加者全員に共有される状態でした。しかし、場合によっては参加者ごとに異なる状態が必要になることもあるでしょう。例えばカードゲームガジェットで各参加者の手札を共有状態で管理してしまうと、その手札は例え他の参加者の画面で表示されていないとしても、wave サンドボックスならデバッグメニューを利用して共有状態を確認できますし、通常の Wave でも JavaScript のデバッグツールを利用したちょっとしたハックで手札を見られてしまう可能性があります。そのような場合に利用できるのがプライベート状態です。

プライベート状態は wave によって管理されますが、同じキーであってもその値は参加者ごとに異なり、他の参加者のプライベート状態を見ることはできません。今回のガジェットではプライベート状態を、ユーザーごとに異なるガジェットの色設定を保持するために使用しました。

コンボボックスから色を選択し、「色を変更」ボタンをクリックするとカレンダーの色が変わります。この色は誰とも共有されておらず、それぞれのユーザーが自分の好きな色に設定できます。

```
<div id="viewer">
  <select id="color">
    <option value="black" selected>黒</option>
    <option value="blue">青</option>
    <option value="red">赤</option>
  </select>
  <button id="color-button" onclick="setColor()">色を変更</button>
  <div id="calendar"></div>
</div>
```

```
// カレンダーの色を変更する関数
function setColor() {
  var color = colorSel.options[colorSel.selectedIndex].value;
  viewerDiv.className = color;
  var privateState = wave.getPrivateState();
  if (privateState) {
    privateState.submitValue('color', color);
  }
}

// カレンダーの色を読み込んで設定する関数
function loadColor() {
  var color = wave.getPrivateState().get('color', 'black');
  viewerDiv.className = color;
  var index = {'black':0, 'blue':1, 'red':2}[color];
  colorSel.options[index].selected = true;
}

// 初期化関数
function init() {
```

```
21    // ...snip...
22    wave.setPrivateStateCallback(loadColor);
23    // ...snip...
24  }
```

カレンダーの色は DIV 要素のクラス名を変更することで設定しています。API の使用方法についてはプライベート状態も共有状態も全く同じです。

ところで、自分しか扱うことができない状態なら、どうして `setPrivateStateCallback` 関数でコールバック関数を設定する必要があるのか疑問に思う人もいるかも知れません。これは実際に初期化関数の該当部分をコメントアウトしてみるとわかります。実はコールバック関数は wave を開き直したときにも一度呼び出されます。そのため、コールバック関数をコメントアウトすると、wave を開き直したときに前回の設定内容が画面に反映されません。

・・・

これでスケジュール調整ガジェットの紹介が終わりました。参加者の予定を調節するのに必要な最低限度の機能は入っているのではないかと思いますが、実際に使用するにはまだまだ機能が足りません。読者の皆さんの工夫でどんどん使い易いガジェットに改造していただければと思います。また、本ガジェットはガジェット API に含まれるほぼ全ての種類のオブジェクトを利用していますが、全ての関数を利用しているわけではありません。それぞれのオブジェクトをさらに詳細に知りたい場合は 4.2.11 章を参照してください。

4.2.8 ガジェットで利用可能なフィーチャー

これまでの例では `wave` と `dynamic-height` という 2 つのフィーチャーを利用しました。wave ガジェットで利用可能なフィーチャーはこの 2 つだけではありません。

setprefs　ユーザープレファレンスの値をプログラムから設定できるようになります
dynamic-height　ガジェットが自分自身をリサイズできるようになります
tabs　タブインターフェースをガジェットで利用可能にします
minimessage　小さく一時的なメッセージを表示します
flash　Flash ムービーをガジェット内で使用可能にします

`minimessage` フィーチャーについてはこの後の項で利用方法を紹介します。これらのフィーチャーを利用すれば、少ない労力で複雑な機能をガジェットに付加できます。ぜひとも活用してください。それぞれのフィーチャーの詳細な機能・使用方法についてはガジェットのドキュメント[*7]を参照してください。

4.2.9 ガジェットのデバッグ方法

ガジェットはコンテナ上でなければ動作しない上に要素が iframe 内に収められるため、そのデバッグは一筋縄ではいきません。残念ながらその困難を一息に解決してくれる「銀

[*7] http://code.google.com/intl/en/apis/gadgets/docs/reference.html#feature-libs

の弾丸」は存在しませんが、デバッグに利用できるツールやテクニックはいくつかあります[*8]。ここではそれらの中から特に役に立ちそうなものをピックアップして紹介します。

4.2.9.1　キャッシュを無効にする

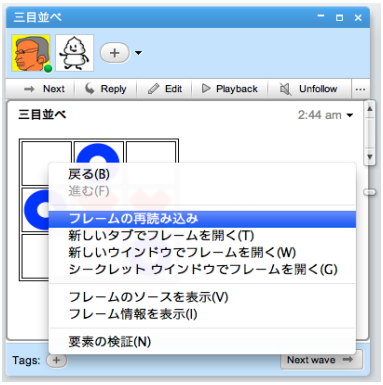
ガジェットの開発で一番厄介なのがコンテナによるガジェットのキャッシュです。Wave サンドボックスはガジェットをキャッシュしないので、ガジェットの開発はサンドボックスで行った方がいいでしょう。

通常の Wave はガジェットをキャッシュします。つまり、ガジェットを修正してサーバーに上げ直してもすぐには有効になりません。しばらくはコンテナにキャッシュされた古いガジェットが参照されます。この挙動がガジェットのデバッグに不便なのは明らかなので、wave にはガジェットのキャッシュを無効にするパラメータが用意されています。キャッシュを無効にするには URL の最後の"/"と"#"との間に`?gadget_cache=0` を付与してください。

- https://wave.google.com/wave/**?gadget_cache=0**#restored:wave:...

4.2.9.2　ガジェットをリロードする

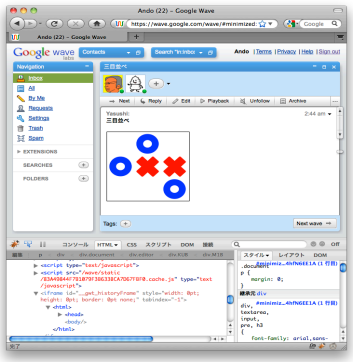
ガジェットのリロードは Google Wave 自体を開き直す必要はなく、ガジェットを含んでいる wave を閉じて、また開き直すだけでリロードできます。さらに、ブラウザによっては iframe だけをリロードできるものもあります。wave のレンダリングにはそれなりに時間がかかってしまうので、なるべく小さい範囲でリロードするように心がけるとストレスなく開発できるでしょう。



4.2.9.3　開発ツールを活用する

FireFox を使っているのなら FireBug Addon[*9]をインストールしておきましょう。Chrome を使用している場合はメニューから表示->開発/管理->デベロッパーツールを選択すれば同様なツールが利用できます。これらの開発ツールを使えばガジェットの DOM 構造や JavaScript のスタックトレースを確認できます。

[*8] http://code.google.com/intl/ja/apis/wave/articles/gadgetdebugging.html
[*9] https://addons.mozilla.org/ja/firefox/addon/1843

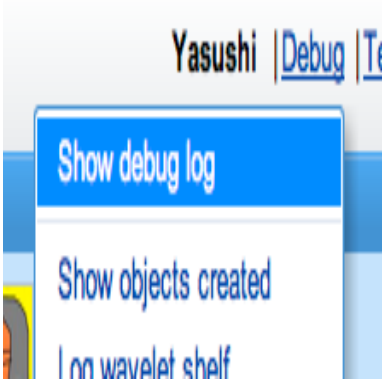


4.2.9.4 Wave サンドボックスのログを利用する

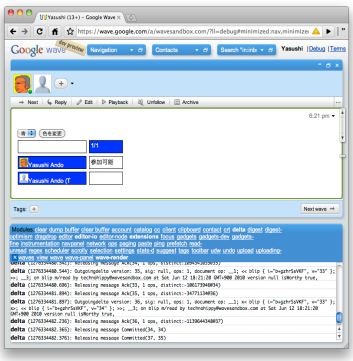
Wave サンドボックスの機能でガジェットのデバッグに特に有用なのが debug log です。ただし、これを表示させるためには少しだけハックが必要です。URL の最後の"/"と"#"との間にパラメータとして ll=debug を追加してリロードしてください。

- https://wave.google.com/wave/**?ll=debug**#restored:wave:...

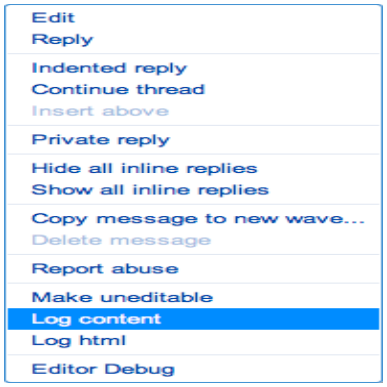
リロード後に Debug メニューを開くと一番上に Show debug log というメニューアイテムが追加されているはずです。



クリックしてデバッグコンソールを表示してみましょう。このログには `wave.log` 関数を使用してガジェットからメッセージを表示することができます。ガジェットのいわゆる print デバッグに活用してください。



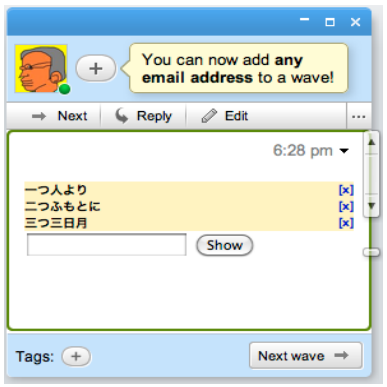
なお、ll=debug パラメータを指定すると、エディタメニューにもログ出力するためのアイテムがいくつか追加されます。



4.2.9.5　デバッグメッセージの表示

先ほどのデバッグログは便利ですが残念ながら Wave サンドボックスでしか使えません。通常の Wave でデバッグメッセージを表示するにはどうすればいいのでしょうか。最も簡単なのは `console.log` 関数を使うことです。`console.log` 関数はほとんどのモダンブラウザでサポートされている関数で、`window.alert` 関数とは異なり、処理をブロックせずにログを出力できます。

そしてもう一つ、ガジェットでのログの出力に利用できるが `minimessage` フィーチャーです。MiniMessage は閉じるボタン [x] がついた小さなメッセージボックスで、wave ガジェットでは次のように表示されます。



上のガジェットのソースコードは次のようになります。

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <Module>
3    <ModulePrefs title="Minimessage Gadget" height="100">
4      <Require feature="wave" />
5      <Require feature="minimessage" />
6    </ModulePrefs>
7    <Content type="html">
8      <![CDATA[
```

```
 9        <input id="msg" />
10        <button onclick="showMessage()">Show</button>
11        <script>
12        var minimsg = new gadgets.MiniMessage(__MODULE_ID__);
13        function showMessage() {
14          var message = document.getElementById('msg').value;
15          minimsg.createDismissibleMessage(message);
16        }
17        </script>
18      ]]>
19    </Content>
20  </Module>
```

gadgets.MiniMessage オブジェクトをグローバルに保持しておき、必要な時に createDismissibleMessage 関数を使ってメッセージを表示します。この例では Show ボタンをクリックされるたびに、テキストフィールドに入力された内容の MiniMessage を表示していますが、もちろんログメッセージを表示するために利用することもできます。MiniMessage オブジェクトには createDismissibleMessage 関数のほかにも便利な関数がありますので、必要であればドキュメント[*10]を参照してください。

4.2.10 おわりに

ガジェットはロボットと比べるとできることが非常に限られています。しかしその代わりに、必要なものは XML ファイルとそれを公開する環境だけ、実際に作成するにあたっての技術的なハードルもかなり低めです。公開する環境については、Google Code[*11]や GitHub[*12]の Gist など無料で簡単に利用できるサービスもたくさんあります。Wave を使っていて何か不便に感じ、こういう機能が欲しいと思うことがあれば、それがガジェットを作るいい機会です。ぜひ本書を参考にその不満を解消するガジェットを作って、公開していただければと思います。

4.2.11 API リファレンス

gadgets.io オブジェクト

本オブジェクトに含まれているのは外部コンテンツを使用するための関数です。

encodeValues(object)　引数として受け取ったオブジェクトからエンコードされたリクエストパラメータを生成します。例えば encodeValues({key1:"値 1", key2:"値 2"}) は"key1=%E5%80%A41&key2=%E5%80%A42"を返します。

getProxyUrl(url, options)　ガジェットから makeRequest を通じて外部のコンテンツを要求されると、HTTP リクエストはコンテナが代理で実行し、その際レスポンスが

[*10] http://code.google.com/intl/ja/apis/gadgets/docs/ui.html#Mini
[*11] http://code.google.com/intl/ja/projecthosting/
[*12] https://github.com/

キャッシュされます。getProxyUrl 関数を利用すると、コンテナにキャッシュが存在する場合はそちらを利用することでリクエストにかかる時間を短縮できます。

makeRequest(url, callback, options)　非同期に URL をリクエストして、レスポンスをコールバック関数に渡します。

gadgets.json オブジェクト

本オブジェクトはその名の通り JSON を扱うための関数を保持しています。

parse(string)　JSON をパースして JavaScript オブジェクトを返します。

stringify(object)　JavaScript オブジェクトを JSON 形式の文字列にします。

gadgets.util オブジェクト

gadgets.io、gadgets.json に含まれないユーティリティ関数群を持つオブジェクトです。

escapeString(string)　文字列に含まれる一部の文字を HTML エンティティに変換します。変換対象の文字は \n, \r, ", ', <, >, Unicode line separator, Unicode paragraph separator です。

getFeatureParameters(feature)　指定したフィーチャーのパラメータを取得します。

hasFeature(feature)　ガジェットのコンテナが指定されたフィーチャーをサポートしているかどうかを返します。

registerOnLoadHandler(callback)　ガジェットがロードされた時に実行されるハンドラ関数を登録します。ガジェットでは window.onLoad 関数が使用できないので、初期化にはこの関数を使う必要がある点に注意してください。

unescapeString(string)　escapeString 関数の逆変換です。

wave オブジェクト

getHost()　このガジェットを貼り付けた参加者を返します。ただし、その参加者がまだ wave に参加しているという保証はありません

getMode()　ガジェットのモード（編集モード・プレビューモードなど）を返します

getParticipantById(id)　指定された id を持つ Participant オブジェクトを返します

getParticipants()　Wave の参加者の配列を返します

getState()　wave.State オブジェクトを返します

getTime()　"gadget time"を返します。これはプレイバックモードの時はプレイバックフレームの時間で、それ以外は現在時刻になります

getViewer()　ガジェットが見ているユーザーに対応する Participant オブジェクトを返します

getWaveId()　シリアライズされた Wave ID を返します。不明な場合は null を返します

isInWaveContainer() ガジェットが wave コンテナ上で動作しているかどうかを返します

log(message) コンテナにログメッセージを出力するように要求します。使い方は 121 ページを参照してください

setModeCallback(callback, context) モードが変更されたときに呼び出されるコールバック関数を登録します。第二引数にはコールバック関数内で this としてアクセスされるオブジェクトを渡します。`setModeCallback` 関数呼び出しとコールバック関数が同じコンテキストで実行される場合は、第二引数は省略できます。

setParticipantCallback(callback, context) 参加者が変化したときに呼び出されるコールバック関数を登録します。参加者情報がすでに読み込み終わっているときは、コールバック関数は現在の参加者情報を伴ってすぐに実行されます。コールバック関数は一つしか登録できず、この関数が複数回呼ばれた場合は、最後に設定されたコールバック関数だけが有効になります

setPrivateStateCallback(callback, context) ガジェットのプライベート状態が変化したときに呼び出されるコールバックです。`setStateCallback` 関数と同様の動作をしますが、対象はプライベート状態になります

setStateCallback(callback, context) カジェットの共有状態が変化したときに呼び出されるコールバック関数を登録します。共有状態がすでに読み込み終わっているときは、コールバック関数は現在のガジェットの共有状態を伴ってすぐに実行されます。コールバック関数は一つしか登録できず、この関数が複数回呼ばれた場合は、最後に設定されたコールバック関数だけが有効になります

wave.Mode オブジェクト

DIFF_ON_OPEN ガジェットを含む blip をユーザーが最後に開いた後で変更がありました

EDIT ガジェットを含む blip が編集中です

PLAYBACK ガジェットを含む blip がプレイバック中です

UNKNOWN ガジェットを含む blip のモードが不明です

VIEW ガジェットを含む blip が閲覧中です

wave.Participant クラス

Wave の参加者情報を表すためのクラスです。

getDisplayName() 参加者の表示名を取得します

getId() 参加者の ID を取得します

getThumbnailUrl() 参加者のサムネイル画像の URL を取得します

wave.State クラス

ガジェットの共有状態を表すクラスです。

get(key, default)　共有状態からキーに対応する値を取り出します。第二引数はオプションで、値が設定されていない場合に返される値です。現在のところ常に文字列を返しますが、将来的にはあらゆる種類のオブジェクトを返すようになります

getKeys()　共有状態に登録されているキー一覧を取得します

reset()　共有状態のすべてのキー・値ペアを削除します

submitDelta(delta)　共有状態を更新します。更新は非同期に行われるので、呼び出し直後に完了している保証はありません。あるキーの値が null の場合はそのキーを削除し、そうでなければ指定された値を新しく登録、または更新します

submitValue(key, value)　キー・値ペアをひとつだけ更新する場合に使用します。詳細は `submitDelta` の説明を参照して下さい

toString()　デバッグ用の文字列表現を返します

wave.ui オブジェクト

本オブジェクトに含まれるのはガジェット内で wave のルック&フィールを設定するための関数です。

loadCss()　ガジェット内で、フォント・リンク・ボタン・ダイアログ・フレームなどについて、wave らしい見た目を実現するための CSS をロードします

makeButton(target)　引数として受け取ったボタン要素の見た目を wave らしく変更します

makeDialog(target, title)　引数として受け取った要素を wave らしい見た目のダイアログに変換します。ただし、現在のところ中央寄せされたボックスが表示されるだけです。右上にあるクローズボタンはデフォルトでは何も実行しません

makeFrame(target)　引数として受け取った要素を wave らしい見た目のフレームに変換します

wave.util オブジェクト

printJson(obj, pretty, tabs)　JSON を文字列で出力します。第二引数で出力形式を指定できます

4.3 ロボット API

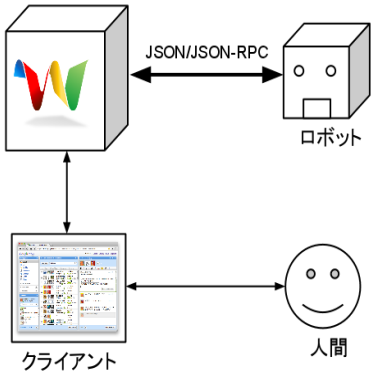
4.3.1 はじめに

ロボットとは wave の参加者として振舞うことができるプログラムのことです。ロボットを参加者として wave に追加すると、そのロボットは自分が参加する wave に対して人間の参加者とほぼ同じこと、具体的には次のようなことができます。

- 会話の内容を読み取る
- 会話の内容を書き換える

- 会話に返答する
- 参加者を追加する
- ガジェットを追加する
- ガジェットの状態を読み取る
- ガジェットの状態を書き換える
- 新しい wave を開始する

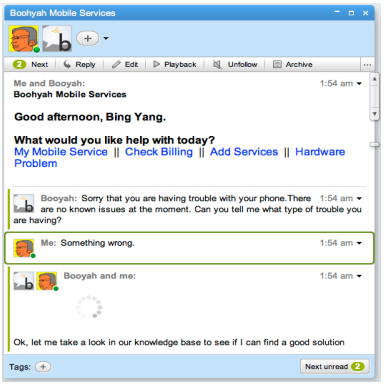
ロボットはサーバー上（ほとんどの場合 Google App Engine 上）で動作し、人間の参加者のようにクライアントを通じて wave サーバーとやりとりするのではなく、wave サーバーと専用のプロトコルで直接情報をやりとりします。[*13]



ロボットは単なるサーバーサイドプログラムであるため、ロボット API に含まれる機能というわけではありませんが、次のようなことも可能です。

- 外部のサービスと連携する
- wave の情報を外部 DB に保存する

ロボット API を利用すれば、例えばユーザーと wave 上で勝手に会話する人工無脳、チェスガジェット貼りつけて対戦の相手をしてくれるチェスロボットといった参加者として振舞うプログラムから、バグトラッキングシステムを定期的にチェックしてバグレポートが届いたときに wave で通知してくれるロボットのように外部サービスとのゲートウェイとして働くプログラムまで、さまざまな機能を wave 上で実現することができます。



[*13] ロボットサーバーと wave サーバーが具体的にどのようなプロトコルでやり取りするかの詳細については第 5.4.3 節で説明します

注意

Python

現在のところ、wave ロボット用の公式のライブラリは Python 版と Java 版しか存在しません[*14]。本節での説明はコード例を短く済ますため Python 版を使用します。API を使用して実現可能なことが何かという点に関しては Python 版も Java 版も大きく違いません。Java 版に興味がある方も、まず「何ができるか」に注目して本節を読み進めてみてください。本節で登場するサンプルコードの Java 版に関してはサポートサイト[*15]で公開しています。初めに Python 版に関する説明を一読し、どういうことをするロボットなのかを理解した後で、Java 版のサンプルコードに目を通していただければと思います。

Google App Engine

ロボットは GAE 以外のサーバー上でも動作しますが、執筆時点ではまだほとんど例がなく、公式のロボットライブラリも GAE に依存した部分が残っています。実際にロボットを試してみる場合も、自分でサーバーを用意するよりは GAE の方が楽でしょう[*16]。本節では基本的にはロボットは GAE 上で動かすものとして説明を進めます。GAE を使用しないロボットについては 4.3.7 節で簡単に紹介していますので、参考にして下さい。

4.3.2　Google App Engine 超入門

本節の大部分は GAE の利用を前提としているので、まずはじめに GAE の利用方法について簡単に説明しておきます。すでに GAE を利用したことのある人はこの項を飛ばして次に進んで構いません。

4.3.2.1　Google App Engine とは

Google App Engine（GAE）は Google が提供する Web アプリケーションのホスティングサービスです。CPU 利用率や帯域が一定の上限を超えるまでは無料で、Google の強力なインフラを利用できるため急なアクセスの増加にも（プラン次第で）難なく対応できます。GAE の主な特徴としては次のようなものがあります。

- 負荷に応じて自動的にスケールする
- ストレージは SQL DB ではなくデータストア（Key-Value ストア）を使用する

アプリケーション ID .appspot.com というドメイン名を持つプロダクション環境が提供される

- ローカルでアプリを実行できる開発環境が付属している
- 管理コンソールを使用してブラウザから設定やログの確認ができる
- 利用できる言語は現在のところ Python と Java[*17]のみ

[*14] 非公式な GO 言語版も存在します: http://github.com/nf/go-wave-robot-api

[*15] http://wave-book.appspot.com

[*16] もちろん、GAE 外でロボットを動かせるようになったのが最近で、いまさら大きく文章を書き変えたくない、という著者の都合も否定しません

[*17] 実際には Java だけでなく JVM 上で動く言語であれば実行できます。例: JRuby, Scala, Clojure, Groovy など

4.3.2.2 開発環境の準備

それではここから実際に GAE でアプリケーションを作成する準備に移りましょう。GAE での開発にはバージョン 2.5 以上の Python が必要です。開発機上で以下のコマンドを実行してコマンドが見つからないか、またはバージョンが 2.5 未満だった場合は最新版の Python をインストールしておいてください。

```
1  $ python -version
2  Python 2.5.2
```

次に Google App Engine SDK を準備します。下記のサイトにアクセスして開発に使用するプラットフォームに応じた SDK をダウンロードして下さい。なお、日本語ダウンロードページの SDK はまれに古いままのことがあります。ダウンロードは英語ページから行うことを推奨します。

- http://code.google.com/intl/en/appengine/downloads.html

Max OSX と Windows に関してはダウンロードしたインストーラを起動し、その支持に従えばインストールは完了します。それ以外のプラットフォームにはインストーラはありません。ダウンロードした zip を展開し、自分の環境に応じた手順で google_appengine ディレクトリにパスを通してください。次のコマンドが実行できれば SDK のインストールは完了です。

```
1  $ appcfg.py -help
2  Usage: appcfg.py [options] <action>
3  ..後略..
```

4.3.2.3 アプリケーションを作成

GAE での Web アプリケーション開発がどのようなものか理解するために、簡単なアプリケーションを作成して実際にデプロイしてみることにします。ただしこの本の主題はあくまでも Google Wave ですので、SDK の詳細には立ち入りません。ロボットの作成に必要な機能についてのみ都度説明しています。網羅的な説明に興味がある方はデベロッパーガイド[*18]か、他の書籍を参考にしてください。

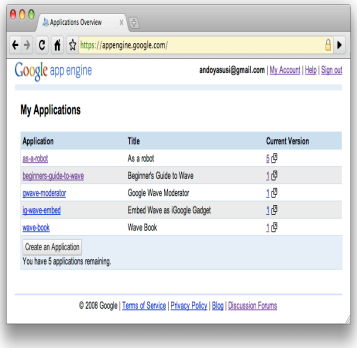
[*18] http://code.google.com/intl/ja/appengine/docs/

アプリケーション ID を登録

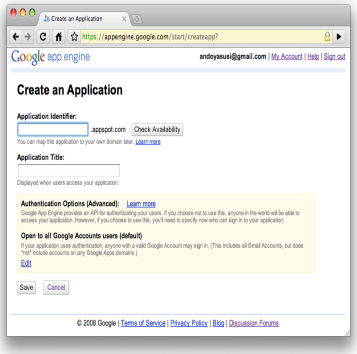
GAE のアプリケーション ID はドメイン名に使用されるため、他のユーザーがまだ使用していないものを選ぶ必要があります。希望する ID が利用できるかどうかは試してみなければ分かりません。また、確認するだけで後で登録しようとしても、その間の期間に誰かに登録されてしまうかもしれません。後でそのアプリケーション ID が不要になれば削除もできますので、GAE で何かを作るときには何をおいても一番最初にアプリケーション ID を登録してしまいましょう。

ブラウザを開き、次の URL にアクセスしてください。

- http://appengine.google.com/



「Create an Application」というボタンをクリックするとアプリケーションの登録画面に遷移します。なお、デフォルトの登録できるアプリケーション数は 10 個です。それを超えてしまっていた場合は「You have 0 applications remaining.」とだけ表示され、「Create an Application」ボタンは表示されません。Google App Engine Group[*19]でお願いして制限を緩めてもらうか、新しく Google アカウントを取得して、そちらでアプリケーションを登録してください。



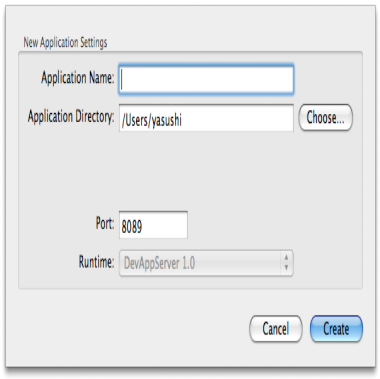
Application Identifier に利用できるのは 6 文字から 30 文字の小文字の英数字とハイフンのみです。「Check Availability」ボタンをクリックして、「Yes, "アプリケーション

[*19] http://groups.google.com/group/google-appengine

ID" is available!」と表示されればその ID は利用可能ですから、「Save」ボタンをクリックして確定してしまいましょう。

アプリケーション生成

アプリケーション ID を取得したら、アプリケーションの作成に移りましょう。Mac OSX ユーザーと Windows ユーザーの方はランチャーを起動して、File メニューから New Application...[*20]を選択します。



Application Name に先ほど登録したアプリケーション ID を入力し、「Create」ボタンをクリックすると指定したディレクトリにアプリケーションの雛形（app.yaml と main.py）が生成されます。ランチャーのないプラットフォームを使用している場合は、作業ディレクトリとしてアプリケーション ID と同名のディレクトリを作成しておいてください。

app.yaml

app.yaml はアプリケーションのメタ情報を記述するためのファイルです。ランチャーでアプリケーションの雛形を生成した場合はすでに次のようなファイルが生成されています。今回は編集の必要はありません。

```
1  application: アプリケーション ID
2  version: 1
3  runtime: python
4  api_version: 1
5
6  handlers:
7  - url: .*
8    script: main.py
```

ランチャーのない環境で開発している方は作業ディレクトリ内にテキストエディタで上記の内容を持つ app.yaml ファイルを作成してください。なお、それぞれの設定項目は以下のような意味です。

[*20] Windows 版は Create New Application...

設定項目	意味
application	アプリケーション ID
version	アプリケーションのバージョン
runtime	python または java
api_version	GAE の API バージョン。現在のところ"1"固定
handlers	リクエストの URL と実行されるスクリプトの組

表 4.2　app.yaml

main.py

main.py は先ほどの app.yaml で、全ての URL（.*）を処理するように設定されたリクエストハンドラです。ランチャーでアプリケーションを生成した場合は、リクエストに対して"Hello world!"と表示するシンプルなハンドラが生成されています。Wave ロボットを作成する場合にはこの部分はライブラリによって隠蔽されていますし、今回はデプロイまでの流れを確認することが主な目的なので細かな説明は省略します。コメントを参照してください。

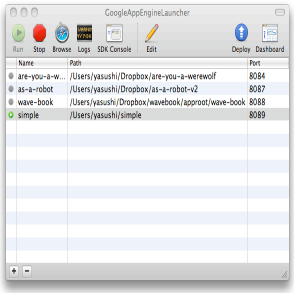
```
 1  #!/usr/bin/env python
 2
 3  # SDK をインポート
 4  from google.appengine.ext import webapp
 5  from google.appengine.ext.webapp import util
 6
 7  # リクエストハンドラクラス
 8  class MainHandler(webapp.RequestHandler):
 9    # GET リクエストに対する応答
10    def get(self):
11      # 'Hello world!' と出力
12      self.response.out.write('Hello world!')
13
14  def main():
15    application = webapp.WSGIApplication(
16      # ルートディレクトリへアクセスがあれば MainHandler を起動
17      [('/', MainHandler)],
18      debug=True)
19    util.run_wsgi_app(application)
20
21  if __name__ == '__main__':
22    main()
```

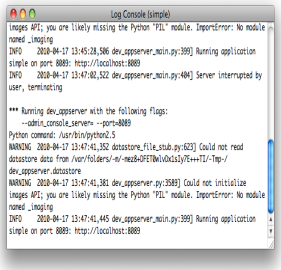
ランチャーのない環境で開発している方はテキストエディタで作業ディレクトリ内に上

記の内容を持つ main.py ファイルを作成してください。

4.3.2.4 ローカル環境での動作確認

これでアプリケーションが完成です（ランチャーを使った人は何もコーディングしていませんが）。さっそく開発機上で動作を確認してみましょう。ランチャーから開発サーバーを起動するには、アプリケーション一覧から先ほど作成したアプリケーションを選択してツールバーの「Run」ボタンをクリックします。右端のアイコンが緑になればアプリケーションは正常に起動しています。一応、エラーが出ていないか「Logs」ボタンをクリックして出力を確認しておきましょう。

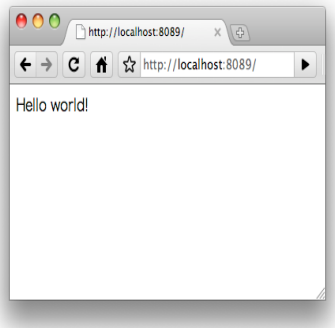




ランチャーを使わずに実行するには作業ディレクトリで次のコマンドを実行します。

```
1  $ dev_appserver.py .
2  INFO     2010-03-07 16:40:26,805 py_zipimport.py:108] zipimporter('/opt/local/Library/Frameworks/Pyt
'simplejson/')
3  INFO     2010-03-07 16:40:27,046 appengine_rpc.py:159] Server:
appengine.google.com
4  INFO     2010-03-07 16:40:27,063 dev_appserver_main.py:399] Running
application アプリケーション ID on port 8080: http://localhost:8080
```

ランチャーの「Browse」ボタンをクリックするか、ログの一番最後に表示されているアプリケーションの URL をアドレスバーに直接打ち込んで、ブラウザで表示してみましょう。



確かに、"Hello, world!"と表示されています。

4.3.2.5　アプリケーションをデプロイ

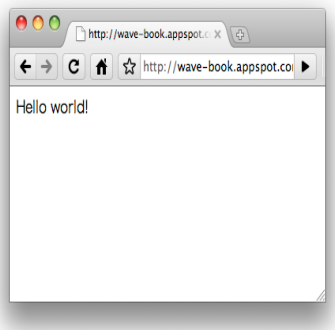
それでは最後にこのすばらしいアプリケーションを全世界に公開しましょう。例によってランチャーを利用している方の作業は非常に簡単です。アプリケーション一覧からアプリケーションを選択して「Deploy」ボタンをクリックしてください。ダイアログが開きますので、Google アカウントのユーザー名とパスワードを入力して「ログイン」ボタンをクリックすればデプロイが始まります。

ランチャーを使わずに実行するには作業ディレクトリで次のコマンドを実行します。

```
1  $ appcfg.py update .
2  Application: アプリケーション ID; version: 1.
3  Server: appengine.google.com.
4  Scanning files on local disk.
5  Initiating update.
6  Email: メールアドレス
7  Password for andyjpn@gmail.com: パスワード
8  ... 以下略 ...
```

途中で Google アカウントとパスワードを聞かれた場合は入力してください。

デプロイが完了したら、ブラウザで http://アプリケーション ID.appspot.com を開いてみましょう。



先ほどローカルで確認した内容が表示されれば正しくデプロイされています。

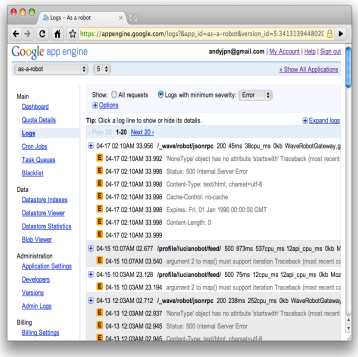
管理コンソール

Web サービスはデプロイしたら完成ではありません。むしろその先が本番です。本番環境でのテストも必要ですし、リリース後もアクセスログやエラーログの解析など、必要な作業は多いでしょう。GAE の管理コンソールを使うとそのような作業をブラウザ上から行うことができます。管理コンソールから可能な作業には、例えば次のようなものがあります。

- アクセスログの確認
- エラーログの確認
- リソース利用率の確認
- データストア管理
- デフォルトバージョンの変更

上記はいずれも wave ロボットの開発でよく使うことになると思いますが、中でも重要なのはやはりエラーログの確認でしょう。GAE にはローカルで動作する開発サーバーが付属していますが、wave ロボットは wave サーバーからのリクエストを処理するものであるため、残念ながらその開発サーバー上で動かすことは今のところできません[*21]。wave ロボットの動作を確認するには実際にデプロイして wave に参加させるしかなく、問題があれば管理コンソールでエラーログを確認することになります。

それでは管理コンソールで GAE アプリケーションのログを確認してみましょう。ブラウザで http://appengine.google.com を開き、ログを確認したいアプリケーション ID をクリックしてください。サイドバーに「Logs」という項目がありますので、それをクリックするとロボットのログが確認できます。



ロボット起動時の Debug パラメータを True にしておけば、この画面にはロボットが受け取る wave の情報や、ロボットが wave サーバーへ送り返すオペレーションなど、全ての入出力が表示されています。もちろん開発者自身がデバッグログを書きだすこともできますので、ロボットの動作に問題があるときは、まず管理コンソールでログを確認しましょう。

[*21] wave ロボットのプロフィールやアイコンを表示させて一部の文法エラーを確認すること、ダンプした JSON を POST して単一の処理を確認することは可能です

また、もう一つ wave ロボットの開発に便利なのがデフォルトバージョンの変更です。GAE アプリケーションは app.yaml（131 ページ）の `version` でバージョンを設定でき、特定バージョンのアプリケーションには http://[バージョン番号].latest.[アプリケーション ID].appspot.com という URL でアクセスすることができます。デフォルトバージョンはそのバージョン指定がないとき、つまり http://[アプリケーション ID].appspot.com にアクセスされたときに実行されるバージョンです。GAE の既存アプリケーションに大きな機能追加をするときには、開発バージョンとデフォルトバージョンを使い分けるのが常道です。

wave ロボットも [バージョン番号].latest.[アプリケーション ID]@appspot.com またはその短縮版 [アプリケーション ID]#[バージョン番号]@appspot.com というアドレスでバージョンを指定して実行できます。先にも書いたように wave ロボットは開発環境ではほとんどテストができません。特に機能追加の際には、このバージョン切り替え機能を使って本番環境上で動作を確認してから、デフォルトバージョンを切り替えてユーザーに公開しましょう。

4.3.3 はじめての Wave ロボット

GAE を使ったアプリケーション開発の流れを一通り確認しました。ここからはロボット API を利用して wave 上で動く簡単なチャットボットを作っていきます。まずは大まかな流れを確認するために、ロボット自身が wave に参加したときに他の参加者に挨拶する簡単なロボットを作りましょう。ここで作成するロボットを土台にして後の節で徐々に高度な機能を追加していくことにします。

ただし、ロボット API は wave の持つ 4 種類の API の中で最大のもので、紙面の都合上全ての API を紹介することはできません。また、これは API 全体に言えることですが、wave は現在も活発に開発が進められていて、執筆時点より機能が追加されていたり仕様が変更されていることも十分に考えられます。自分でロボットを作りはじめる前に、なるべく公式ドキュメント[*22]に目を通しておくようにしてください。

4.3.3.1 ロボット作成準備

ロボット一つに付き GAE のアプリケーション ID が一つ必要になります[*23]。先ほどの GAE の説明に従って既にアプリケーション ID を一つ取得している人は以降もそれを利用して構いません。まだ取得していない人は先に進む前に管理コンソールでこれから作成するサンプル用のアプリケーション ID を取得して、アプリケーションの雛形を作成しておいてください。

まずは作業ディレクトリ[*24]にロボットクライアントライブラリをダウンロードしましょう。svn コマンドが使えて、ライブラリの最新バージョンを追いかけたい方は作業ディレクトリで次のコマンドを実行してください。

```
1  $ svn checkout http://wave-robot-python-client.googlecode.com/svn/trunk/src/w
```

[*22] http://wave-robot-python-client.googlecode.com/svn/trunk/pydocs/index.html

[*23] 拙作 appengine_multi_robot_runner を使うと一つのアプリケーション ID で複数のロボットを動かすことができます。詳細は http://github.com/technohippy/appengine_multi_robot_runner/ を参照してください

[*24] app.yaml の存在するディレクトリ

waveapi

開発環境に svn が入っていない方や最新を追いかける必要を感じない方は下の URL から zip ファイルをダウンロードして、作業ディレクトリに展開しても構いません。ただし最新バージョンでは修正されているバグが zip 版にはまだ残っている場合がありますので注意してください。

- http://code.google.com/p/wave-robot-python-client/downloads/list

先の GAE の節の手順に沿って作業を進めてきた場合は、waveapi 取得または展開後の作業ディレクトリは次のような構成になっているはずです。

```
1  .
2  `- アプリケーション ID
3       |- app.yaml
4       |- index.yaml
5       |- main.py
6       `- waveapi
7            |- __init__.py
8            |- appengine_robot_runner.py
9            |- robot.py
10           |- ... 以下略...
```

*25

4.3.3.2 アプリケーション設定

まずはアプリケーションが wave サーバーからのリクエストを処理できるように app.yaml を編集しましょう。

```
1  application: アプリケーション ID
2  version: 1
3  runtime: python
4  api_version: 1
5
6  handlers:
7  - url: /_wave/.*
8    script: アプリケーション ID.py
9  - url: /assets
10   static_dir: assets
11 - url: .*
12   script: main.py
```

wave サーバーからロボットサーバーへのリクエストは全て/_wave/以下に届きますので、それらを全てアプリケーション ID.py というスクリプトで処理するように設定しています。また handlers の下に static_dir という見慣れない項目がありますが、これ

*25 index.yaml についてはこれまで説明していませんが、検索インデックスの定義ファイルで基本的には GAE が自動生成するものです。詳細については GAE のドキュメントを参照してください

は/assets にアクセスがあったときに作業ディレクトリ下の assets ディレクトリにあるファイルをそのまま返すように指示するものです。本書の例ではプロフィール画像やロボットが使用するガジェットは全てこのディレクトリの下に置くことにします。そして最後の 2 行ではそれら以外のリクエストを全て main.py で処理するように設定しています。今回はロボットのプロフィール URL として http://アプリケーション ID.appspot.com を使用するようにしたため、GAE の例で使用したこの設定を残してあります。

4.3.3.3 ハローロボット

それではロボットの実装を開始しましょう。作業ディレクトリにアプリケーション ID.py というファイルを作成し、次のように入力してください。（もちろんコメント行[*26]を入力する必要はありません）

```
1  # -*- coding: utf-8 -*-
2
3  # 利用するモジュールを読込
4  from waveapi import events
5  from waveapi import robot
6  from waveapi import appengine_robot_runner
7  import logging
8
9  # イベントハンドラ定義
10 def SayHello(event, wavelet):
11   # ログメッセージを出力
12   logging.info(u'SayHello 実行')
13   # 新しく wavelet を追加して挨拶を表示
14   wavelet.reply(u' こんにちは。Wave に呼んでくれてありがとう')
15
16 # ライブラリとしてではなく、直接に実行されたとき処理を実行
17 if __name__ == '__main__':
18   # プロフィール情報を指定してロボット定義
19   robot = robot.Robot(u' ロボット名',
20     image_url='http://アプリケーション ID.appspot.com/assets/icon.png',
21     profile_url='http://アプリケーション ID.appspot.com/')
22   # ロボットにイベントハンドラを登録
23   #   ロボットが参加者として Wave に追加（WaveletSelfAdded）されたときに
24   #   SayHello 関数が実行される
25   robot.register_handler(events.WaveletSelfAdded, SayHello)
26   # GAE を使用してロボットの処理を開始
27   appengine_robot_runner.run(robot)
```

標準的なロボットの実装の大まかな流れは次のようになります。

[*26] #で開始している行

1. イベントハンドラを定義する
2. ロボットクラスに必要なプロパティを渡してインスタンス化する
3. ロボットインスタンスにイベントハンドラを登録する
4. appengine_robot_runner を使用してロボットの処理を開始する

ソースコードの詳細については後ほど説明します。まずはロボットを実際に動かしてみましょう。アプリケーション ID.py が完成したら、静的ファイル用の assets ディレクトリを作成し、そこに好きな PNG 画像を icon.png に改名して置いてください。それがロボットのアイコン画像になります。

4.3.3.4 ローカルでの動作確認

ロボットを GAE にデプロイする前に開発機上で簡単な動作確認を行っておきましょう。wave のドキュメントへの操作をローカルで確認することは難しいのですが、とりあえず capabilities.xml という自動生成される設定ファイルを表示させて最低限の確認とします。アプリケーションをローカルで実行しポート番号を確認したら、ブラウザで http://localhost:[ポート番号]/_wave/capabilities.xml を表示してください。ライブラリのバージョンによって XML の細かい内容は異なるかもしれませんが、エラーにならずに次のような内容が表示されれば、単純な文法エラーはないと考えていいでしょう。

```
<w:robot xmlns:w="http://wave.google.com/extensions/robots/1.0">
<w:version>0x33050c9L</w:version>
<w:protocolversion>0.2</w:protocolversion>
<w:capabilities>
  <w:capability name="OPERATION_ERROR"/>

  <w:capability name="WAVELET_SELF_ADDED"/>
</w:capabilities>
</w:robot>
```

なお確認用の URL にアクセスするとブラウザによってはファイルダウンロードが始まったり、XML タグ以外のテキストだけが表示されたりするかもしれません。ダウンロードされた場合はそのファイルの内容を、妙なテキストが表示された場合はブラウザに応じた方法で「ソースを表示」してください。

capabilities.xml

ところで今確認に使用した capabilities.xml とはなんなのでしょう。ロボット動作確認用の無意味なファイル・・・ではもちろんありません。capabilities.xml はその名の通りロボットの能力（capability）、具体的にはロボットが反応するイベントを指定するためのファイルです。例えば先の例からはこのロボットが OPERATION_ERROR イベントと WAVELET_SELF_ADDED イベントに反応することが分かります。wave サーバーはロボットが参加者として追加されるとまずこの capabilities.xml を確認して、どのイベントをロボットサーバーに通知するか決定しています。なお、実際には capabilities.xml はクライアントライブラリによって自動的に生成され、開発者が直接編集することはありません。

4.3.3.5　Google Wave 上の動作確認

ローカルでの動作が確認できたので実際に GAE にデプロイして Google Wave 上でロボットを呼び出してみましょう。GAE の節で説明した通りランチャーを使用している場合は「Deploy」ボタンをクリック。コマンドラインを使用する場合は作業ディレクトリで次のコマンドを実行してください。

```
1 $ appcfg.py update .
2 Application: アプリケーション ID; version: 1.
3 Server: appengine.google.com.
4 Scanning files on local disk.
5 Initiating update.
6 Email: メールアドレス
7 Password for andyjpn@gmail.com: パスワード
8 ... 以下略 ...
```

デプロイが完了したら Google Wave を開いて新しい wave を開始し、ロボットを参加者として追加します。ロボットのアドレスは「アプリケーション ID@appspot.com」です。次のようにロボットからのメッセージが表示されればロボットは正しく動作しています。もし期待通りに表示されていないときには、管理コンソールでログを確認してみてください。

初めての wave ロボットが無事完成しました。どうでしょう？ここでは wave ロボット開発の全体的な流れを一通り見ることが目的だったので詳細には立ち入っていませんが、それでも単純なチャットボット程度ならすぐに作れそうな気がしてきたのではないでしょうか。

4.3.3.6　データモデル

ロボットの実装の説明に移る前に、ロボットが扱うデータモデルの内で代表的なものとそれらの関係を簡単に紹介します。

Wavelet　参加者（Participants）と一連の発言（Blips）を管理するデータモデルです

Participants　Wavelet の参加者一覧を表すデータモデルです

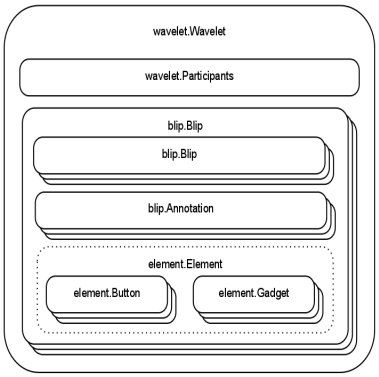
Blip　一つの発言を管理するデータモデルです。Blip は複数の子 Blip を持つことができ、全体として木構造を持ちます。また Blip は子 Blip 以外にもメタ情報（Annotation）

や特殊な表示要素（Element）を持つことがあります

Annotation　Blip のある範囲に設定されるキーと値のペアです。Blip の特定の範囲に設定される文字色やサイズなどの表示上の設定や、リンクなどの情報を管理します

Element　テキスト以外の表示要素です。例えばボタンや画像、ガジェットなどを表します

上に挙げたモデルの関係をまとめると次のようになります。



非常に大雑把に言えば、ロボットとは wave サーバーからこれらの情報を受け取り、新規に作成した要素を追加したり既存の要素を変更したりして、wave サーバーに送り返すプログラムのことです。

4.3.4 受け身なロボットと積極的なロボット

本節では GAE 上で動作するロボットを 2 つ作成します[*27]。それらのロボットはそれぞれタイプが異なり、一つが wave 内で発生するイベントに応じて動作するもの、もう一方がそれ以外のきっかけで動作するものです。前者は例えば先ほど作成したような自身が wave に追加されたときに発言するロボット、後者は例えば cron で定期的に起動して wave に情報を送るロボットや、他のウェブサービスにメッセージが登録されたときに起動して wave に送るロボットなど、アイデアによってさまざまなロボットがありえます。

両者が特に異なるのは wave の情報をどうやって取得するかという部分です。前者、wave 内のイベントを起点にしたときには wave の情報はそのイベントに付随して送られてきますが、後者の場合は起動された後でロボットサーバーから wave サーバーにリクエストを送り能動的に wave の情報を取得しなければいけません。そのため後者に使用する API は特に Active API と呼ばれたりします。

	イベント駆動型ロボット	アクティブロボット
動作の起点	Wave の内部イベント	Wave とは関係のないイベント
Wave の情報	イベントに付随する	自ら API を発行して取得する

表 4.3　二種類のロボットの比較

ただしイベント駆動の API とアクティブな API は排他的なものではありません。例

[*27] 加えて、GAE を使わない簡単なロボットを一つ

えばある wave 内で発生したイベントに反応して、全く別の wave を書き換えるようなロボットは両方の API を使用することになります。

4.3.5　イベント駆動型のロボット

まずは最初に作成したあいさつロボットのソースコードを説明し、その後そのロボットを拡張しながらイベント駆動型のロボットを作成する際に使用される API について説明します。機能拡張の説明ではソースコードは変更のあった部分だけを掲載しています。最終的なロボットのソースコードは 171 ページにありますので、適宜参照して下さい。

4.3.5.1　全体的な流れ

先ほども書きましたが基本的なイベント駆動型のロボットの実装は次のような流れになります。

1. 必要なモジュールのインポート
2. 各種イベントハンドラ[*28]を定義
3. プロフィール情報を指定してロボットを初期化
4. ロボットにイベントハンドラを登録
5. appengine_robot_runner を使用してロボットの処理を開始

あいさつロボットはイベント駆動型のロボットですから上記の流れにそって実装されています。ソースコードを上の項目に分けて順番に見ていきましょう。

必要なモジュールのインポート

```
1  # -*- coding: utf-8 -*-
2
3  # 利用するモジュールを読込
4  from waveapi import events
5  from waveapi import robot
6  from waveapi import appengine_robot_runner
7  import logging
```

一行目はソースコードが UTF-8 で書かれていることを指定するものです。python ではこの指定と合わせて u'...' という形式の UTF-8 文字列リテラルを使うことで日本語文字列が利用できます。

4 行目以降は利用するモジュールを読み込んでいます。events, robot, appengine_robot_runner の 3 つはおそらくほとんど全てのロボットで必要でしょう。イベントハンドラで複雑な処理をする場合はこれら以外にもモジュールをインポートする必要があるかもしれません。今回は先の 3 つに加えて、GAE にログを書きこむための logging モジュールをインポートしています。

[*28] 規定された引数を取る単なる関数です

各種イベントハンドラを定義

```
1  # イベントハンドラ定義
2  def SayHello(event, wavelet):
3    # ログメッセージを出力
4    logging.info(u'SayHello 実行')
5    # 新しく blip を追加して挨拶を表示
6    wavelet.reply(u' こんにちは。Wave に呼んでくれてありがとう')
```

イベントハンドラは、第一引数にイベントオブジェクト、第二引数に引数にイベントソースである wavelet を取る単なる関数です。引数の数さえ間違っていなければ関数名は任意です。ロボットの初期化の際に特定のイベントと、ここで定義したイベントハンドラを関連付けます。

今回のイベントハンドラでは、Wavelet#reply メソッドを使って、返信メッセージが書き込まれた新しい blip を wave の最後に追加しています。

プロフィール情報を指定してロボットを初期化

```
1  # ライブラリとしてではなく、直接に実行されたとき処理を実行
2  if __name__ == '__main__':
3    # プロフィール情報を指定してロボット定義
4    robot = robot.Robot(u' ロボット名',
5      image_url='http://アプリケーション ID.appspot.com/assets/icon.png',
6      profile_url='http://アプリケーション ID.appspot.com/')
```

if __name__ == '__main__':はこのモジュールが（他のファイルにインポートされたのではなく）直接呼び出されたときに、続くブロックを実行するという条件文ですが、よく分からない場合はいわゆる「お約束」と考えて構いません。

robot.Robot クラスは名前の通り wave ロボットを表すクラスで、ロボットのプロフィール情報やイベントハンドラを管理します。robot.Robot クラスのリファレンスは 4.3.9.1 節を参照してください。ここではコンストラクタが次のような引数をとることだけ把握していれば十分でしょう。なお、第一引数のロボットの名前以外は省略可能です。

1. ロボットの名前
2. (オプション) ロボットのプロフィール画像 URL
3. (オプション) ロボットのプロフィール URL

これらの情報は参加者アイコンをクリックしたときに表示されるプロフィール画面で確認できます。

ロボットにイベントハンドラを登録

```
# ロボットにイベントハンドラを登録
#   ロボットが参加者として Wave に追加（WaveletSelfAdded）されたときに
#   SayHello 関数が実行される
robot.register_handler(events.WaveletSelfAdded, SayHello)
```

Robot#register_handler メソッドでイベントとイベントハンドラを関連付けることができます。Python では関数名が関数オブジェクトを表しているので第二引数には事前に定義したイベントハンドラ関数の名前を渡してください。今回のロボットが処理するイベントは events.WaveletSelfAdded ひとつだけですが、処理するイベントが複数ある場合には当然#register_handler は複数回呼び出されることになります。

appengine_robot_runner を使用してロボットの処理を開始

```
# GAE を使用してロボットの処理を開始
appengine_robot_runner.run(robot)
```

最後に appengine_robot_runner.run 関数に設定が完了したロボットオブジェクトを渡すことで処理を開始します。もちろん、この関数は GAE を利用しているときに使用するもので、GAE を使わない場合については 4.3.7 節を参照してください。

4.3.5.2　イベントクラス

イベント駆動のロボットを作るにあたっては、そもそもどのようなイベントをハンドリングできるのか知っておくことが肝要です。あいさつロボットの機能追加に入る前に、ロボット API に含まれるイベントクラスを一通り確認しておきましょう。今、クラスと書きましたが、イベントハンドラに渡されるイベントはイベントの種類を区別するための単なる定数などではなく、そのイベントの処理に必要なプロパティを保持しているイベントクラスのインスタンスです。

全てのイベントクラスは events.Event クラスを継承していて、その events.Event クラスで定義されている次のプロパティは全イベントオブジェクトで利用できます。

modified_by　イベントを発生させた参加者 ID

timestamp　イベントがサーバーに届いた時間

type　イベントのタイプ。例えば events.BlipSubmitted なら"BLIP_SUBMITTED"になります

raw_data　wave サーバーから受け取った JSON

properties　特殊なプロパティを保持するための辞書。各イベントに特有のプロパティは通常は具象サブクラスの属性として保持されますが、実験的なプロパティは Event クラスの持つこの辞書に保持されることがあります

blip_id　イベントに関連する blip の blip_id か、またはイベントに関連する wavelet のルート blip の blip_id

blip　もし存在すれば、blip_id が示す blip そのもの

proxying_for　もし存在すれば、イベントを発生させたロボットの Proxying For ID。proxying_for については第 4.3.6.7 節を参照してください

ロボットが処理できるイベントの一覧は表 4.4 にまとめています。イベント駆動型のロボットにできることを把握するために、一度は目を通しておいてください。イベントごとの特殊な属性は名前を上げるに留めていますが、おそらく推測可能でしょう。詳細な情報が必要なときは API リファレンス[*29]を参照してください。

[*29] http://wave-robot-python-client.googlecode.com/svn/trunk/pydocs/index.html#module-events

イベントクラス	独自の属性
説明	
events.AnnotatedTextChanged	name, value
アノテーションを持つテキストが変更されたときに発生するイベントです。具体的にはテキストの色やリンクなどの情報が変更されたときに発生します	
events.BlipContributorsChanged	contributors_added, contributors_removed
blip のコントリビューターが変更されたときに発生するイベントです。具体的にはある発言を、発言者以外の人が修正したときなどに発生します	
events.BlipSubmitted	-
blip がサブミットされたときに発生するイベントです。具体的には編集モードを「Done」ボタンで終了したときに発生します	
events.DocumentChanged	-
ドキュメントが変更されたときに発生するイベントです。ほぼ、一文字打つごとに発生します。トラフィックが大きくなるので注意して使用し、必要に応じて適切なフィルタを設定してください	
events.FormButtonClicked	button_name
フォームボタンがクリックされたときに発生するイベントです。なお、ここでいうフォームボタンとはロボットによって動的に追加されたボタンのことで、Blip の編集を確定する「Done」ボタンのことではありません。注意してください	
events.GadgetStateChaged	index, old_state
ガジェットの状態が変更されたときに発生するイベントです	
events.OperationError	operation_id, error_message
ロボットサーバーから送られた処理に失敗したときに発生するイベントです	
events.WaveletBlipCreated	new_blip_id, new_blip
新しく blip が作成されたときに発生するイベントです	
events.WaveletBlipRemoved	removed_blip_id, removed_blip
blip が削除されたときに発生するイベントです	
events.WaveletCreated	message
新しく wavelet が作成されたときに発生するイベントです。ロボットが新しい wavelet を作成し、さらにその wavelet を初期化可能なときにだけ発生します。他の参加者が作成した wavelet は自身がその wavelet に追加されるまで見ることはできず、そのときには `WaveletSelfAdded` イベントが発生します	
events.WaveletFetched	message
Wavelet がフェッチされたときに発生するイベントです。ロボットが他の wavelet を参照するリクエストを送った後で発生します。ロボットは事前にその wavelet に参加していなければいけません	
events.WaveletParticipantsChanged	participants_added, participants_removed
wave の参加者が変更されたときに発生するイベントです。参加者が追加されたときと、除かれたときの両方で発生します	
events.WaveletSelfAdded	-
Wavelet の参加者としてロボット自身が追加されたときに発生するイベントです	
events.WaveletSelfRemoved	-
Wavelet の参加者からロボット自身が除かれたされたときに発生するイベントです	
events.WaveletTagsChanged	-

4.3.5.3 新しい参加者に挨拶する

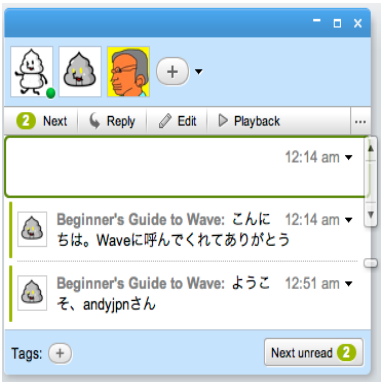
それでは、ここからはサンプルロボットに機能を追加しながら、ロボット API に含まれるさまざまなモジュールの使い方を見ていきます。まずは小手調べとして新しく参加者が追加されたときに、ロボットにその参加者に対して挨拶をさせます。

wave に新しい人が参加してきたときに発生するのは events.WaveletParticipantsChanged です。参加者にあいさつをするイベントハンドラを作成して Robot#register_handler メソッドでイベントと関連付けてください。

```
 1  # 誰かが追加されたときに挨拶
 2  def SayHelloToParticipants(event, wavelet):
 3    # ログメッセージを出力
 4    logging.info(u'SayHelloToParticipants 実行')
 5
 6    # 新しく来た参加者に挨拶
 7    for participant in event.participants_added:
 8      if participant != 'アプリケーション ID@appspot.com':
 9        wavelet.reply(u'ようこそ、%s さん' % GetName(participant))
10
11  # 参加者 ID からドメインを除いたものを取得
12  def GetName(participant_id):
13    return participant_id.split('@')[0]
 1    robot.register_handler(events.WaveletParticipantsChanged,
 2      SayHelloToParticipants)
```

イベントハンドラの第一引数として渡されるのは、events.WaveletParticipantsChanged クラスのインスタンスです。このオブジェクトは participants_added というプロパティ名で新しい参加者 ID のリストを保持しているので、そのリストの要素を一つ一つトラバースして、それぞれの参加者に Wavelet#reply メソッドを使って挨拶しています。ただし、自分自身が追加されたときにも WaveletParticipantsChanged が発生するので、participant が自分の ID と等しい時には処理をスキップしています。

参加者が増えた時の表示は次のようになります。



ここで、もしかすると「一度に追加される参加者は一人だけのはずなのにどうしてリ

ストにする必要があるのか」と疑問に思う方がいるかもしれません。その理由は大きく二つあります。まず現在の Google Wave の UI がそうであるからと言って今後もそうであるとは限らないこと。また仮に Google Wave の UI が未来永劫変わらないとしても、Google 以外による wave クライアントが登場したときに異なる UI を採用する可能性もあります。現時点で参加者を一人づつしか追加できないように制限してしまうことはあまり得策ではないでしょう。しかし本当に重要な理由は次です。

実は現在の Google Wave でも `WaveletParticipantsChanged` イベントに付随してロボットに複数の参加者が送られてくる可能性があります。というのも wave クライアント上で発生したイベントがすぐにロボットに伝えられる保証があるわけではないからです。wave のあらゆる局面、クライアントー wave サーバー間、wave サーバーー wave サーバー間、そして wave サーバーーロボットサーバー間でイベントは常に遅延する可能性があります。そして、その過程でイベントの順番が入れ替わるだけでなく、複数のイベントが一つにまとめられてサーバーに届く可能性さえあります。つまり仮に一人ずつしか参加者を追加できないという UI であっても、どこかの段階で一つにまとめられ、ロボットサーバーに届いたときには複数参加者の追加というイベントに変わっている可能性もあるのです。

ロボットを作成するときには「イベントが必ずしも発生順に届くとは限らない」ということを常に意識しておくようにしましょう。

4.3.5.4　参加者の発言に反応する

一般にチャットボットと言えば、参加者の発言に意味不明な返答をするものと相場は決まっています。今度はロボットが参加者の発言に応じて何らかの反応を返すようにしてみましょう。先のイベント一覧（表 4.4）に目を通した人は「参加者の発言に反応」と聞くと `events.DocumentChanged` を使いたくなるかもしれませんが、実はこのイベントは参加者が一文字入力するたびに発生するものです。発言に対して返事をするだけであれば参加者が入力を完了（submit）したときに処理が実行されれば十分で、それには `events.WaveletBlipCreated` か `events.BlipSubmitted` を使用します。[*30]

上記二つのイベントの違いはその名の通り、前者が blip が新しく作成されたときにだけ発生するイベント、後者はそれに加えて既存の blip に変更があった場合にも発生するイベントです。同時に設定すると blip が作成された時に両方のイベントに紐づいたイベントハンドラが起動してしまいます。意図して行う分には問題ありませんが、うっかり想定していないイベントハンドラが実行されないように注意してください。今回はより汎用的な `events.BlipSubmitted` に反応することにします。

ロボットにどういう反応をさせるかですが、発言に英単語があるときにその単語の意味を聞き返すことにしました。とりたてて意味はありませんが、日本生まれで英語が理解できないロボットということにでもしてください。ソースは次のようになります。先ほどと同様に差分だけを抜き出しています。

今回は発言の一部を抜き出すために正規表現モジュール `re` を利用しますので、最初にインポートしておきます。

[*30] `DocumentChanged` はスペルチェックやリンクの追加など、リアルタイムな処理が必要なときに使用します

```
1  import re
```

発言に含まれる英単語の意味を聞き返すイベントハンドラの実装は次です。

```
1  # 英単語があれば聞き返す。なければ分かったふりをする
2  def MaybeUnderstand(event, wavelet):
3    # ログメッセージを出力
4    logging.info(u'MaybeUnderstand 実行')
5
6    # イベントに紐づいた blip を取得
7    modified_blip = event.blip
8    if len(modified_blip.text.strip()) == 0: return
9
10   # リプライ blip を作成
11   reply_blip = modified_blip.reply()
12
13   # 連続するアルファベットがあるかどうか
14   en_pattern = re.compile(r'[a-zA-Z]{2,}')
15   en_match = en_pattern.search(modified_blip.text)
16   if en_match:
17     # 英単語があれば聞き直す
18     unknown_word = en_match.group()
19     reply_blip.append(u'%s さん、「%s」ってなんですか？'
20       % (GetName(modified_blip.creator), unknown_word))
21   else:
22     # 英単語がなければ分かったふりをする
23     reply_blip.append(u' なるほど。わかります。')
```

最後にその関数を BlipSubmitted イベントのハンドラとしてロボットに登録します。

```
1    robot.register_handler(events.BlipSubmitted, MaybeUnderstand)
```

それではイベントハンドラの解説に移りましょう。イベントハンドラではまずはじめに変更があった blip を取得しています。イベントオブジェクトは第 4.3.5.2 節（p.144）で紹介したように blip というプロパティを持ち、イベントが blip に由来するものであればその blip を、そうでなければルート blip を返します[*31]。BlipSubmitted はもちろん blip に由来するイベントですから、このプロパティで取得できるのは新しく生成された blip、または変更のあった blip です。その後、その blip を元に返信をするための blip を新しく生成して（modified_blip.reply()）、返信の本文を追加（reply_blip.append(...)）するという流れになっています。

さて、前回の参加者への挨拶では Wavelet#reply というメソッドを使用したことを覚えているでしょうか。今回の返信ではそれとは別の Blip#reply メソッドを使用しました。おそらくほとんど全てのロボットで参加者の発言に何らかの返信をする機会が一度はあるでしょう。ここで返信に関連するメソッドをまとめておきます。なお、ここでいう

[*31] イベントクラス名が Wavelet で始まるものは blip 由来のイベントではないのでルート blip を返します。それ以外はイベントの発生元を含む blip を返します

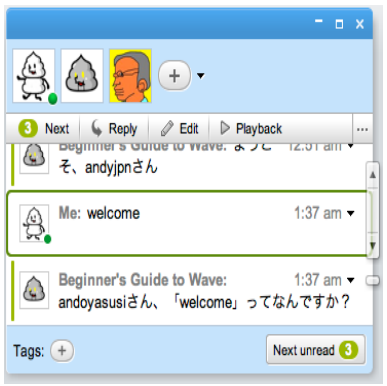
「返信」とは、より実装に即して言えば「新しく blip を作成して追加すること」です。その方法は先の二つに加えてあと一つ、つまり 3 種類あります。[*32]

`Wavelet#reply(content=None)`　このメソッドは、どういうコンテキストで実行されたとしても常に wave の一番最後に blip を追加します。他の二つのメソッドとは異なり、このメソッドだけが引数としてメッセージを渡すことができます。特定の blip に関係しない、会話全体に対するメッセージを返したいときにはこのメソッドが最適です。

`Blip#reply()`　このメソッドは、blip 下部にマウスオーバーして表示されるリプライマークをクリックした際の動作と同じです。つまり、レシーバーとなった blip の直下に新しい blip を作成して返します。このメソッドは引数を取りません。メッセージは返り値の blip を受け取って `append` メソッドで追加してください。特定の blip に対する返答はこのメソッドを使用します。

`Blip#insert_inline_blip(position)`　このメソッドは名前の通り、レシーバーとなる blip の、引数で指定された位置にインライン blip を作成します。クライアントでの操作としては、単語を選択して Enter キーを押下することと同じです。選択された単語の直後に表示/非表示を選択できる blip を挿入します。このメソッドの引数は挿入位置だけで、メッセージの内容を渡すことはできません。メッセージは返り値の blip を受け取って `append` メソッドで追加してください。blip 内の特定の単語・文章に対する返信の他、表示・非表示を切り替えられることを利用してオプショナルな情報を表示するのにも利用できます。

今回の目的は分からない単語を含む特定の blip に対する返答なので `Blip#reply` メソッドを使用しました。実行結果は次のようになります。



4.3.5.5　参加者の発言の表示を変える

ロボットは分からない単語を聞き返すようになりましたが、もし元の発言が長文だとどの単語について質問しているのか分からないかもしれません。そういう事態を避けるために、先ほどのイベントハンドラを改造してロボットに理解できなかった単語の背景色を黄色くしてみましょう。背景色に限らず、表示されているデータになんらかの属性を持た

[*32] ブラウザからの操作だともう一つ、プライベートリプライが可能ですが、これは現在のところロボット API では実現できないようです

せるには blip.Annotation というオブジェクトを使用しますので、まず最初に blip モジュールをインポートしておきます。

```
1  from waveapi import blip
```

発言の一部の背景色を変更するソースは次のようになります。

```
1  def MaybeUnderstand(event, wavelet):
2    # .. 省略..
3    if en_match:
4      # .. 省略..
5
6      # 英単語の背景色を黄色くする
7      blip_ref = modified_blip.first(unknown_word)
8      blip_ref.annotate(blip.Annotation.BACKGROUND_COLOR, 'yellow')
```

たった 2 行の変更ですが、ここに出てくる Annotation と BlipRefs はどちらも非常に重要なオブジェクトです。

Annotation

blip.Annotation は blip の特定の範囲に設定できるキーと値の組です。キーと値は基本的には自由な文字列を設定でき、開発者が好きな用途に利用して構いません。ただし次のアノテーションキーは予約されていて他の用途には使用できません。なお、執筆時点でのロボットライブラリでは予約済みのアノテーションの内、表示用のアノテーション[*33]だけが blip.Annotation **クラス**の定数として定義されていて、それ以外のアノテーションは直接文字列でキーを設定するしかないるようです。

blip.Annotation.BACKGROUND_COLOR（"style/backgroundColor"） テキストの背景色

blip.Annotation.COLOR（"style/color"） テキストの色

blip.Annotation.FONT_FAMILY（"style/fontFamily"） テキストのフォントファミリー（sans-serif、serif など）

blip.Annotation.FONT_SIZE（"style/fontSize"） テキストのフォントサイズ

blip.Annotation.FONT_STYLE（"style/fontStyle"） テキストのスタイル（italic、oblique など）

blip.Annotation.FONT_WEIGHT（"style/fontWeight"） テキストのフォントウェイト（太さ）

blip.Annotation.TEXT_DECORATION（"style/textDecoration"） テキストの修飾（underline、line-through など）

blip.Annotation.VERTICAL_ALIGN（"style/verticalAlign"） 垂直方向の文字揃え

"link/manual" ハイパーリンク

"link/wave" wave ID によるリンク

"lang" 入力された言語

[*33] style/で始まるもの

"conv/title" Wavelet のタイトル（通常は wavelet の最初のセンテンス）

blip.Annotation で定義されているアノテーションが設定されたテキストは、ブラウザ上ではいずれも span 要素と style 属性を使用して、**対象のテキスト**という形で表示されます。

BlipRefs

さて、Annotation は blip の特定の範囲に設定するものと書きました。ではその blip の範囲をどうやって指定するかというと、先の例のように blip.BlipRefs というオブジェクトを使用します。ただし blip.BlipRefs オブジェクトを直接生成することはほとんどなく、多くの場合は blip.Blip オブジェクトが持つ次のようなメソッドを使用して blip.BlipRefs オブジェクトを取得します。

all(findwhat=None, maxres=-1, **restrictions) 指定された条件に合う全て要素を含む BlipRefs を返します。例えば blip.all(element.Gadget, -1, url=GADGET_URL) とすれば、blip に含まれる GADGET_URL という URL を持つガジェットを全て抜き出すことができます

first(findwhat=None, **restrictions) 指定された条件に合う最初の要素を含む BlipRefs を返します。使用方法については all メソッドとほぼ同じです

at(index) 指定された位置から次の文字までを含む BlipRefs を返します

range(start, end) 指定された範囲を含む BlipRefs を返します

なお今回の例では取得した BlipRefs を利用してその範囲のコンテンツにアノテーションを設定しましたが、BlipRefs を使用するとそれ以外にもその範囲のコンテンツに要素を追加・削除・置換することができます。例えば次のように annotate メソッドの代わりに replace メソッドを使用すると、分からない言葉の背景色を変更するのではなく、伏字に置換できます。

```
1  #blip_ref.annotate(blip.Annotation.BACKGROUND_COLOR, 'yellow')
2  blip_ref.replace(u'？？？')
```

BlipRefs の持つ replace 以外のメソッドについては 199 ページを参照してください。

なお、blip には text というプロパティもありますが、このプロパティは読み取り専用です。text プロパティの値を文字列操作しても blip に反映させることはできません。今回説明したように BlipRefs オブジェクトを経由してコンテンツを操作すると言うのは少し面倒に感じるかもしれませんが、これは wave のドキュメントの構造に起因します。詳細については第 6.2 節を参照してください。

この時点での会話の様子は次のようになります。ロボットが尋ねている「come」という単語の背景色が変更されていることがわかるでしょう。

4.3.5.6 アノテーションを読み取る

??

先ほどはロボットの側から blip にアノテーションを設定しました。今度は反対にユーザーが blip に設定したアノテーションを読み取ってロボットに処理させてみましょう。ユーザーがロボットに覚えさせたい単語の背景色を赤色に設定したら、ロボットはその単語の意味を質問します。

アノテーションが変更されたときに発生するイベントは `events.AnnotatedTextChanged` です。まずはこのイベントとイベントハンドラをロボットに登録します。

```
1  robot.register_handler(events.AnnotatedTextChanged, AskWords)
```

イベントハンドラ関数の名前は `AskWords` です。早速その実装を見てみましょう。

```
1  def AskWords(events, wavelet):
2    # ログメッセージを出力
3    logging.info(u'AskWords 実行')
4
5    # 背景色を赤に設定するアノテーションかどうか
6    def target_annotation(annotation):
7      return annotation.name == blip.Annotation.BACKGROUND_COLOR \
8        and annotation.value == 'rgb(229, 51, 51)'
9
10   # 背景色を赤に設定されたことによるイベントなら処理
11   if target_annotation(events):
12     modified_blip = events.blip
13     # 変更のあった blip のアノテーションをすべてチェック
14     for annot in modified_blip.annotations:
15       # 背景色を赤に設定するアノテーションかどうか
16       if target_annotation(annot):
17         # 背景色が赤の文字を取得して質問
18         colored_text = modified_blip.text[annot.start:annot.end]
19         reply_blip = modified_blip.reply()
20         reply_blip.append(u'「%s」ってなんですか？' % colored_text)
```

アノテーションが背景色を赤色に設定するためのものかどうかというチェックはハンドラ内で複数回実行されるので関数（target_annotation）にしておきます[*34]。Google Wave のツールバーで背景色を赤色に設定した時のアノテーションの値として rgb(229, 51, 51) を使っていますが、これは実際に動作させた結果をログに出力して確認しました。

events.AnnotatedTextChanged イベントが発生するのは背景色が変更されたときだけではありません。リンクが貼られたときやフォントが変更されたときにも発生します。そのため、まず最初にどのようなアノテーションの変化があったのかを確認して、処理する必要があるかどうかを判断しています。具体的には、表 4.4 に書いているとおり、events.AnnotatedTextChanged イベント変化のあったアノテーションの名前（name）と値（value）を持っているので、それらを確認して背景色が赤色に変更されて発生したイベントだった場合にだけ処理を実行します。

ユーザーに質問を投げかける部分はこれまでのサンプルと同様です。アノテーションが設定されている部分文字列を抜き出している modified_blip.text[annotation.start:annotation
の部分はちょっとしたイディオムとして覚えておいてもいいかもしれません。

4.3.5.7　フォームを利用する

ここまででロボットはユーザーにいろいろと質問ができるようになりましたが、その質問は投げっぱなしで回答を受け取る手段がありません。実はロボットはテキストだけではなく、テキストフィールドやボタンなど、次のリストにあるいろいろなものを blip に追加することができます。本項ではそれらを利用してロボットがユーザーからの回答を受け取れるようにしてみましょう。

element.Button(name, caption)　ボタン
element.Check(name, value='')　チェックボックス
element.Input(name, value='')　テキストフィールド
element.Label(label_for, caption)　ラベル
element.Password(name, value)　パスワードフィールド
element.RadioButton(name, group)　ラジオボタン
element.RadioButtonGroup(name, value)　ラジオボタングループ

[*34] Python は関数定義の中で関数を定義できます。内側の関数のスコープは外側の関数定義の中だけです

element.TextArea(name, value)　テキストエリア
element.Gadget(url, props=None)　ガジェット
element.Image(url='', width=None, height=None, attachmentId=None, caption=None)
　　画像
element.Attachment(caption=None, data=None, mimeType=None, attachmentId=None, data=data, mimeType=mimeType, atta
　　添付ファイル
element.Installer(manifest)　インストーラー
element.Line(line_type=None, indent=None, alignment=None, direction=None)　一行のテキスト

ここではロボットが理解できない単語があったときに、その答えを入力してもらうためのテキストフィールドとボタンを blip に追加し、さらにボタンがクリックされるとテキストフィールドに入力した値をロボットに送られるようにしてみましょう。[*35]

単語の意味を尋ねる部分は英語を入力された時と背景色を赤くしたときの 2 ヶ所あるので、それらを一つの関数にまとめておき、その中でフォーム要素を追加します。

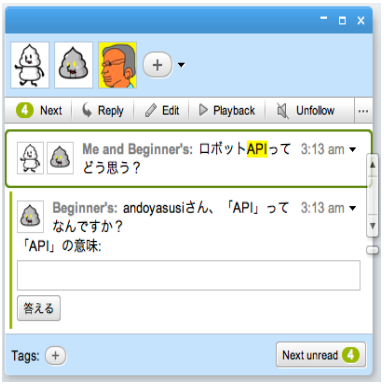
```
1  def AskMeaning(reply_blip, word, message):
2    reply_blip.append(message)
3    reply_blip.append(element.Line())
4    reply_blip.append(element.Label(
5      'meaning', u'「%s」の意味:' % word))
6    reply_blip.append(element.TextArea('meaning', ''))
7    reply_blip.append(element.Button('answer', u'答える'))
```

クラス名から読み取れるとは思いますが、メッセージ、改行、ラベル、テキストエリア、ボタンを順に blip に追加しています。さらに、これまで質問メッセージを表示していた部分をこの関数を使うように修正しておきましょう。

```
1  # 英単語があれば聞き返す。なければ分かったふりをする
2  def MaybeUnderstand(event, wavelet):
3    #.. 省略..
4
5    #reply_blip.append(u'%s さん、「%s」ってなんですか？'
6    #  % (GetName(modified_blip.creator), en_match.group()))
7    known_word = en_match.group()
8    AskMeaning(reply_blip, known_word, u'%s さん、「%s」ってなんですか？'
9      % (GetName(modified_blip.creator), known_word))
1  def AskWords(events, wavelet):
2    # .. 省略..
3
4    #reply_blip.append(u'「%s」ってなんですか？' % colored_text)
5    AskMeaning(reply_blip, colored_text, u'「%s」ってなんですか？' % \
6      colored_text)
```

[*35] 英単語の意味を通常の Reply で受け取ることもできますし、むしろその方が wave ロボットらしいようにも思えますが、今回はフォーム要素を使う練習の意味で敢えてこのようにしました

この時点でロボットの質問 blip の表示は次のようになっています。



ただし、まだボタンやテキストフィールドは表示されているだけです。テキストフィールドに回答を入力してボタンを押してもなにもおきません。次に、ロボットが「答える」ボタンに反応できるようにします。ロボットによって追加されたボタンが押されたときに発生するイベントは events.FormButtonClicked です。このイベントが発生したらロボットが参加者がテキストエリアに入力した値を読み取るようにしましょう。回答を読み取るためのハンドラ、ReadAnswer と events.FormButtonClicked をロボットに登録します。

```
1  robot.register_handler(events.FormButtonClicked, ReadAnswer)
```

ReadAnswer イベントハンドラの実装は次のようになります。

```
1  def ReadAnswer(events, wavelet):
2    # 押されたボタンの名前が answer なら処理
3    if events.button_name == 'answer':
4      modified_blip = events.blip
5
6      # ラベルから理解できない英単語を取得
7      label_ref = modified_blip.first(element.Label, name='meaning')
8      label = label_ref.value()
9      word = label.value[len(u'「'):-len(u'」の意味:')]
10
11     # テキストエリアから単語の意味を取得
12     text_area_ref = modified_blip.first(element.TextArea,
13       name='meaning')
14     text_area = text_area_ref.value()
15     meaning = text_area.value
16
17     # ボタンを取得（消去するため）
18     button_ref = modified_blip.first(element.Button, name='answer')
19
20     # 答えを聞いたら要素を全て消去
21     label_ref.delete()
```

```
22      text_area_ref.delete()
23      button_ref.delete()
24
25      # 返答
26      reply_blip = modified_blip.reply()
27      reply_blip.append(u'「%s」の意味は「%s」ですか。なるほど' %
28        (word, meaning))
```

少し長いような気がするかもしれませんが、ラベルとテキストエリアとボタンについてほとんど同じ処理を繰り返しているだけです。ここでは Blip#first メソッドの使い方と、blip オブジェクトからフォーム要素を取得して値を読み取る手順を確認してください。Blip#first メソッドでテキスト以外の要素を検索する場合は第一引数に要素のクラス、第二引数以降には検索条件を渡します[*36]。検索条件は要素のクラスに特有のプロパティとその値で、今回の様にフォーム要素を検索する場合には name と value が使用できます。また、Blip#first メソッドの返す値は要素そのものではなく BlipRefs オブジェクトなので、実際に要素オブジェクトを取得するにはさらに BlipRefs#value メソッドを呼ばなければいけません。最後に、参加者が間違えて同じ質問に 2 度答えてしまわないよう、答えを聞いたらフォーム要素を全て削除しています。

4.3.5.8 データストアを利用する

[*37]

これで分からない単語の意味を参加者に教えてもらい、さらにそれをロボットが受け取ることができるようになりました。しかしこれだけではまだ足りません。今は意味を教えてもらってもロボットは反射的に「分かった」と答えているだけで実際には何も分かっていません。同じ単語が出てくるとまた同じようにその意味を質問してしまうでしょう。これでは教えてくれた人に対して失礼なので、教わった単語の意味はきちんと GAE のデータストアに保存しておきましょう。

データストア

データストアは GAE から利用できるデータストレージで、皆さんの使い慣れている関連データベースとは異なる、次のような特徴を持っています。

スキーマレス 事前にスキーマを定義しておく必要はありません。同種のデータが異なるプロパティを持つことすら可能です。例えばアプリケーションのバージョンが上がり、以前はなかったプロパティが後で追加されたとしても、それまでに登録したデータも含めて正常に動作します

NoSQL データストアはいわゆるキーバリューストア（KVS）の一種で、従来広く使われてきた関連データベースとは異なります。キーバーリューストアは RDB が得意とする表結合のような複雑な処理があまり得意ではない代わりに、単純な検索であれば非常に高速に動作します

[*36] テキストを検索する場合には単純に検索対象の文字列を引数に取ります
[*37] 本項はロボット API に関する説明ではありません

分散型アーキテクチャ　データは分散して管理されます。アプリケーション的に関連のあるデータが物理的・トポロジー的、どちらの意味においても近くに配置される保証というはありません。近くに配置したいデータ群がある場合はそのことをプログラムからデータストアに明示的に伝える必要があります

トランザクション　トランザクションを使用して複数の操作をアトミックに実行できます

GQL　SQL に似た問い合わせ言語を利用できます

おそらくほとんどの wave ロボットは GAE を利用することになるでしょう。その場合、データを永続化する必要があるときにはデータストアを利用することになります。データストアの詳細に関しては公式のドキュメントまたは他の書籍に譲りますが、今回のサンプルで使用する機能については、ここで一通り説明しておきます。

データ定義　データストアのデータ定義は db.Model を継承したクラスを作成し、db モジュールに定義されている各種プロパティクラス型を持つクラス変数を定義するだけです。これについては今回のロボットが使用するモデルクラスの定義を見てみるのが一番わかり易いでしょう。

```
1  # 単語帳クラス
2  class Wordbook(db.Model):
3    word = db.StringProperty(required=True)
4    meaning = db.StringProperty(required=True)
5    learned_at = db.DateProperty(auto_now_add=True)
```

単語帳クラスは単語と単語の意味と登録日という 3 つのプロパティを持っていて、単語と意味は必須入力項目、登録日は自動的に現在時刻が設定されます。関係データベースとは違い、データストアを利用するために必要なのはモデルクラスを定義することだけです。モデルクラスさえあれば、別にテーブルを定義する必要はありません。なお、プロパティクラスはこの例で使用した db.StringProperty と db.DateProperty 以外にも数多くのタイプが利用できます。詳細については公式ドキュメント[*38]を参照してください。

データの CRUD　CRUD のうち、データの作成・更新・削除についてはサンプルコードを見れば十分でしょう。先ほどの Wordbook モデルを使用した次のコードを見てください。

```
1  # データ新規作成
2  wordbook = Wordbook(word='football', meaning=u'サッカー')
3  wordbook.put()
4
5  # データ更新
6  wordbook.meaning = u'アメフト'
7  wordbook.put()
8
9  # データ削除
10 wordbook.delete()
```

データの新規登録も更新も put メソッドが呼ばれるまではデータストアに反映されていないという点に注意が必要です。

[*38] http://code.google.com/intl/en/appengine/docs/python/datastore/typesandpropertyclasses.html

CRUD の残る一つ、データの検索についてはそれ以外の 3 つほど単純ではありません。GAE ライブラリには検索に利用できるオブジェクトが Query オブジェクトと GqlQuery オブジェクトの二種類ありますが、本書では GqlQuery しか使用しないため、前者の説明は省略します。GqlQuery クラスは GQL という GAE で利用できる SQL に似た問い合わせ言語を扱うためのクラスで、GqlQuery を使って先ほどの Wordbook を検索すると次のようになります。

```
1  query = GqlQuery('SELECT * FROM Wordbook WHERE word = :word ' + \
2    'ORDER BY learned_at DESC', word='football')
3  for wordbook in query:
4    print wordbook.meaning
```

GqlQuery コンストラクタの第一引数が GQL で、SQL を知っている人には説明不要でしょう。SELECT * FROM で検索対象のモデルを指定して、WHERE 句で検索の条件を、ORDER BY 句で結果のソート順序を指示しています。また、GQL 内の:word はパラメータバインディングで、第二引数以降のキーワード引数がバインドされます。従ってこのサンプルでは word プロパティが'football' である Wordbook を検索し、その結果を登録日順に並び替えて、単語の意味を表示しています。

GQL で取得した全てのオブジェクトが必要な場合には、GqlQuery オブジェクトはイテレート可能なので、サンプルのように for 文を使用して全ての結果を処理します。一部のデータだけが必要な場合には GqlQuery オブジェクトの fetch(limit, offset=0) メソッドを使用してください。最初の一件だけで構わない場合には get() メソッドが利用できます。

なお、GqlQuery オブジェクトはモデルクラスの gql メソッドでも取得できます。こちらの方がターゲットのモデルが明確なので本書では主にこちらを使用します。

```
1  wordbooks = Wordbook.gql(
2    'WHERE word = :word ORDER BY learned_at DESC',
3    word='football')
4  for wordbook in wordbooks:
5    print "単語の意味:" + wordbook.meaning
```

GAE の Datastore API にはここで説明したもの以外にも多くの機能があります。本格的なロボットを作成するときには事前に公式ドキュメント[*39]に目を通しておきましょう。

・・・

それではロボットの改良に戻ります。参加者に教えてもらった単語をデータストアに登録して、参加者に同じ単語を二回質問しないようにするという話でした。まずはコードの最初の部分で今回の機能拡張に使用するモジュールをインポートしておきます。

```
1  from google.appengine.ext import db
2  import random
```

データストアを利用するためには google.appengine.ext.db モジュールが必要です。今回の例では検索結果からランダムにデータを抽出する処理があるので、乱数を扱うためのモジュール、randam モジュールもインポートしています。

[*39] http://code.google.com/intl/en/appengine/docs/python/datastore/

データストアを利用するには、まず最初にモデルを定義しておかなければいけません。先のデータストアの例でも使いましたが、もう一度今回のモデルを確認しておきましょう。

```
1  # 単語帳クラス
2  class Wordbook(db.Model):
3    word = db.StringProperty(required=True)
4    meaning = db.StringProperty(required=True)
5    learned_at = db.DateProperty(auto_now_add=True)
```

上記をロボットのソースコードの好きな場所に追加して下さい。Wavebook モデルのプロパティは文字列型の英単語とその意味、日付型のデータ登録日の 3 つです。データ登録日は auto_now_add を True に設定しているので、現在日が自動的に設定されます。それ以外の 2 つのプロパティ、英単語と意味は生成時に明示的に値を設定する必要があります。

モデルを定義したら、次にそのモデルを利用してユーザーが入力した単語の意味をデータストアに保存します。ユーザーが「答える」ボタンをクリックしたときに実行される ReadAnswer メソッドの最後に次のコードを追加して下さい。

```
1  def ReadAnswer(events, wavelet):
2    #.. 省略..
3
4    # データストアに保存
5    wordbook = Wordbook(word=word, meaning=meaning)
6    wordbook.put()
```

データの保存は先ほどのサンプルと同じです。これでユーザーに教えてもらった単語の意味を後から参照できるようになりました。次はユーザーの発言に英単語が含まれていたときに、それらの英単語の意味がすでにデータストアに保存されていたら、データストアに保存されている意味をロボットの方から応えるようにします。ただし、もし知らない英単語が一つでもあったらこれまでと同じで、フォーム要素を使ってユーザーにその意味を尋ねます。

MaybeUnderstand 関数を次のように変更してください。大幅な変更になるので関数全文を載せていますが、ここで重要なのは gql メソッドを呼び出している部分とその結果をイテレートしている部分のたった数行です。

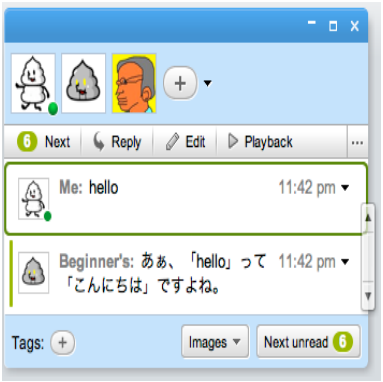
```
1  def MaybeUnderstand(event, wavelet):
2    # ログメッセージを出力
3    logging.info(u'MaybeUnderstand 実行')
4
5    # イベントに紐づいた blip を取得
6    modified_blip = event.blip
7    if len(modified_blip.text.strip()) == 0: return
8
9    # リプライ blip を作成
10   reply_blip = modified_blip.reply()
11
12   # 連続するアルファベットがあるかどうか
```

```
13    en_pattern = re.compile(r'[a-zA-Z]{2,}')
14    # 英単語をすべて抜き出し
15    en_words = en_pattern.findall(modified_blip.text)
16    if 0 == len(en_words):
17      # 英単語がなければ分かったふりをする
18      reply_blip.append(u'なるほど。わかります。')
19    else:
20      # 発言に含まれる英単語をデータストアに登録済みかどうかで分類
21      wordbooks = Wordbook.gql('WHERE word IN :words', words=en_words)
22      unknown_words = en_words
23      known_words = []
24      for wordbook in wordbooks:
25        unknown_words.remove(wordbook.word)
26        known_words.append(wordbook)
27
28      if len(unknown_words) == 0:
29        # 知らない英単語がなければ知ってる単語の意味を返す
30        known_word = random.choice(known_words)
31        reply_blip.append(u'あぁ、「%s」って「%s」ですよね。' %
32          (known_word.word, known_word.meaning))
33      else:
34        # 知らない英単語があれば質問する
35        unknown_word = random.choice(unknown_words)
36        AskMeaning(reply_blip, unknown_word, u'%sさん、「%s」ってなんで
すか？'
37          % (GetName(modified_blip.creator), unknown_word))
38
39        # 英単語の背景色を黄色くする
40        blip_ref = modified_blip.first(unknown_word)
41        blip_ref.annotate(blip.Annotation.BACKGROUND_COLOR, 'yellow')
```

まず初めに RegexObject#findall メソッドで発言内の全ての英単語を抜き出します。一つも英単語がなければ、簡単な返事をして終了です。一つでも英単語が含まれるたらそれらをデータストアから検索します。GQL の IN オペレーターは SQL と同じで、右辺にリストを取り左辺のプロパティの値がそのリストに含まれるときに真になります[*40]。従って wordbooks 変数にはユーザーの発言に含まれた英単語の中でデータストアに保存されている単語だけが含まれています。その結果を元に、発言に含まれる英単語をデータストアで見つかったもの（known_words）と見つからなかったもの（unknown_words）に分類し、結果に応じて次のようにユーザーに反応を返します。

[*40] IN オペレータは内部的には複数の=オペレーターを使った比較の OR 演算として処理されるため、あまり大きなリストを渡さないように注意してください

- 参加者の発言に英単語がない
  ->「なるほど、わかります」
- 参加者の発言に英単語がある
  - データストアにすべての単語が登録済み
    -> 英単語をひとつ選び、「あぁ、[英単語] って [意味] ですよね」
  - データストアに登録されていない単語がある
    -> 英単語をひとつ選び、「[参加者] さん、[英単語] ってなんですか？」



4.3.5.9　ガジェットを利用する

ここまででサンプルロボットはユーザーに英単語の意味を質問して、回答を読取り、それをデーターストアに保存することができます。この節ではガジェットと連携して保存した単語を削除できるようにします。もちろん主眼はデータの削除ではなくガジェットとの連携です。

ロボットは wave のほとんど全ての情報にアクセスできる代わりに、画面に表示できる内容が限られています。テキスト修飾は Annotation で可能な範囲に絞られていますし、表示できる要素もテキストと画像の以外には基本的なフォーム要素とガジェットだけです。一方、ガジェットはできることとできないことがちょうどロボットの反対になります。ガジェットのアクセスできる情報は非常に限られているかわりに、ガジェットの枠内であればどのようなものでも表示できます。 そういった相補的な関係があるので、ロボットとガジェットは組み合わせて利用すると非常に強力です。

なお、ロボットと組み合わせて使用する場合でもガジェット側には特別は処理は一切必要ありません。ガジェットはロボットに対して何もできませんが、ロボットはガジェットの共有状態を監視・変更することができるため、ロボットとガジェットは協働すると言うよりも、ロボットがガジェットを利用すると言う表現のほうが正確でしょう。

それではまず、サンプルロボットが利用するガジェットのソースを見てみます。

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <Module>
3    <ModulePrefs title="Wordbook" height="1">
4      <Require feature="wave" />
5      <Require feature="dynamic-height" />
6    </ModulePrefs>
```

```
 7   <Content type="html">
 8     <![CDATA[
 9     <div id="container"></div>
10     <script>
11       // 指定された単語を共有状態から削除
12       function deleteWord(word) {
13         if (window.confirm(word + 'を削除しますか？')) {
14           var state = wave.getState();
15           state.submitValue(word, null);
16         }
17       }
18
19       // 画面表示
20       function render() {
21         var html = '';
22         if (wave.getMode() != wave.Mode.EDIT) {
23           // 編集モード以外
24           html += '単語帳を編集するには編集モードに切り替えてください'
25         }
26         else {
27           // 編集モード
28           html += '削除したい単語をクリックしてください'
29           html += '<dl>';
30           var state = wave.getState();
31           var keys = state.getKeys();
32           for (var i = 0; i < keys.length; i++) {
33             var word = keys[i];
34             if (word == 'deleted_word') continue;
35             var meaning = state.get(word, '');
36             html += '<dt onclick="javascript:deleteWord(\'' +
37               word + '\')">' + word + '</dt>';
38             html += '<dd>' + meaning + '</dd>';
39           }
40           html += '</dl>';
41         }
42         document.getElementById('container').innerHTML = html;
43         gadgets.window.adjustHeight();
44       }
45
46       // 初期化
47       function init() {
48         if (wave && wave.isInWaveContainer()) {
```

```
49            wave.setStateCallback(render);
50            wave.setModeCallback(render);
51          }
52        }
53
54        // 初期化ハンドラ登録
55        gadgets.util.registerOnLoadHandler(init);
56      </script>
57      ]]>
58    </Content>
59  </Module>
```

このガジェットが使用する共有状態は英単語がキー、その意味が値という連想配列になっています。ガジェットは wave のモードによって表示内容が異なり、編集モードの時には登録されている単語の一覧が表示され単語をクリックするとその単語を削除できます。それ以外のモードでは「単語帳を編集するには編集モードに切り替えてください」というメッセージが表示されます。参考までに"hello"と"world"という単語が登録されているときに編集モードの DOM がどのようになっているかを見てみましょう。

```
1  <div id="container">
2    削除したい単語をクリックしてください
3    <dl>
4      <dt onclick="javascript:deleteWord('hello')">hello</dt>
5      <dd>こんにちは</dd>
6      <dt onclick="javascript:deleteWord('world')">world</dt>
7      <dd>世界</dd>
8    </dl>
9  </div>
```

render 関数で共有状態の内容、つまり登録されている単語を見てこの DOM を組み立てています。単語の削除は共有状態からその単語を取り除くだけです。共有状態からアイテムを削除するには State#submitValue 関数の第二引数に null を渡してください。<dt>要素をクリックしたときに実行される deleteWord 関数内で単語の削除が実行されています。

なお、先に述べたようにこのガジェットはロボットとは無関係なので、単体で wave に貼り付けることもできます。ただし初期状態である単語一覧の設定をロボットで行うように作ってあるため、貼り付けたとしてもそのままではなにもできません。

ではここからこのガジェットを管理するロボットの解説に移ります。まずはイベントハンドラを登録しましょう。ガジェットを管理するロボットが処理しなければいけないイベントは通常は 2 つです。一つは WaveletSelfAdded イベントで、このイベントに対応して自分が管理するガジェットを貼りつけます。そしてもうひとつ重要なイベントが GadgetStateChanged で、これはその名の通りガジェットの共有状態が変更されたときに発生します。今回のサンプルではこのイベントを受け取ってデータストアから単語を削除することになります。

```
1  robot.register_handler(events.WaveletSelfAdded, AddGadget)
```

```
2  robot.register_handler(events.GadgetStateChanged, RemoveWord)
```

なお、WaveletSelfAdded にはすでに他の関数（SayHello）が登録されていますが、問題はありません。一つのイベントに対していくつハンドラを登録したとしても、イベントが発生すれば全てのハンドラが登録した順序で実行されます[*41]。ではまず、そのWaveletSelfAdded を処理する AddGadget 関数の定義を見てみましょう。

```
1  # ガジェットの絶対 URL
2  GADGET_URL = 'http://[アプリケーション ID].appspot.com/assets/wordbook.xml'
3
4  # ガジェットを追加する
5  def AddGadget(event, wavelet):
6    logging.info('AddGadget')
7    # ガジェットの初期値
8    props = {}
9    for wordbook in Wordbook.all():
10     props[wordbook.word] = wordbook.meaning
11
12   # ガジェットをルート blip に追加
13   gadget = element.Gadget(url=GADGET_URL, props=props)
14   wavelet.root_blip.append(gadget)
```

ガジェットを wave に追加するには Blip#append で element.Gadget オブジェクトを追加するだけです。ガジェットの共有状態に初期値が必要な場合はコンストラクタの props 引数で指定します。今回は Wordbook の全データを共有状態の初期値に設定しています。なお、ガジェットはロボットサーバーからアクセスされるわけではありません。ガジェット XML をロボットと同じ GAE スロットでホストしていたとしても、ガジェット URL は絶対 URL で指定してください。

次にガジェットの共有状態が変更されたときに呼び出される RemoveWord 関数をみてみます。

```
1  # 指定された単語をデータストアから削除する
2  def RemoveWord(event, wavelet):
3    logging.info('RemoveWord')
4    # 削除された単語を取得
5    deleted_word = event.old_state.keys()[0]
6    if deleted_word:
7      wordbook = Wordbook.gql('WHERE word = :word',
8        word=deleted_word).get()
9      # データストアから単語を削除
10     wordbook.delete()
```

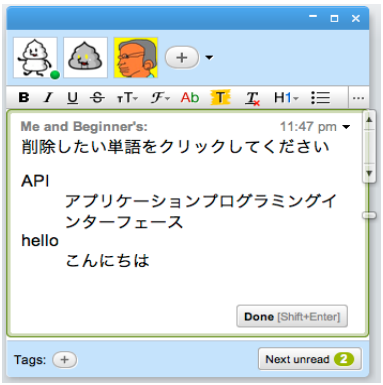
GadgetStateChanged イベントオブジェクトは old_state プロパティに変更のあった共有状態を保持しています。つまり今回のサンプルでは old_state を見れば削除された単語が分かりますので、その単語をデータストアで検索して削除しています。

[*41] ガジェットのイベントハンドラが最後に登録されたもの以外は無効になるのと対照的です

これでガジェットから単語を削除できるようになりました。最後に単語が新しく登録されたときにガジェットにもその単語が追加されるようにしておきましょう。

```
1  def ReadAnswer(events, wavelet):
2    # .. 略 ..
3
4    # ガジェットを更新
5    gadget_ref = wavelet.root_blip.first(element.Gadget,
6      url=GADGET_URL)
7    gadget_ref.update_element({word:meaning})
```

イベントハンドラ内でガジェットのインスタンスを取得するには Blip の検索メソッドでタイプを element.Gadget、検索条件としてガジェット URL を渡して検索します。ガジェットの共有状態は BlipRefs#update_element メソッドに辞書オブジェクトを渡して更新できます。なお、ロボットは自分が起点になったイベントを自分自身で受け取ることはありません。つまり、ここで共有状態を変更したことによって GadgetStateChanged イベントが発生しますが、そのイベントは wave サーバーによって無視され、このロボットには届きません[*42]。



4.3.5.10　添付ファイルを読み取る

今のところロボットに英単語を覚えさせるには、英単語を入力するか単語の背景色を設定してロボットの質問を誘うしかありませんが、これだけでは覚えさせたい単語が多いと非常に手間がかかってしまいます。そこで、辞書ファイルを wave に添付するとファイルに記述されている単語を全て覚えるようにしてみましょう。

添付ファイルはロボット内では elements.Attachment クラスのインスタンスとして表されます。ファイルの添付自体は何もイベントを発しません。blip の編集を終了したとき（BlipSubmitted）にその blip に辞書ファイルが添付されていたら、そのファイルを読み取って処理することにしましょう。辞書ファイルのキャプションは"dictionary"で、一エントリに付き一行、英単語とその意味は":"で区切られるものとします。

```
1  OPI:数あるネイルブランドの中でもトップに君臨する王道ブランド
2  charisma:人を魅了する非日常的な能力、またその非凡な資質を持つ人間
```

[*42] イベントが届かないのは自分自身が原因になったイベントだけです。wave に別のロボットが登録されていて、そのロボットも GadgetStateChanged イベントハンドラを持っていた場合、そちらにはイベントが届きます

```
3  Bath:イギリス南西部、ブリストル市の南東に位置するエイボン川沿いの都市
```

それではソースを見てみます。特に難しいところはないでしょう。

```
1  robot.register_handler(events.BlipSubmitted, ReadAttachment)
1  def ReadAttachment(event, wavelet):
2    logging.info(u'ReadAttachment 実行')
3
4    # 添付ファイルを取得
5    blip = event.blip
6    attachment_ref = blip.first(element.Attachment,
7      caption='dictionary')
8    if attachment_ref:
9      attachment = attachment_ref.value()
10
11     # 添付ファイルの内容を読み取って保存
12     dictionary = urllib2.urlopen(attachment.attachmentUrl
13       ).read().strip()
14     for line in dictionary.splitlines():
15       word, meaning = map(lambda s:unicode(s, 'utf-8'),
16         line.split(':', 2))
17       StoreWordbook(wavelet, word, meaning)
18
19     # 添付を解除
20     attachment_ref.replace(u' 覚えました')
21
22 # データストアとガジェットを更新
23 def StoreWordbook(wavelet, word, meaning):
24   # データストアに保存
25   wordbook = Wordbook(word=word, meaning=meaning)
26   wordbook.put()
27
28   # ガジェットを更新
29   gadget_ref = wavelet.root_blip.first(element.Gadget,
30     url=GADGET_URL)
31   gadget_ref.update_element({word:meaning})
```

単語をデータストアに登録してガジェットを更新する処理はフォームを通じてユーザーに単語を教えてもらうとき（ReadAnswer 関数内）と同じ処理になるので、関数にまとめました。elements.Attachment オブジェクトが保持しているのは添付ファイルの URL だけなので、ファイルの内容が必要なときにはサンプルのように urllib2 を利用します。

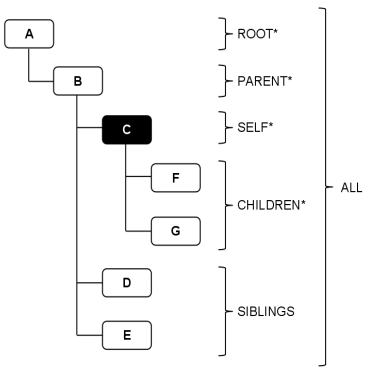
4.3.5.11　wave サーバーから受け取る情報を指定する

機能的には一通り完成しましたが、このロボットが多くの人に使われるようになったときのことを考えると少し不安があります。GAE には CPU 利用率や転送量などのリソースごとに無料で使用できるクォータが決められていて、その割当を超えるとしばらくアプリケーションが動作しなくなります。そのような状態になってもロボットはウェブページとは違いエラーページが表示されるわけではありません。ユーザーの混乱を避けるためにはなるべくクォータを超えない工夫をすることが必要でしょう。この項と次の項ではサーバーへの負荷を少しでも下げる方法について説明します。

これまでに見てきたようにロボットが受け取るイベントには関連する blip や wavelet など様々な情報がついてきています。これらの情報は実際にはロボットサーバーが wave サーバーから受け取っているものです。GAE には受信帯域幅の制限もありますから、アプリケーションで使わない情報は wave サーバーから送られないようにして、必要な情報だけを受け取った方がよいことは明らかでしょう。これを可能にするのが本項で説明する「コンテキスト」です。

これまで `Robot#register_handler` メソッドにはイベントとハンドラ、二つの引数しか指定していませんでした。実はこのメソッドはもう 2 つ引数をとることができ、その内の一つにコンテキストがあります。コンテキストを指定すると wave サーバーからロボットサーバーへ送られる Blip の種類を指定できます。例えばイベントハンドラ登録時に `[events.Context.SELF]` というコンテキストを与えると、wave サーバーはイベントの発生元となった Blip の情報だけをロボットサーバーへ送ります。

設定できるコンテキストの一覧は次です。



events.Context.ROOT ルート blip だけを受け取ります
events.Context.PARENT イベントが発生した blip の親 blip だけを受け取ります。この指定は Wavelet イベントには何の影響も与えません
events.Context.SIBLINGS イベントが発生した blip の兄弟 blip を受け取ります。Blip の場合、同じ Wavelet に属する全ての兄弟 blip を受け取ります
events.Context.CHILDREN イベントが発生した blip の子 blip を受け取ります。Wavelet の場合、含まれる全ての blip を受け取ります
events.Context.SELF イベントが発生した blip 自身だけを受けります
events.Context.ALL イベントが発生した wavelet の全ての blip を受け取ります

デフォルトでは全てのイベントは、PARENT、CHILDREN、ROOT、SELF を受け取ります。これは多くのイベントハンドラにとっての最小公倍数のようなものです。イベントハンドラが何を必要としているかがはっきりしているのなら明示的にコンテキストを指定した方がいいでしょう。単語帳ロボットの register_handler メソッド呼び出しを、無駄のないコンテキストを指定するように修正すると次のようになります。

```
1 robot.register_handler(events.WaveletSelfAdded, SayHello,
2   context=[events.Context.SELF])
3 robot.register_handler(events.WaveletSelfAdded, AddGadget,
4   context=[events.Context.SELF, events.Context.ROOT])
5 # .. 省略..
```

なお、コンテキストは wave サーバーから送られる情報を制限するだけではなく、広げるためにも使用します。例えば全 blip の情報が必要なイベントハンドラは、デフォルトでは PARENT、CHILDREN、ROOT、SELF しか受け取りませんから、register_handler 関数で context=[events.Context.ALL] という指定をする必要があります。

4.3.5.12 ハンドラの実行をフィルタリングする

ロボットが処理できるイベントはたくさんありますが（表 4.4）、中でも DocumentChanged は注意が必要です。DocumentChanged イベントのハンドラをロボットに登録すると、何もしなければクライアント上でユーザーが一文字入力するごとにそのハンドラが呼び出されることになります。ユーザーの入力をリアルタイムに処理するにはこのイベントを使用するしかないとは言え、全入力に対して処理する必要があることはまれでしょう。GAE にはリクエスト回数の制限もありますから、不要なイベント通知は受け取らないようにしましょう。

ロボット API は DocumentChanged のように発生頻度の高いイベントについて、イベントの通知をフィルタリングする仕組みが用意されています。たとえば、wiki のように、[[text]] というパターンを自動リンクするロボットを作成しているのであれば、register_handler で次のように指定します。

```
1 robot.register_handler(events.DocumentChanged, onNewMatch,
2   filter="\[\[.*\]\]")
```

このように指定するとユーザーの入力が filter に設定した正規表現にマッチしたときにだけ、イベントハンドラが呼び出されます。

現在のところフィルタが使用できるイベントは次の 4 つです。

DOCUMENT_CHANGED ユーザーの入力でフィルタリングされます
ANNOTATED_TEXT_CHANGED アノテーションのキー名でフィルタリングされます
GADGET_STATE_CHANGED 共有状態のキー名でフィルタリングされます
FORM_BUTTON_CLICKED ボタン名でフィルタリングされます

せっかくなので今回のサンプルでも使用してみましょう。

```
1 robot.register_handler(events.AnnotatedTextChanged, AskWords,
2   context=[events.Context.SELF],
3   filter=blip.Annotation.BACKGROUND_COLOR)
```

これまでは背景色以外にも文字サイズや文字色など全てのアノテーションが変更されたときにハンドラが呼び出されていましたが、このように指定すると背景色が変更されたときにだけイベントハンドラが実行されるようになります。

4.3.5.13 エラーをハンドリングする

ロボット内で wavelet に blip を追加したり内容を変更しても、その処理は wave サーバーの持つキューに登録されるだけで、実際に実行されるのはほとんどの場合ロボットサーバーでの処理が終了してからです。そのため、ロボットサーバーでの処理は正常に終了したが wave サーバーでエラーが発生した、という事態が起こり得ます。そう言ったエラーをロボットが感知するには events.OperationError イベントのハンドラをロボットに登録します。といっても実は appengine_robot_runner はデフォルトでロボットにこのイベントのハンドラを登録しているので、そのデフォルの処理で構わないのであれば開発者はなにもすることはありません。appengine_robot_runner.py から関係する部分を抜き出してみましょう。

```
1 def operation_error_handler(event, wavelet):
2   if isinstance(event, events.OperationError):
3     logging.error('Previously operation failed: id=%s, message:
%s',
4                   event.operation_id, event.error_message)
5
6 def run(robot, debug=False, log_errors=True, extra_handlers=None):
7   if log_errors:
8     robot.register_handler(events.OperationError,
9       operation_error_handler)
```

デフォルトでは、wave サーバーでエラーが発生するとその内容をエラーログに書き出します。ちょっとしたロボットであればこれで十分が、非常にクリティカルなロボットを運用しているときには、エラーの発生をメールですぐに通知して欲しいこともあるでしょう。そのようなときには自作のハンドラで events.OperationError を処理するようにします。

```
1 inbound_services:
2 - mail
```

GAE でメールを送信するにはまず inbound_services の設定が必要です。app.yaml の最後に上の設定を追加してください。

```
from google.appengine.api import mail

def SendError(events, wavelet):
  if isinstance(event, events.OperationError):
    mail.send_main(
      sender="[ロボット名] <[アプリケーション ID]@appspot.com>",
      to="your@mail.address",
      subject="Robot Error"
      body=("Previously operation failed: id=%s, message: %s" %
        (event.operation_id, event.error_message))

robot.register_handler(events.OperationError, SendError)
```

メールの送信は mail.send_main 関数を使います。引数の詳細についてはサンプル内のコードを参照してください。

4.3.5.14 全ソースコード

以上でイベント駆動型のロボットサンプルが完成しました。以下にロボットの全ソースコードを載せておきます。

```
# -*- coding: utf-8 -*-

# 利用するモジュールを読込
from waveapi import element
from waveapi import events
from waveapi import robot
from waveapi import appengine_robot_runner
import logging

from waveapi import blip
import re

from google.appengine.ext import db
import random

import urllib2

# 単語帳クラス
class Wordbook(db.Model):
  word = db.StringProperty(required=True)          # 単語
  meaning = db.StringProperty(required=True)       # 単語の意味
  learned_at = db.DateProperty(auto_now_add=True) # 登録日

# 自分が追加されたときに挨拶
```

```
25 def SayHello(event, wavelet):
26   # 新しく blip を追加して挨拶を表示
27   wavelet.reply(u'こんにちは。Wave に呼んでくれてありがとう')
28
29 # ガジェットの絶対 URL
30 GADGET_URL = 'http://wave-book.appspot.com/assets/wordbook.xml'
31
32 # ガジェットを追加する
33 def AddGadget(event, wavelet):
34   # ガジェットの初期値
35   props = {}
36   wordbooks = Wordbook.all().fetch(10)
37   for wordbook in wordbooks:
38     props[wordbook.word] = wordbook.meaning
39
40   # ガジェットをルート blip に追加
41   gadget = element.Gadget(url=GADGET_URL, props=props)
42   wavelet.root_blip.append(gadget)
43
44 # 指定された単語をデータストアから削除する
45 def RemoveWord(event, wavelet):
46   # 削除された単語を取得
47   deleted_word = event.old_state.keys()[0]
48   if deleted_word:
49     wordbook = Wordbook.gql('WHERE word = :word',
50       word=deleted_word).get()
51     # データストアから単語を削除
52     wordbook.delete()
53
54 # 誰かが追加されたときに挨拶
55 def SayHelloToParticipants(event, wavelet):
56   # 新しく来た参加者に挨拶
57   for participant in event.participants_added:
58     if not IsMyself(participant):
59       wavelet.reply(u'ようこそ、%s さん' % GetName(participant))
60
61 # 英単語があれば聞き返す。なければ分かったふりをする
62 def MaybeUnderstand(event, wavelet):
63   # イベントに紐づいた blip を取得
64   modified_blip = event.blip
65   if len(modified_blip.text.strip()) == 0: return
66
```

```
67    # リプライ blip を作成
68    reply_blip = modified_blip.reply()
69
70    # 連続するアルファベットがあるかどうか
71    en_pattern = re.compile(r'[a-zA-Z]{2,}')
72    # 英単語をすべて抜き出し
73    en_words = en_pattern.findall(modified_blip.text)
74    if 0 == len(en_words):
75      # 英単語がなければ分かったふりをする
76      reply_blip.append(u'なるほど。わかります。')
77    else:
78      # 発言に含まれる英単語をデータストアに登録済みかどうかで分類
79      wordbooks = Wordbook.gql('WHERE word IN :words', words=en_words)
80      unknown_words = en_words
81      known_words = []
82      for wordbook in wordbooks:
83        unknown_words.remove(wordbook.word)
84        known_words.append(wordbook)
85
86      if len(unknown_words) == 0:
87        # 知らない英単語がなければ知ってる単語の意味を返す
88        known_word = random.choice(known_words)
89        reply_blip.append(u'あぁ、「%s」って「%s」ですよね。' %
90          (known_word.word, known_word.meaning))
91      else:
92        # 知らない英単語があれば質問する
93        unknown_word = random.choice(unknown_words)
94        AskMeaning(reply_blip, unknown_word, u'%sさん、「%s」ってなんで
すか？'
95          % (GetName(modified_blip.creator), unknown_word))
96
97        # 英単語の背景色を黄色くする
98        blip_ref = modified_blip.first(unknown_word)
99        blip_ref.annotate(blip.Annotation.BACKGROUND_COLOR, 'yellow')
100
101 # 単語の意味を尋ねる
102 def AskWords(events, wavelet):
103   # 背景色を赤に設定するアノテーションかどうか
104   def target_annotation(annotation):
105     return annotation.name == blip.Annotation.BACKGROUND_COLOR \
106       and annotation.value == 'rgb(229, 51, 51)'
107
```

```
108   # 背景色を赤に設定されたことによるイベントなら処理
109   if target_annotation(events):
110     modified_blip = events.blip
111     # 変更のあった blip のアノテーションをすべてチェック
112     for annon in modified_blip.annotations:
113       # 背景色を赤に設定するアノテーションかどうか
114       if target_annotation(annon):
115         # 背景色が赤の文字を取得して質問
116         colored_text = modified_blip.text[annon.start:annon.end]
117         reply_blip = modified_blip.reply()
118         AskMeaning(reply_blip, colored_text, u'「%s」ってなんですか？'
 %
119           colored_text)
120
121 # 単語の意味を尋ねるフォームを表示
122 def AskMeaning(reply_blip, word, message):
123   reply_blip.append(message)
124   reply_blip.append(element.Line())
125   reply_blip.append(element.Label('meaning', u'「%s」の　意　味:' %
 word))
126   reply_blip.append(element.TextArea('meaning', ''))
127   reply_blip.append(element.Button('answer', u'答える'))
128
129 # 回答を読み取る
130 def ReadAnswer(events, wavelet):
131   # 押されたボタンの名前が answer なら処理
132   if events.button_name == 'answer':
133     modified_blip = events.blip
134
135     # ラベルから理解できない英単語を取得
136     label_ref = modified_blip.first(element.Label, name='meaning')
137     label = label_ref.value()
138     word = label.value[len(u'「'):-len(u'」の意味:')]
139
140     # テキストエリアから単語の意味を取得
141     text_area_ref = modified_blip.first(element.TextArea,
142       name='meaning')
143     text_area = text_area_ref.value()
144     meaning = text_area.value
145
146     # ボタン取得（消去するため）
147     button_ref = modified_blip.first(element.Button, name='answer')
```

```
148
149       # 答えを聞いたら要素を全て消去
150       label_ref.delete()
151       text_area_ref.delete()
152       button_ref.delete()
153
154       StoreWordbook(wavelet, word, meaning)
155
156       # 返答
157       reply_blip = modified_blip.reply()
158       reply_blip.append(u'「%s」の意味は「%s」ですか。なるほど' %
159         (word, meaning))
160
161   # 添付ファイルを読み取る
162   def ReadAttachment(event, wavelet):
163     # 添付ファイルを取得
164     blip = event.blip
165     attachment_ref = blip.first(element.Attachment,
166       caption='dictionary')
167     if attachment_ref:
168       attachment = attachment_ref.value()
169
170       # 添付ファイルの内容を読み取って保存
171       dictionary = urllib2.urlopen(attachment.attachmentUrl
172         ).read().strip()
173       for line in dictionary.splitlines():
174         word, meaning = map(lambda s:unicode(s, 'utf-8'),
175           line.split(':', 2))
176         StoreWordbook(wavelet, word, meaning)
177
178       # 添付を解除
179       attachment_ref.replace(u'覚えました')
180
181   # データストアとガジェットを更新
182   def StoreWordbook(wavelet, word, meaning):
183     # データストアに保存
184     wordbook = Wordbook(word=word, meaning=meaning)
185     wordbook.put()
186
187     # ガジェットを更新
188     gadget_ref = wavelet.root_blip.first(element.Gadget,
189       url=GADGET_URL)
```

```
190   gadget_ref.update_element({word:meaning})
191
192 # 自分自身かどうかをチェック
193 def IsMyself(participant_id):
194   return participant_id == 'wave-book@appspot.com'
195
196 # 参加者 ID からドメインを除いたものを取得
197 def GetName(participant_id):
198   return participant_id.split('@')[0]
199
200 # ライブラリとしてではなく、直接に実行されたとき処理を実行
201 if __name__ == '__main__':
202   # プロフィール情報を指定してロボット定義
203   robot = robot.Robot('Beginner\'s Guide to Wave',
204     image_url='http://wave-book.appspot.com/assets/icon.png',
205     profile_url='http://wave-book.appspot.com/')
206
207   # ロボットにイベントハンドラを登録
208   robot.register_handler(events.WaveletSelfAdded, SayHello,
209     context=[events.Context.SELF])
210   robot.register_handler(events.WaveletSelfAdded, AddGadget,
211     context=[events.Context.SELF, events.Context.ROOT])
212   robot.register_handler(events.GadgetStateChanged, RemoveWord,
213     context=[events.Context.SELF])
214   robot.register_handler(events.WaveletParticipantsChanged,
215     SayHelloToParticipants, context=[events.Context.SELF])
216   robot.register_handler(events.BlipSubmitted, MaybeUnderstand,
217     context=[events.Context.SELF])
218   robot.register_handler(events.BlipSubmitted, ReadAttachment,
219     context=[events.Context.SELF, events.Context.ROOT])
220   robot.register_handler(events.AnnotatedTextChanged, AskWords,
221     context=[events.Context.SELF],
222     filter=blip.Annotation.BACKGROUND_COLOR)
223   robot.register_handler(events.FormButtonClicked, ReadAnswer,
224     context=[events.Context.SELF, events.Context.ROOT])
225   robot.register_handler(events.OperationError, SendError)
226
227   # GAE を使用してロボットの処理を開始
228   appengine_robot_runner.run(robot)
```

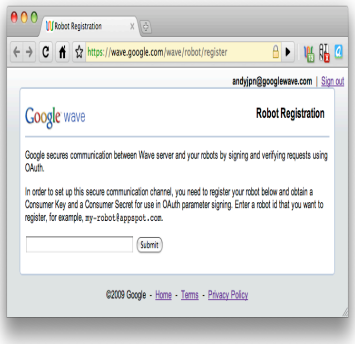
4.3.6 積極的なロボット

これまでのロボットは wave 内で参加者が何かアクションを起こしたことに反応して処理を実行するだけでした。しかしロボットによっては、例えば定期的に wave に情報を投稿したり twitter に DM が届いたときに wave で通知したりと、wave 内のイベントとは無関係に処理を実行したいこともあるでしょう。そういったことを実現するのが本節で紹介するアクティブロボット API です。ここからは単語帳ロボットとはまた別のロボットを作りながらその Active API の機能を確認していきます。

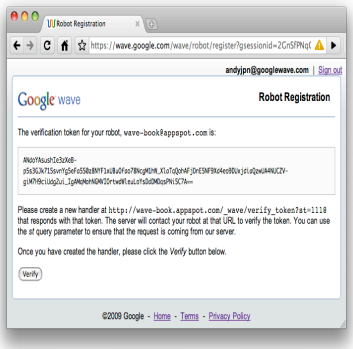
4.3.6.1 Active API の準備

Google Wave はロボットが Active API を用いて情報にアクセスする権限があるかどうかを OAuth プロトコルを用いてチェックします。そのため Active API を利用するロボットを作る前に、下記のサイトからロボットが使用するコンシューマ・キーとコンシューマ・シークレットを取得しておく必要があります。

- https://wave.google.com/wave/robot/register



テキストフィールドにロボットのアドレスを入力して「Submit」ボタンをクリックすると、コンシューマ・キーとコンシューマ・シークレットを得るための照合用トークンが発行されます。



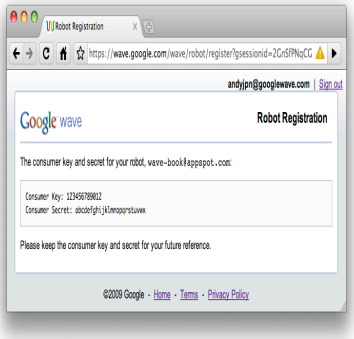
要求者が実際にロボットの管理者である事を示すために、wave サーバーは「http://[ロボット ID].appspot.com/_wave/verify_token?st=[セキュリティトークン]」にアクセス

し、その結果として先ほど発行された照合用トークンが返ってくることを確認します。URL とトークンの紐付けはどのようなやり方で実現しても構いませんが[*43]、ロボットクラスには照合用トークンを返すためのメソッドが用意されていますから、今回は素直にそれを使いましょう。なお照合用トークン発行ページはすぐ後で使用します。開いたままにしておいてください。

照合用トークンを返すために次のようなロボットを作成します。verification_token と security_token は先ほどのページに表示されているものを使用してください。security_token は指定された URL の st パラメータの値のことです。

```
 1  from waveapi import robot
 2  from waveapi import appengine_robot_runner
 3
 4  VERIFICATION_TOKEN = 'abcdef...uvwxyz'
 5  SECURITY_TOKEN = '1234'
 6
 7  if __name__ == '__main__':
 8    robot = robot.Robot('Beginner\'s Guide to Wave')
 9    robot.set_verification_token_info(VERIFICATION_TOKEN,
10      SECURITY_TOKEN)
11    appengine_robot_runner.run(robot)
```

ローカルサーバーを起動しブラウザで http://localhost:[ポート番号]/_wave/verify_token?st=[セキュリティトークン] にアクセスして照合用トークンが表示されることを確認してください。確認できたら GAE にデプロイし、開いたままにしておいた照合用トークン発行ページ下側の「Verify」ボタンをクリックします。問題がなければ遷移後の画面にコンシューマ・キーとコンシューマ・シークレットが表示されているはずです。



もし「Unable to verify the ownership of your robot.」というエラーが出た場合はトークンが間違っているか、トークンの発行から Verify までに時間が経ちすぎたのかもしれません。その場合はもう一度照合用トークンを発行するところからやり直してください。

無事にコンシューマ・キーとコンシューマ・シークレットが取得できたらロボットの set_verification_token_info メソッド呼び出しはもう不要です。削除しておきましょう。

[*43] 例えば照合用トークンをファイルに保存し、app.yaml で URL と紐付けるだけでも構いません

4.3.6.2 Active API

キーとシークレットが手に入ったので、早速 Active API を使って wave を外部から操作してみましょう。イベント駆動のロボットは app.yaml で/_wave/.*という URL に関連付け、robot.Robot オブジェクトを appengine_robot_runner で起動する必要がありましたが、Active API にそのような制約はありません。好きなところで API を使って wave の情報を取得したり wave に情報をプッシュできます。せっかくなので二つ目のサンプルは少し趣向を変えて、GAE の XMPP サービスを利用して GTalk と Google Wave を連携してみることにします。

4.3.6.3 XMPP サービス

GAE では XMPP サービスを利用して GTalk などのチャットサービスとメッセージをやり取りすることができます。まずは Active Robot API は使用せず、GAE で XMPP サービスを使う例を見てみましょう。

```
1  application: アプリケーション ID
2  version: 1
3  runtime: python
4  api_version: 1
5
6  handlers:
7  - url: .*
8    script: main.py
9
10 inbound_services:
11 - xmpp_message
```

XMPP を使用する場合は app.yaml の inbound_services で xmpp_message を設定してください。

```
1  # -*- coding: utf-8 -*-
2  from google.appengine.ext import webapp
3  from google.appengine.ext.webapp import util
4  from google.appengine.ext.webapp import xmpp_handlers
5
6  class XmppHandler(xmpp_handlers.CommandHandler):
7    def help_command(self, message=None):
8      message.reply(u'頼ろうなんて・・そんな気持ちだから見失う、大事なこと・・・！')
9
10   def text_message(self, message=None):
11     message.reply(u'%s ！そういうのもあるのか！' % message.arg)
12
13 def main():
14   application = webapp.WSGIApplication([
```

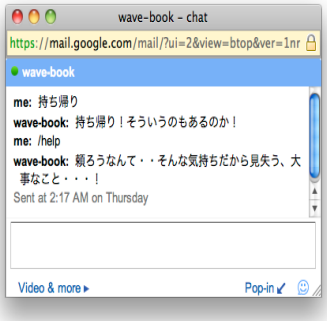
```
15      ('/_ah/xmpp/message/chat/', XmppHandler)
16    ], debug=True)
17    util.run_wsgi_app(application)
18
19  if __name__ == '__main__':
20    main()
```

GAE では XMPP メッセージは/_ah/xmpp/message/chat/にマッピングされます。まずはじめに WSGI アプリケーションのコンストラクタにその URL と XMPP のハンドラを渡します。XMPP の処理を簡単に書けるようにするため、GAE には xmpp_handlers.CommandHandler というハンドラクラスが用意されています。XMPP のハンドラはこのクラスを継承しましょう。もちろん import 文も忘れないようにします。

XMPP ハンドラはコマンドも処理でき、例えばユーザーが「/foo」で始まるメッセージを入力すると、ハンドラクラスの foo_command メソッドが呼び出されます。先のサンプルでは「/help」コマンドを処理するための help_command メソッドを定義しています。コマンド以外のメッセージは全て text_message メソッドで処理されます。また、ハンドラメソッドの message 引数は次のようなプロパティとメソッドを持っています。サンプルを参考に利用してください。

sender　メッセージ送信者アドレス
arg　コマンドの引数またはテキストメッセージ全文
reply()　メッセージに返信する

それでは実際に GTalk からメッセージを送ってみましょう。GTalk から GAE にアクセスするアドレスは wave ロボットと同じ、[アプリケーション ID]@appspot.com です。アプリケーションをデプロイした後、そのアドレスに対してメッセージを送ってください。



上のように反応が帰ってくれれば成功です。このように GAE を利用すれば非常に簡単にチャットボットを作れます。

4.3.6.4 モデルと定数の定義

ここから GTalk でコマンドやメッセージを入力して wave を操作できるようにしていきますが、その前にイベントハンドラで使用する定数とモデルを定義しておきます。

```
1  from waveapi import robot
2  from google.appengine.ext import db
3
4  # Web サイトで取得したキーとシークレット
5  CONSUMER_KEY = '123456789012'
6  CONSUMER_SECRET = 'abcdefghijklmnopqrstuvwx'
7
8  # プレビュー用
9  DOMAIN = 'googlewave.com'
10 RPC_BASE = 'http://gmodules.com/api/rpc'
11
12 # 参照中の Wave を保持するためのモデル
13 class WaveletRef(db.Model):
14   wave_id = db.StringProperty(required=True)
15   wavelet_id = db.StringProperty(required=True)
16   xmpp_address = db.StringProperty(required=True)
17   updated_at = db.DateTimeProperty(auto_now=True)
```

コンシュマー・キーとコンシューマ・シークレットは先ほど取得したものです。wave サーバーのドメインと RPC 送信先は、wave のサンドボックスを利用するのか通常の wave を利用するのかで異なる値を設定する必要があり、サンプルには通常用の値を設定しています。サンドボックスでサンプルを実行する場合には次の表??を参考に値を書き換えてください。

	ドメイン	RPC 送信先
プレビュー	googlewave.com	http://gmodules.com/api/rpc
サンドボックス	wavesandbox.com	http://sandbox.gmodules.com/api/rpc

表 4.5 Active API の設定

WaveletRef モデルはロボットが操作中の wave を記憶しておくために使用します。

4.3.6.5 新しく wave を作成する

最初に wave を新規作成する/new コマンドを実装してみましょう。コマンドを実装するには XMPP ハンドラクラスに [コマンド名]_command というメソッド、今回であれば new_command メソッドを実装します。

```
1  # Active API を利用するための OAuth 設定を済ませたロボットを返す
2  def get_active_robot(self):
3    active_robot = robot.Robot('xmpp')
```

```
4     active_robot.setup_oauth(CONSUMER_KEY, CONSUMER_SECRET,
5       server_rpc_base=RPC_BASE)
6     return active_robot
7
8   # 新しい Wave を作成
9   def new_command(self, message=None):
10    active_robot = self.get_active_robot()
11
12    # 新しい Wave を作成
13    wavelet = active_robot.new_wave(domain=DOMAIN, submit=True)
14
15    # 作成した Wave の情報を保持
16    wavelet_ref = WaveletRef(
17      wave_id=wavelet.wave_id,
18      wavelet_id=wavelet.wavelet_id,
19      xmpp_address=message.sender
20    )
21    wavelet_ref.put()
22
23    # 結果をユーザーに返信
24    message.reply(u'新しい Wave を作成しました:%s' % wavelet.wave_id)
```

ロボットオブジェクトを生成する部分は Active API を利用する全てのメソッドで必要なので get_active_robot メソッドとして切り出しました。Acitive API を使用するロボットは事前に Robot#setup_oauth メソッドを使用してコンシューマー・キーとコンシューマー・シークレットを設定しておかなければいけません。

setup_oauth メソッドで OAuth の設定を済ませると、Robot オブジェクトの new_wave メソッドを呼び出すだけで新しい wave が作成できます。ただし、デフォルトの設定では wave の作成は非同期です。今回は作成した wave に後からアクセスするために ID をデータストアに保存したかったので、引数の submit を True に設定しています。submit がデフォルトの False のままだと new_wave メソッド実行直後はまだ ID が決定していないため、wave_id プロパティには TBD（To Be Done）というプレースホルダしか入っていません。注意してください。

4.3.6.6 既存の wave を読み取って変更する

先の項で実装した/new コマンドによって GTalk から wave を新規作成できるようになったはずです。ただ、今の状態ではその wave の参加者がロボットしかいないため、内容を確認するどころか本当に存在するかどうかすらわかりません。今度は既存の wave に指定した参加者を追加する/invite コマンドを実装しましょう。

```
# メッセージ送信者の最新の WaveletRef を取得
def get_wavelet_ref(self, xmpp_address):
  return WaveletRef.gql(
    'WHERE wave_address = :wave_address ORDER BY updated_at DESC',
    xmpp_address=xmpp_address,
  ).get()

# Wave に参加者を追加
def invite_command(self, message=None):
  wavelet_ref = self.get_wavelet_ref(message.sender)
  active_robot = self.get_active_robot()

  # wavelet を Wave サーバーから取得
  wavelet = active_robot.fetch_wavelet(wavelet_ref.wave_id,
    wavelet_ref.wavelet_id)

  # wavelet に参加者を追加
  wave_address = message.arg.strip()
  wavelet.participants.add(wave_address)

  # 処理を確定
  active_robot.submit(wavelet)

  # 結果をユーザーに返信
  message.reply(u'参加者を追加しました:%s' % wave_address)
```

get_wavelet_ref メソッドは処理対象の Wavelet の ID をデータストアから取得します。簡単のために、そのユーザーが最も最近/new コマンドで作成した wavelet を処理対象にしています。

既存の wavelet に対して処理をする場合は、まずはじめに Robot#fetch_wavelet メソッドを使用して waveliet を取得してください。ここでは取得した wavelet に参加者を追加しています。/invite コマンドの引数は新規参加差の wave アドレスで、message.arg で取得できます。参加者を追加すると、最後に submit メソッドを実行します。イベント駆動のロボットの場合はハンドラの終了時にそれまで wave に行った変更が自動的に wave サーバーへ送られますが、Active API を使用した変更は Robot#submit() メソッドまたは Wavelet#submit_with() メソッドを明示的に呼ぶまで、処理が確定しないことに注意しましょう。

これで GTalk で新しく wave を作成して、さらにそのことを Google Wave 上で確認できるようになりました。GTalk 上で次のように発言して、[*44]Google Wave を開くと Inbox の一番上にロボットと自分が参加している新しい wave が表示されているはずです。

4.3.6.7　ユーザーの代理として動作する

GTalk から wave を作成できると言っても、なにも書かれていない wave 送るだけではあまり使い道がありません。今度は GTalk から wave にメッセージを書き込めるようにしましょう。XMPP ハンドラではコマンド以外のメッセージは全て `text_message` メソッドで処理します。次のコードを追加して下さい。

```
1  # コマンド以外のすべての発言を Wave に送信する
2  def text_message(self, message=None):
3    wavelet_ref = self.get_wavelet_ref(message.sender)
4    active_robot = self.get_active_robot()
5
6    # Wave サーバーから情報を取得
7    wavelet = active_robot.fetch_wavelet(wavelet_ref.wave_id,
8      wavelet_ref.wavelet_id)
9
10   # メッセージ送信
11   wavelet.reply().append(message.arg)
```

[*44] もちろん/invite する wave アドレスは自分のものに変更してください

```
12
13      # 変更を確定
14      active_robot.submit(wavelet)
```

fetch_wavelet メソッドで取得した wavelet を使ってイベント駆動ロボットと同様に返信を作成したあとで submit メソッドを実行するだけ、非常に簡単です。しかしこの実装には一つ問題があります。GTalk でこのロボットを利用して wave にメッセージを書き込めるのは一人ではありません。ところがこのままだと GTalk を経由の発言が誰のものであっても全て同じロボットの発言として書き込まれてしまいます。他の参加者が wave を見たときに、ロボットの発言が実際には誰の発言を代弁しているのか判断できた方がいいことは明らかでしょう。それを実現するのが Proxy For という機能です。

ロボットまたは操作にプロキシ ID を設定すると、その操作は [アプリケーション ID]+[プロキシ ID]@appspot.com というアドレスを持つロボットが実行したものとみなされます。プロキシ ID が異なるロボットは Google Wave 内で異なるロボットとして扱われ、参加者リストにも複数のロボットアイコンが並びます[*45]。

それでは先ほどのソースコードを、Proxy For を使うように修正してみましょう。wavelet.reply().append(message.arg) を次のように変更してください。

```
1  # wavelet に Proxy For ID を設定してからメッセージ送信
2  wavelet.robot_address = 'アプリケーション ID@appspot.com'
3  proxy_for_id = message.sender.split('@')[0]
4  wavelet.proxy_for(proxy_for_id).reply().append(message.arg)
```

reply メソッドを実行する前に wavelet の proxy_for メソッドでプロキシ ID を設定するだけです。ただし、Active API を利用したロボットで Proxy For を使用するには wavelet に明示的にロボットのアドレスを設定しておく必要があります。プロキシ ID としては発言者の名前（アドレスの@マークより前の部分）を使用しました。これでこのメッセージは [l アプリケーション ID]+[プロキシ ID]@appspot.com というロボットが実行したものとみなされます。早速試してみましょう。

参加者リストに同じロボットアイコンが二つ見えますが、片方はロボットアドレスに本当の発言者の名前である +andoyasusi が追加されています。発言者情報ダイアログで確認して下さい。

[*45] 同じロボットアイコンが複数並んでしまいますが、ちょっとした Hack でそれぞれ異なるアイコンを表示させることもできます

なお proxy_for を利用する方法はこれだけではなく、Blip オブジェクトにも proxy_for メソッドがありますし、Robot オブジェクトの Active メソッドの引数としても proxy_for を受け渡せます。外部サービスのプロキシとして動作するロボットを作成するときにはこれらの機能を利用してできるだけ情報元を明示するようにした方がいいでしょう。

4.3.6.8　全ソースコード

最後にアクティブロボットのソースコード全文を掲載します。

```
 1  # -*- coding: utf-8 -*-
 2  from google.appengine.ext import webapp
 3  from google.appengine.ext import db
 4  from google.appengine.ext.webapp import util
 5  from google.appengine.ext.webapp import template
 6  from google.appengine.ext.webapp import xmpp_handlers
 7  from waveapi import robot
 8  import os
 9
10  CONSUMER_KEY = '123456789012'
11  CONSUMER_SECRET = 'abcdefghijklmnoparstuvwx'
12  DOMAIN = 'googlewave.com'
13  RPC_BASE = 'http://gmodules.com/api/rpc'
14
15  # wave ID を保持するためのモデル
16  class WaveletRef(db.Model):
17    wave_id = db.StringProperty(required=True)
18    wavelet_id = db.StringProperty(required=True)
19    wave_address = db.StringProperty(required=True)
20    updated_at = db.DateTimeProperty(auto_now=True)
21
22  class XmppHandler(xmpp_handlers.CommandHandler):
23    # メッセージ送信者の最新の WaveletRef を取得
24    def get_wavelet_ref(self, mail_address):
```

```
25      return WaveletRef.gql(
26        'WHERE wave_address = :wave_address ORDER BY updated_at
DESC',
27        wave_address=mail_address,
28      ).get()
29
30    # Active API を利用するための OAuth 設定を済ませたロボットを返す
31    def get_active_robot(self):
32      active_robot = robot.Robot('xmpp')
33      active_robot.setup_oauth(CONSUMER_KEY, CONSUMER_SECRET,
34        server_rpc_base=RPC_BASE)
35      return active_robot
36
37    # 新しい Wave を作成
38    def new_command(self, message=None):
39      active_robot = self.get_active_robot()
40
41      # 新しい Wave を作成
42      wavelet = active_robot.new_wave(domain=DOMAIN, submit=True)
43
44      # 作成した Wave の情報を保持
45      wavelet_ref = WaveletRef(
46        wave_id=wavelet.wave_id,
47        wavelet_id=wavelet.wavelet_id,
48        wave_address=message.sender
49      )
50      wavelet_ref.put()
51
52      # 結果をユーザーに返信
53      message.reply(u'新しい Wave を作成しました:%s' % wavelet.wave_id)
54
55    # Wave に参加者を追加
56    def invite_command(self, message=None):
57      wavelet_ref = self.get_wavelet_ref(message.sender)
58      active_robot = self.get_active_robot()
59
60      # wavelet を Wave サーバーから取得
61      wavelet = active_robot.fetch_wavelet(wavelet_ref.wave_id,
62        wavelet_ref.wavelet_id)
63
64      # wavelet に参加者を追加
65      wave_address = message.arg.strip()
```

```
66      wavelet.participants.add(wave_address)
67
68      # 処理を確定
69      active_robot.submit(wavelet)
70
71      # 結果をユーザーに返信
72      message.reply(u'参加者を追加しました:%s' % wave_address)
73
74    # コマンド以外のすべての発言を Wave に送信する
75    def text_message(self, message=None):
76      wavelet_ref = self.get_wavelet_ref(message.sender)
77      active_robot = self.get_active_robot()
78
79      # Wave サーバーから情報を取得
80      wavelet = active_robot.fetch_wavelet(wavelet_ref.wave_id,
81        wavelet_ref.wavelet_id)
82
83      # wavelet に Proxy For ID を設定してからメッセージ送信
84      wavelet.robot_address = 'アプリケーション ID@appspot.com'
85      proxy_for_id = message.sender.split('@')[0]
86      wavelet.proxy_for(proxy_for_id).reply().append(message.arg)
87
88      # 変更を確定
89      active_robot.submit(wavelet)
90
91  def main():
92    application = webapp.WSGIApplication([
93      ('/_ah/xmpp/message/chat/', XmppHandler)
94    ], debug=True)
95    util.run_wsgi_app(application)
96
97  if __name__ == '__main__':
98    main()
```

4.3.7　GAE 以外のサーバーでロボットを動かす

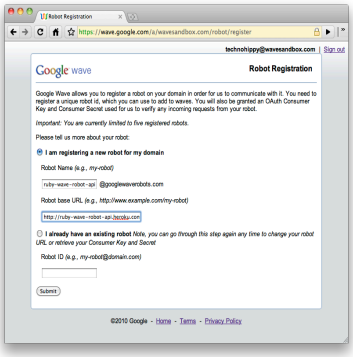
ここまでの 2 つのロボットはどちらも GAE 上で動かすものでしたが、wave のロボットは GAE 以外のサーバー上で動作させることもできます。また、公式のライブラリを使わないのであれば、ロボットを実装する言語も自由です。本節では GAE 以外のサーバーで Ruby を利用したロボットを動かす簡単な例を紹介します。[*46]

[*46] 現在のところ GAE 以外で動かすロボットはサンドボックス上でしか動作しません。今後、通常の wave でも可能になった場合は、特に事前の準備に関してはまた違った手順になる可能性もあります。ご注意く

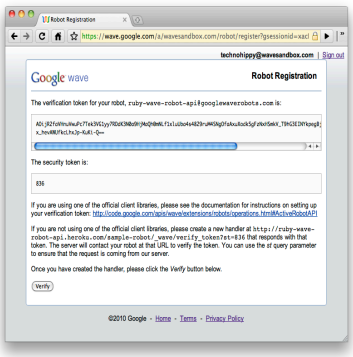
4.3.7.1 ロボットを登録する

GAE 以外のサーバーでロボットを動かす場合、ロボットの名前とアプリケーションのベース URL を事前に登録しなければいけません。次のアドレスにアクセスしてください。

- https://wave.google.com/a/wavesandbox.com/robot/register



上記のサイトでロボットの名前とベース URL を登録します。ロボットの名前はロボットアドレスに使用され、ベース URL は wave サーバーからロボットサーバーへのアクセスで使用されます。ベース URL はルートでなくても構いません。従って、一つのドメインで複数のロボットを動かすことも可能です。

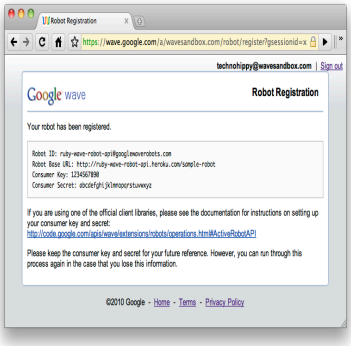


名前とベース URL を登録すると、アクティブ API のときと同様に照合用トークンとセキュリティトークンが表示されます。「http://[ベース URL]/_wave/verify_token?st=[セキュリティトークン]」にアクセスされたときに照合用トークンを返すように準備をして「Verify」ボタンをクリックしてください。今回の例では次のようにして照合用トークンを返しました。

```
1  require 'rubygems'
2  require 'sinatra'
3
4  get '/sample-robot/_wave/verify_token' do
```

ださい。

```
5    'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZabcd...'
6  end
```



トークンの照合に成功するとコンシューマ・キーとコンシューマ・シークレットが発行されます。以上でロボットを動かす準備は完了です。

4.3.7.2 ロボットを作成する

それでは Ruby でロボットを作成します。ロボットが対応する必要がある URL は次の 3 つです。

- http://[ベース URL]/_wave/robot/profile
- http://[ベース URL]/_wave/capabilities.xml
- http://[ベース URL]/_wave/robot/jsonrpc

まずは Ruby 製のロボットがどのようなファイルで構成されているのかを見ておきましょう。

```
./
|-- config.ru             : Rack の設定ファイル
|-- ruby-robot.rb         : ロボット本体
`-- views                 : テンプレートディレクトリ
    |-- profile.erb       : プロフィールテンプレート
    |-- capabilities.erb  : capabilities.xml テンプレート
    `-- jsonrpc.erb       : JSON-RPC レスポンステンプレート
```

ソースコードの詳細を理解する必要はありませんが、一応それぞれのファイルの内容を見ておきましょう。

```
1  require 'ruby-robot'
2  run Sinatra::Application
```

```
1  require 'rubygems'
2  require 'sinatra'
3  require 'erb'
4
5  VERSION = 1
6
```

```
get '/' do
  'Ruby Wave Robot'
end

get '/sample-robot/_wave/robot/profile' do
  content_type :json
  erb :profile
end

get '/sample-robot/_wave/capabilities.xml' do
  content_type :xml
  erb :capabilities
end

post '/sample-robot/_wave/robot/jsonrpc' do
  content_type :json
  wave_id = $1 if request.body.read =~ /"waveId":"(.*?)"/
  erb :jsonrpc, {}, :wave_id => wave_id
end
```

```
{
  'name': 'Ruby Wave Robot Sample',
  'imageUrl': 'http://ruby-wave-robot.heroku.com/images/icon.png',
  'profileUrl': 'http://ruby-wave-robot.heroku.com'
}
```

```
<?xml version="1.0"?>
<w:robot xmlns:w="http://wave.google.com/extensions/robots/1.0">
<w:version><%= VERSION %></w:version>
<w:protocolversion>0.21</w:protocolversion>
<w:capabilities>
  <w:capability name="BLIP_SUBMITTED" />
</w:capabilities>
</w:robot>
```

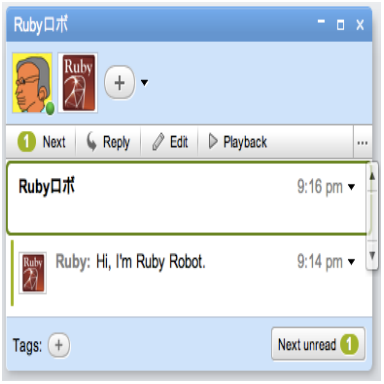
```
[
  {
    "id": "0",
    "method": "robot.notifyCapabilitiesHash",
    "params": {
      "capabilitiesHash": "<%= VERSION %>",
      "protocolVersion": "0.21"
    }
  },
  {
```

```
11      "id": "op2",
12      "method": "wavelet.appendBlip",
13      "params": {
14        "waveletId": "wavesandbox.com!conv+root",
15        "waveId": "<%= wave_id %>",
16        "blipData": {
17          "waveletId": "wavesandbox.com!conv+root",
18          "blipId": "TBD_wavesandbox.com!conv+root_0x4233e6090834f589",
19          "waveId": "<%= wave_id %>",
20          "content": "Hi, I'm Ruby Robot.",
21          "parentBlipId": null
22        }
23      }
24    }
25  ]
```

capabilities.erb を見ると分かるように、このロボットは wave 内で BLIP_SUBMITTED イベントが起きたときに起動するイベント駆動型のロボットです。残念ながら Ruby 用のライブラリを見つけられなかったので、簡単な Python ロボットのログから出力をコピーして、JSON-RPC のレスポンスを作成しました。元になった Python 版のロボットのソースコードは次です。

```
1  # -*- coding: utf-8 -*-
2  from waveapi import events
3  from waveapi import robot
4  from waveapi import appengine_robot_runner
5
6  def OnBlipSubmitted(event, wavelet):
7    wavelet.reply("Hi, I'm Ruby Robot.")
8
9  if __name__ == '__main__':
10   robot = robot.Robot('Ruby Wave Robot Dummy')
11   robot.register_handler(events.BlipSubmitted, OnBlipSubmitted)
12   appengine_robot_runner.run(robot)
```

つまりこの Ruby ロボットは blip が確定されたときに wavelet の一番最後に「Hi, I'm Ruby Robot.」という返信を追加します。実際に動作させてみましょう。



プロフィールを見ると確かに GAE 以外のサーバーで動いています。

このように GAE 以外のサーバーでもライブラリさえ存在すれば簡単に wave ロボットを動かすことができます。

4.3.8 おわりに

ロボットは wave の情報を読み取ったり内容を更新したり、まるでユーザーのように振る舞えるだけではなく、Web サービスとして外部サービスと連携することもできます。さらにガジェットと組み合わせれば Google Wave のデフォルトの UI には存在しない新しいインターフェースをユーザーに提供することまで可能です。もしかするとロボットは人間の参加者以上に wave の力を引き出すことができるのかもしれません。ロボットにどこまでのことができるのか、全てはみなさんの想像力にかかっています。本書の説明が Google Wave の可能性を広げる役に立てば嬉しく思います。

4.3.9 API リファレンス

ここにはよく使うクラスの API だけを掲載しています。全クラスのリファレンスが必要な場合は公式のリファレンス[*47]を参照してください。

[*47] http://wave-robot-python-client.googlecode.com/svn/trunk/pydocs/index.html

4.3.9.1　robot.Robot

ロボットの名前やプロフィールなどの情報を管理したり、イベントハンドラを管理してイベントに応じたハンドラを起動するクラス。

コンストラクタ

robot.Robot(name, image_url='', profile_url='http://code.google.com/apis/wave/extensions/rtutorial.html')

プロパティ

name　ロボットの名前
image_url　アバターとして使用される画像の URL
profile_url　ロボットの詳細情報を記載したウェブページの URL

メソッド

get_verification_token_info()　照合用のトークンとセキュリティトークンを返します
capabilities_hash()　capabilitiy_hash を 16 進文字列で返します
register_handler(event_class, handler, context=None, filter=None)　特定のイベントタイプに対するハンドラを登録します。一つのイベントに対して複数のハンドラを登録することもでき、その場合は登録した順で実行されます。ハンドラはイベントオブジェクトと関連する wavelet を引数にとります
set_verification_token_info(token, st=None)　ロボットのオーナーシップを確認するための照合用トークンを設定します
set_http_post(url, data, headers)　HTTP POST リクエストを実行するときに使用するハンドラを設定します
setup_oauth(consumer_key, consumer_secret, server_rpc_base='http://gmodules.com/api/rpc')　ロボットが OAuth で認可された JSON RPC を実行するための設定をします
register_profile_handler(handler)　ロボットのプロフィールハンドラを登録します。プロフィールハンドラはプロフィールが必要なときに呼び出されます。ハンドラにはプロフィールが必要な（proxy_for された）名前か、ロボット自身のプロフィールが必要な場合には None が渡されるので、name と imageUrl と profileUrl をキーに持つ辞書オブジェクトを返さなければいけません
make_rpc(operations)　RPC コールを発行して、指定されたオペレーションをサブミットします
capabilities_xml()　ロボットの capabilities を XML 文字列で返します
profile_json(name=None)　プロフィールを JSON 形式で返します。このメソッドはロボットの基本プロフィールと、proxy_for で name が設定されているときのプロフィールの両方で呼び出されます。デフォルトでは登録時に与えられた情報を返し

ます。デフォルトの挙動を変更するには register_profile_handler を上書きしてください

process_events(json) JSON でエンコードされたイベントの組を受け取って処理します

new_wave(domain, participants=None, message='', proxy_for_id=None, submit=False) 与えられた初期参加者を持つ新規 wave を作成します。このメソッドは必要なオペレーションキューを返すだけで、そのオペレーションをサーバーにサブミットする責務は呼び出し側が持ちます。返り値に対して robot.submit() か .submit_with() を呼び出してください

fetch_wavelet(wave_id, wavelet_id, proxy_for_id=None) REST インターフェースで wave をフェッチしてそれを返します。返り値である wavelet はその時点での wavelet の状態のスナップショットです。wavelet を変更することもできますが、フェッチ後に変更されていることもあるので注意が必要です。また wavelet はそのオペレーションキューを返すだけで、robot.submit() または.submit_with() を呼び出してサーバーに実際に wavelet への変更をサブミットするのは呼び出し元の責務です

blind_wavelet(json, proxy_for_id=None) JSON 文字列かブラインド wave を構成します。イベントコールバックの外で処理したい wave のスナップショットがあるときはこのメソッドを呼んでください。最後に見たあとで wave が変更されていることもあるので、できるだけ安全なオペレーションをサブミットするように気をつけなければいけません

submit(wavelet_to_submit) wavelet_to_submit に関連したまだ実行されていないオペレーションをサブミットします。典型的なケースでは wavelet は fetch_wavelet または blind_wavelet または new_wavelet の返り値です

4.3.9.2 wavelet.Wavelet

単一の wavelet のためのモデルです。

wavelet はメタデータ、参加者、blip で構成されます。全ての blip を使用可能にするにはイベントを登録する際に Context.ALL を指定しなければいけません。

コンストラクタ

wavelet.Wavelet(json, blips, robot, operation_queue)

プロパティ

blips この wavelet の持つ全 blip

creation_time この wavelet が初めて作られた時間

creator この wavelet の作成者の参加者 ID

data_documents この wavelet のデータドキュメント

domain この wavelet が属するドメイン

last_modified_time この wavelet が最後に変更された時間

participants　この wavelet の全参加者
robot　この wavelet を所有するロボット
robot_address　この wavelet を所有するロボットのアドレス
root_blip　この wavelet のルート blip
tags　この wavelet のタグのリスト
title　この wavelet のタイトル
wave_id　この wavelet の ID
wavelet_id　この wavelet の親 wave の ID

メソッド

get_operation_queue()　この wavelet の `OperationQueue` を返します
serialize()　この wavelet のプロパティの辞書を返します
proxy_for(proxy_for_id)　特定の ID のプロキシとして働くこの wavelet のビューを返します。`proxy_for_id` が設定された、現在の wavelet の浅いコピーを返します。このコピーに対する全ての変更は `proxy_for_id` を使って実行されます。つまり、`robot+<proxy_for_id>@appspot.com` がアドレスとして使用されます
add_proxying_participant(id)　wave にプロキシ参加者を追加します。プロキシ参加者とは `robot+proxy@domain.com` という形式の参加者です。このメソッドは `id` を使ってアドレスを作成したあとで、`participants.add` を呼び出します
submit_with(other_wavelet)　この wavelet を引数として受け取った他の wavelet がサブミットされるのと同時にサブミットします。イベントコールバックの外で構成した wavelet は `robot.submit(wavelet)` を使って明示的にサブミットするか、後でサブミットされるかイベントコールバックに含まれている別の wavelet と関係付ける必要があります
reply(initial_content=None)　この wavelet の会話に返信します
delete(todelete)　この wavelet から blip を削除します

4.3.9.3　wavelet.BlipThread

wave 内の blip のグループを表すモデル

コンストラクタ

wavelet.BlipThread(id, location, blip_ids, all_blips, operation_queue)

プロパティ

blip_ids　スレッドに含まれる blip ID 一覧
blips　スレッドに含まれる blip 一覧
id　スレッド ID
location　スレッドの wavelet 内での位置

4.3.9.4 wavelet.DataDocs

たくさんのデータドキュメントを Python らしい形で扱うモデル

コンストラクタ

wavelet.DataDocs(init_docs, wave_id, wavelet_id, operation_queue)

メソッド

keys() データドキュメントのキーを返します
serialize() データドキュメントの辞書を返します

4.3.9.5 wavelet.Participants

参加者の組を Python らしいやり方で扱うためのモデル。

コンストラクタ

wavelet.Participants(participants, roles, wave_id, wavelet_id, operation_queue)

定数

ROLE_FULL フルアクセス（読み書き可能）権限
ROLE_READ_ONLY 読み込み専用権限

メソッド

add(participant_id) 参加者 ID（アドレス）を追加します
get_role(participant_id) `participant_id` で示される参加者の権限を返します
serialize() 参加者のリストを返します
set_role(participant_id, role) `participant_id` で示される参加者の権限を設定します

4.3.9.6 wavelet.Tags

タグのリストを表すモデル

コンストラクタ

wavelet.Tags(tags, wave_id, wavelet_id, operation_queue)

メソッド

append(tag)　tag がまだ存在しなければ追加します
emove(tag)　tag が存在すれば削除します
serialize()　タグのリストを返します

4.3.9.7　blip.Blip

一つの blip を表すモデル。

blip は本質的には会話を構成するドキュメントです。blip は階層を成していて、ルート blip は親 blip ID を持ちません。全ての bilp は自身が関連する wave と wavelet の ID を持っています。

blip はさらにドキュメントオブジェクトを通じて関連のあるアノテーション、コンテンツ、エレメントなども保持しています。

コンストラクタ

blip.Blip(json, other_blips, operation_queue)

プロパティ

annotations　ドキュメントのアノテーション
blip_id　この blip の ID
child_blip_ids　この blip の子 blip の ID
child_blips　この blip の子 blip
contributors　この blip を編集したことのある参加者の ID
creator　この blip を作成した参加者の ID
elements　ドキュメントのエレメント。ドキュメントのエレメントとはフォーム要素・ガジェットなどのプレーンなテキストとして表現できないもののことです。ドキュメントのエレメントプロパティはドキュメント内の位置からエレメントのインスタンスを引ける辞書のようなオブジェクトです。ドキュメントをテキストとして取得する際には、通常はエレメントの代わりに空白文字がプレースホルダとして入ります。
inline_blip_offset　この blip がインライン blip のときはオフセット。そうでなければ-1。親 blip がコンテキストに含まれていないときは判断ができないため常に-1 が返ります。
inline_reply_threads　blip へのインラインリプライのリスト
last_modified_time　blip が最後に変更された時間
parent_blip　親 blip。ルートのときは None
parent_blip_id　親 blip の ID。ルートのときは None
reply_thread　blip へリプライするスレッドのリスト

text　ドキュメントのテキストコンテンツ
version　この blip のバージョン
wave_id　この blip が所属する wave の ID
wavelet_id　この blip が所属する wavelet の ID

メソッド

is_root()　wavelet のルート blip かどうかを返します
all(findwhat=None, maxres=-1, **restrictions)　検索の全結果を表す BlipRefs オブジェクトを返します。あるエレメントを検索するなら、フィルタに使用する追加のエレメントプロパティ（例えば Gadget の url プロパティ）を第三引数で指定できます
first(findwhat=None, **restrictions)　検索の結果一件だけを表す BlipRefs オブジェクトを返します。あるエレメントを検索するなら、フィルタに使用する追加のエレメントプロパティ（例えば Gadget の url プロパティ）を第三引数で指定できます
at(index)　指定された位置にある 1 キャラクタの幅を持つ BlipRefs を返します
range(start, end)　範囲を表す BlipRefs オブジェクトを返します
serialize()　この blip を表す、json 形式に変換可能な辞書を返します
proxy_for(proxy_for_id)　特定の ID のプロキシとして働くこの blip のビューを返します。proxy_for_id が設定された現在の blip の浅いコピーが返ります。このコピーになされた全ての変更は proxy_for_id を使って実行されます。つまり、robot+<proxy_for_id>@appspot.com というアドレスが使用されます
find(what, **restrictions)　コンテンツのマッチした部分をイテレートします。エレメントかテキストの一部を返します
append(what)　要素を追加するための一般的なパターンに対応した便利メソッドです
reply()　この blip のリプライを生成して返します
append_markup(markup)　マークアップテキストを xhtml とみなして解釈し、ドキュメントに追加します
insert_inline_blip(position)　この blip の指定された位置にインライン blip を挿入します

4.3.9.8　blip.BlipRefs

blip のコンテンツの参照の集まりを表します。

例えば RlipRefs インスタンスは検索結果や、明示的に指定された範囲や、正規表現の結果を表す他、blip 全体を参照することもできます。BlipRefs は blip に対する操作を一貫したやり方でサーバーに送信しやすくするために使用されます。

BlipRefs オブジェクトを作成するための標準的は方法は、Blip オブジェクトのセレクタメソッドを使用することです。開発者が直接 BlipRefs オブジェクトをインスタンス化することは通常ありません。

コンストラクタ

blip.BlipRefs(blip, maxres=1)

クラスメソッド

all(blip, findwhat, maxres=-1, **restrictions)　検索されたテキストまたは要素を表すインスタンスを生成します

range(blip, begin, end)　明示的な範囲を設定されたインスタンスを生成します

メソッド

insert(what)　マッチする位置に受け取ったものを挿入します

insert_after(what)　マッチする位置の直後に受け取ったものを挿入します

replace(what)　マッチする位置を受け取ったものと置換します

delete()　マッチする位置にあるコンテンツを削除します

annotate(name, value=None)　マッチする位置にあるコンテンツにアノテーションを付加します

clear_annotation(name)　マッチする位置のアノテーションを削除します

update_element(new_values)　引数の値で既存のエレメントを更新します

value()　`BlipRefs` を最初にマッチする値に変換する便利メソッドです

4.3.9.9　blip.Blips

blip ID をキーに複数の blip を管理する辞書に似たオブジェクト。

メソッド

get(blip_id, default_value=None)　`blip_id` を持つ `Blip` オブジェクトを返します。存在しない場合には、None か `default_value` を返します

serialize()　シリアライズした blip のリスト

values()　保持している blip のリストを返します

4.3.9.10　blip.Annotation

ドキュメントのアノテーションを表すモデルです。

アノテーションはコンテンツのある範囲に設定できるキーと値のペアです。アノテーションはデータを保存するのに使用したり、クライアントがその表示にあたって解釈する値を設定するのにしようできます。

コンストラクタ

blip.Annotation(name, value, start, end)

定数

BACKGROUND_COLOR（style/backgroundColor） テキストの背景色を設定するために予約されています

COLOR（style/color） テキストの色を設定するために予約されています

FONT_FAMILY（style/fontFamily） テキストのフォントファミリー（sans-serif、serif など）を設定するために予約されています

FONT_SIZE（style/fontSize） テキストのフォントサイズを設定するために予約されています

FONT_STYLE（style/fontStyle） テキストのスタイル（italic、oblique など）を設定するために予約されています

FONT_WEIGHT（style/fontWeight） テキストのフォントウェイト（太さ）を設定するために予約されています

TEXT_DECORATION（style/textDecoration） テキストの修飾（underline、line-through など）を設定するために予約されています

VERTICAL_ALIGN（style/verticalAlign） 垂直方向の文字揃えを設定するために予約されています

プロパティ

name アノテーションの名前

value アノテーションの値

start アノテーションが設定される前端点

end アノテーションが設定される後端点

メソッド

serialize() アノテーションをシリアライズします

4.3.9.11 blip.Annotations

アノテーション名をキーに複数のアノテーションを管理する辞書に似たオブジェクト。

メソッド

names() 保存されているアノテーションの名前

serialize() シリアライズしたアノテーションのリスト

4.3.9.12 waveservice.Waveservice

OAuth を利用して Wave サービスとやりとりするためのベースクラス。

コンストラクタ

waveservice.WaveService(use_sandbox=False, server_rpc_base=None, consumer_key='anonymous', consumer_secret='anonymous', http_post=None)

メソッド

blind_wavelet(json, proxy_for_id=None) wave のスナップショットを取っておいてイベントハンドラの外で処理するにはこのメソッドを利用します。wave の本体とスナップショットは同じ内容とは限らないので注意してください。

fetch_request_token(callback=None) OAuth ダンスを開始するためのリクエストトークンを取得します。

fetch_wavelet(wave_id, wavelet_id=None, proxy_for_id=None) REST インターフェースを利用して wavelet を取得します。取得される wavelet はスナップショットで、本体とは同じ内容を持つとは限らないので注意してください。また、ここで得られる wavelet は独自のオペレーションキューを持っているので、操作を確定するには開発者が自分で `robot.submit()` または wavelet.submit_with() を呼んでやる必要があります。

generate_authorization_url(request_token=None) OAuth の認証 URL を生成します

get_token_from_request(oauth_request) リクエストからトークンを取得します

http_post(url, data, headers) HTTP POST リクエストを実行します。必要に応じて置き換えてください。例えば `appengine_runner_runner` では GAE の機能を利用するようにこのメソッドを書き換えています。

make_rpc(operations) 指定された操作をサブミットする RPC 呼出を作成します

new_wave(domain, participants=None, message='', proxy_for_id=None, submit=False) 指定された初期参加者を持つ wave を新規作成します。ここで得られる wave は独自のオペレーションキューを持っているので、操作を確定するには開発者が自分で `robot.submit()` または wavelet.submit_with() を呼んでやる必要があります。

search(query, index=None, num_results=None) 検索リクエストを実行します。

set_access_token(access_token) OAuth アクセストークンを設定します。

set_http_post(http_post) HTTP POST ハンドラを設定します。

submit(wavelet_to_submit) `wavelet_to_submit` に関連する待ち状態のオペレーションをサブミットします。

upgrade_to_access_token(request_token, verifier=None) `request_token` をアクセストークンにアップグレードします。

4.3.9.13 search.Digest

wave のダイジェストを表すモデル。

コンストラクタ

search.Digest(json)

プロパティ

blip_count この wave 内の blip の数
unread_count この wave 内の未読 blip の数
last_modified wave が最後に変更された日
wave_id wave ID
snippet 本文の抜粋
domain wave が所属するドメイン
participants この wave の参加者一覧
title wave のタイトル

メソッド

serialize() ダイジェストのプロパティを辞書にしたものを返します

4.3.9.14 search.Results

検索結果を表すモデル

コンストラクタ

search.Results(json)

プロパティ

query 検索クエリ
num_results 検索結果数
digests ダイジェストのリスト

メソッド

serialize() ダイジェストのプロパティを辞書にしたものを返します

4.3.9.15　appengine_robot_runner モジュール

ロボットを Google AppEngine 上で動かすために使用するモジュールです。

関数

run(robot, debug=False, log_errors=True, extra_handlers=None)　引数として受け取ったロボット用の webapp ハンドラを設定して、リスンを開始します。ライブラリが作成する webapp ハンドラとデフォルトでマッピングされる URL は次の 4 つです。

- /_wave/capabilities.xml
- /_wave/robot/profile
- /_wave/robot/jsonrpc
- /_wave/verify_token

これ以外の URL を処理する場合は `extra_handlers` 引数に URL パターンとハンドラのタプルの配列を渡してください。

4.4　インストーラー

4.4.1　はじめに

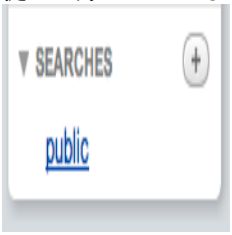
ここまでの説明ではロボットやガジェットを利用するには、ユーザーがそれぞれアドレスと URL を知っていなければいけませんでした。自分で作った拡張ならそれでも大丈夫かもしれませんが、実際には他の開発者が作った便利な拡張を利用することの方がずっと多いはずです。それらのアドレスと URL まで全部覚えておかなければ、拡張は利用できないのでしょうか？もちろん、そんなことはありません。

Google Wave にはインストーラーという仕組みがあり、このインストーラーを利用することで Google Wave にさまざまな追加機能をインストールできます。インストーラーはフックとアクションで構成され、フックは機能が拡張される場所、アクションが拡張される機能です。

フックには次のようなものがあります。

- エディターツールバーアイテム

- New wave サブメニュー


- 編集中テキスト。編集中のテキストが特定の正規表現にマッチしたら処理が実行されるように設定できます
- 保存された検索条件（本項目はフックと言うよりもアクションのショートカットと捉えた方がいいかもしれません）


アクションには次のようなものがあります。

- wave に参加者（含ロボット）を追加
- 保存された検索条件を追加
- 選択部分にアノテーション追加
- ガジェットを挿入

これらを組み合わせて、Google Wave に簡単に機能を追加する仕組みを提供するものがインストーラーです。

なお、これまでに説明したロボットやガジェットの仕様とは異なり、これから説明するインストーラーは Google Wave 独自のものです。もちろんまだ Wave クライアントは Google Wave だけですからそれほど意識する必要はありませんが、今後 Google 社以外による Wave クライアントが登場した際には全く異なる仕組みのインストーラーが提供されることもありえます。

4.4.2 エクステンションマニフェスト

インストーラーの実体はマニフェストと呼ばれる XML ファイルです。この XML ファイルが表示内容やコンポーネントの追加される場所、コンポーネントをクリックしたときに実行されるアクションなどを定義します。どのようなものなのかさっそく見てみましょう。

```
1  <extension
2      name="スケジュール調整ガジェット"
3      thumbnailUrl="http://.../gadget_installer_thumb.png"
```

```
 4      description="イベントスケジュールを調整します">
 5    <author name="あんどうやすし" />
 6    <menuHook location="toolbar" text="スケジュール調整ガジェット"
 7        iconUrl="http://.../gadget_installer_icon.png">
 8      <insertGadget url="http://.../scheduler.xml" />
 9    </menuHook>
10  </extension>
```

このマニフェストはスケジュール調整ガジェットを追加（<insertGadget>）するためのボタンを、エディタツールバーに追加（<menuHook location="toolbar">）します。この例からわかるように名前やサムネイル画像などインストーラー自身に関する情報はルート要素である extension タグの属性で指定され、クライアントの拡張される場所や、拡張される機能についてはその子要素で指定されます。

それではここからはこのマニフェストファイルの細かい仕様を見ていきましょう。[*48]

4.4.2.1　<extension>

<extension>タグはマニフェスト XML のルート要素です。インストーラーのメタ情報を定義するために次のような属性を使用できます。

name　拡張の名前
description　拡張の説明
infoUrl　拡張に関するより詳細な情報が記述されているサイトへのリンク
thumbnailUrl　インストーラーのジクソーパズル状の枠内部に表示される画像。画像サイズは 120x120px が推奨されます
version　拡張のバージョン。Wave サーバーはバージョンのチェックとして文字列比較を行うので、数字でなくても構いません。指定されていない場合は"0"とみなされます

<extension>タグの子要素に設定できるタグは現在のところ 4 種類です。

<author>　拡張の作者情報。属性として name と email が設定できます。作者を公開する必要がなければ省略して構いません
<menuHook>　メニューにアイテムを追加
<regexHook>　編集中のテキストが指定した正規表現にマッチした場合に実行されるアクションを追加
<savedSearchHook>　名前付きの検索条件を追加

4.4.3　フック

4.4.3.1　<menuHook>

Google Wave クライアントの拡張可能なポイントはフックと呼ばれます。メニューフックはその中でも Google Wave クライアントのメニューに UI を追加するために使用

[*48] 本節の例は全て http://code.google.com/intl/ja/apis/wave/extensions/installers/index.html に依りました

します。UI を追加する場所は location 属性で指定します。

属性

location 拡張する場所。設定できる値は次の二種類です

"toolbar" エディターツールバー

"newwavemenu" New wave サブメニュー

text ツールバーのツールチップに使用されるテキスト。New wave の場合は使用されません

iconUrl ツールバーに表示されるアイコン URL。New wave の場合は使用されません

例

```
1 <menuHook location="NewWaveMenu" text="New Friends Wave">
2   <addParticipants>
3     <participant id="jon@example.com"/>
4     <participant id="tom@example.com"/>
5   </addParticipants>
6 </menuHook>
```

4.4.3.2 <regexHook>

編集中のテキストが指定した正規表現にマッチした場合に実行されるアクションを追加するために使用します。例えば、電話番号や、住所などを正規表現で見つけて、それらを処理するロボットを wave に参加させる、といった利用方法が考えられます。

属性

pattern 正規表現パターン

例

```
1 <regexHook pattern="\:\)">
2   <addParticipants>
3     <participant id="skimmy-wave@appspot.com"/>
4   </addParticipants>
5 </regexHook>
```

4.4.3.3 <savedSearchHook>

このフックはユーザーの操作に反応するのではなく、このフックを含むインストーラーが実行されたときに名前付きの検索条件を保存します。ほとんどの場合、他のフックと同時に使用され、それらによってインストールされる拡張に関連する wave を見つけるための検索条件、with:[**ロボット名**] や gadgeturl:[**ガジェット** URL]、を SEARCHERS に追加します。

属性

name 検索条件の名前

query 検索クエリ

color (オプション) ラベルの背景色。CSS に指定できる背景色を設定します。背景色を

設定すると検索ペインのダイジェストの条件にマッチする wave にもその色つきのラベルが表示されます

例

```
1  <extension name="Unread Wave Search"
2      description="Installs a saved search that shows all unread
waves.">
3    <savedSearchHook name="Show unread" query="is:unread"
4        color="#00FF00"/>
5  </extension>
```

4.4.4 アクション

4.4.4.1 <addParticipants>

wave に参加者を追加します。参加者は子要素の<participant>タグを使用して複数人同時に追加できます。

例

```
1  <menuHook location="NewWaveMenu" text="New Chess Match">
2    <addParticipants>
3      <participant id="chess-bot@googlewaverobots.com"/>
4    </addParticipants>
5  </menuHook>
```

4.4.4.2 <addSavedSearch>

savedSearchHook と同様に、名前付きの検索条件を作成します。使用できる属性もフックの場合と同じです。現在のところ拡張をアンインストールしても保存した検索条件は削除されないので注意して下さい。

例

```
1  <menuHook location="toolbar" text="Add Chess Gadget"
2      iconUrl="http://wave-chess.appspot.com/WaveChess.png">
3    <insertGadget
4        url="http://wave-chess.appspot.com/wavechess/com.google.wave.chess.clie
5    <addSavedSearch title="Chess Game" query="has:gadget ChessGame"
6        color="green"/>
7  </menuHook>
```

4.4.4.3 <annotateSelection>

編集モードでテキスト選択中にこのアクションを実行すると選択部分にアノテーションを設定します。アノテーションのキーと値はそれぞれ key 属性と value 属性で指定します。定義されたアノテーションを設定してテキストフォーマットを変更する他、同時のアノテーションを設定してロボットに処理させることもできます。ロボットでアノテーションを扱う方法については??節を参照してください

例

```
1  <menuHook location="toolbar" text="Boldicize">
2    <annotateSelection key="style/fontWeight" value="bold"/>
3    <annotateSelection key="style/fontStyle" value="italic"/>
4  </menuHook>
```

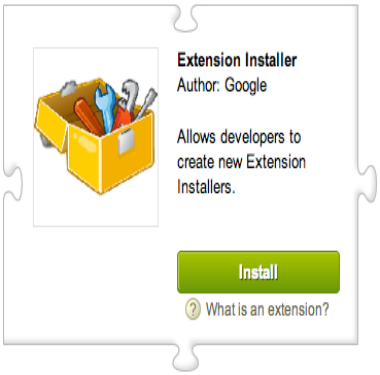
4.4.4.4 <insertGadget>

現在のカーソル位置にガジェットを挿入します。ガジェットの URL は `url` 属性で指定します。

例

```
1  <menuHook location="toolbar" text="Add Chess Gadget"
2      iconUrl="http://wave-chess.appspot.com/WaveChess.png">
3    <insertGadget
4        url="http://wave-chess.appspot.com/wavechess/com.google.wave.chess.client.ChessGadget.gadget.x
5  </menuHook>
```

4.4.5 エクステンションインストーラーを使用する



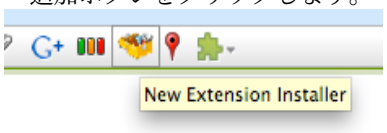
マニフェスト XML を元にインストーラーを wave に貼り付けるにはまず初めにエクステンションインストーラー（Extension Installer）をインストールして、Google Wave にインストーラーを貼り付けるための UI を追加します。つまり、エクステンションインストーラー（Extension Installer）自体が「Google Wave にインストーラーを追加するボタン」を追加するためのインストーラーになっています。説明を重ねても混乱が増すだけなので、さっそく具体的な説明に移りましょう。

エクステンションインストーラーはナビゲーションパネルの EXTENSIONS にあります。All をクリックして、検索ペインから「Extension Installer」を選択してください。

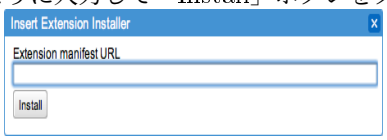
インストールするには「Install」ボタンをクリックするだけです。エクステンションインストーラーはツールバーにインストーラー挿入ボタンを作成します。  がツールバーに追加されていることを確認してください。

4.4.6　インストーラーを作成する

インストーラーを作成するには新しく wave を作成し、先ほど追加されたインストーラー追加ボタンをクリックします。



するとマニフェストファイルの URL を入力するダイアログが開くので、間違いのないように入力して「Install」ボタンをクリックしてください。



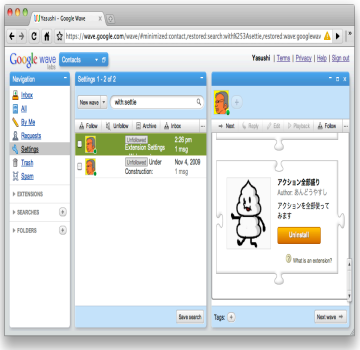
wave にマニフェストファイルで指定したとおりのインストーラーが挿入されます。

4.4.7　拡張をアンインストールする

インストールした拡張をアンインストールするにはどうしたらいいのでしょう。もしインストールに使用した wave がすぐに見つかるのなら簡単です。インストール済みの拡張

の場合は、インストーラーのボタンが「Install」ではなく「Uninstall」になっているので、そのボタンをクリックしてください。すぐにアンインストールが完了します。

インストールに使用した wave がどこにあるか分からないときには、Navigation ペインから Settings を選んでください。Inbox に Extension Settings というタイトルの wave があるはずです。そこにはこれまでにインストールした全ての拡張のインストーラーが揃っていますので、アンインストールしたい拡張のインストーラを探し、「Uninstall」ボタンをクリックするとその拡張をアンインストールできます。

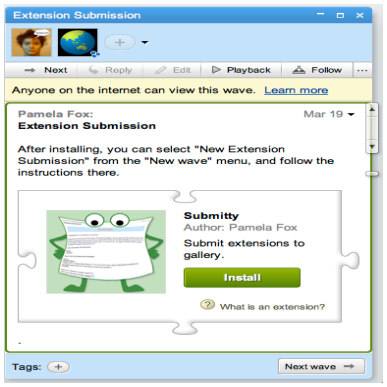


4.4.8　ギャラリーに登録する

拡張を作成し、インストーラーまで作成したのであれば、せっかくなのでもう少し頑張ってその拡張をギャラリーに登録し、できるだけ多くの人に使ってもらいましょう。本項では、インストーラー自体の説明からは少し外れますが、wave の拡張ギャラリーに自分の作った拡張を登録する手順を紹介します。

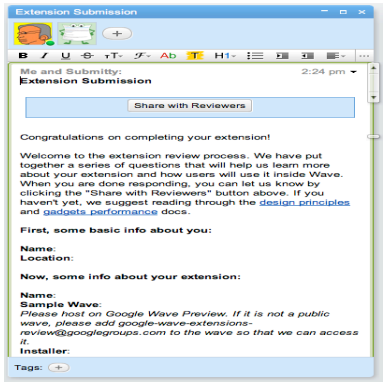
ギャラリーへの登録は Submitty という拡張を利用します。まずはその Submitty をインストールしましょう。Submitty インストーラーは「with:public "Extension Submission"」という条件で検索して見つけるか、下の URL の「Submitty extension」というリンクをクリックしてください。

- http://code.google.com/intl/ja/apis/wave/submitextension.html



以降の説明は上の URL の説明とほとんど同じになります。Submitty インストーラーが表示されたら「Install」ボタンをクリックして Submitty をインストールします。

Submitty をインストールすると New Wave メニューに「New Extension Submission」という項目が追加されます。ギャラリーに自分の拡張を投稿するにはその新しく追加された項目をクリックしてください。



登録に必要な入力項目は以下です。なお、レビュアーはおそらく日本語を理解できないので、これらは全て英語で記入する必要があります。

- 作者の基本情報
  Name　作者の名前
  Location　作者の住んでいる地域
- 拡張の情報

Name 拡張の名前

Sample Wave レビュアーが動作を確認するための wave の URL。サンプルはサンドボックスではなくプレビューに作成してください。レビュアーがアクセスできるように public にするか、参加者に google-wave-extensions-review@googlegroups.com を追加する必要があります

Installer 拡張のインストーラー URL

Does your extension include a robot? 拡張にロボットが含まれる場合はチェックしてください。チェックするとさらに次の入力項目が追加されます。

- ロボットはどういうイベントを受け取って、それぞれどのような反応を返すか
- 入力を変換するようなロボットの場合、ロボットの機能を利用するための特別な記法などはあるか

Does your extension include a gadget? 拡張にロボットが含まれる場合はチェックしてください。チェックするとさらに次の入力項目が追加されます。

- 共有オブジェクトのキーと値のマッピングについて
- 共有オブジェクトはどのくらいの頻度で更新されるか
- setModeCallback を使用して編集状態と閲覧状態で異なるビューを提供するか。もしそうならモードの違いでどのような挙動の違いがあるか
- プレイバックモードでユーザーには何が見えるか
- ユーザーは他のユーザーがガジェットを操作している時に、そのことをどうやって理解するか
- ガジェットのサイズは固定か。違うのであれば、ガジェットのリサイズについて説明
- ガジェットはユーザーに操作説明を提供するか
- ガジェットは画像などの外部リソースを使用するか。使用するのであればそれぞれのリソースについてリソースを提供するサーバーとどのようなキャシュをするかについて簡単に説明

Does your extension use App Engine? 拡張が GAE を使用する場合はチェックしてください。チェックするとさらに次の入力項目が追加されます。

- どのような追加のウェブサービスと連携するか。それぞれのウェブサービスについてサービスに掛かる負荷を説明
- ロボット用の GAE アカウントは課金しているか
- ロボットをプロファイリングして CPU 使用量の面で最適化を試みたか
- GAE のデータストアを使用するのであれば、何を読み書きするのか、また memcache を使用するかを説明
- GAE で静的ファイルを保持する場合は、それらに使用しているキャッシュヘッダを説明
- GAE で動的にページを生成する場合は、描画のキャッシュに memcache をどう使っているか説明

チェック項目が非常に多くて少しうんざりするかもしれませんが、wave の拡張は小さ

なリクエストを数多く受け取ることになるため、拡張に人気が出た場合に下手な作りをしているとあっという間に負荷に耐えられなくなります。上記のチェックリストは Google が提供してくれたベストプラクティスと考え、ギャラリーに自作の拡張を公開するつもりがない人も一通り目を通しておくといいでしょう。

4.5 データ API

4.5.1 はじめに

データ API を使うと、ユーザーの代理として Wave プロバイダにアクセスし、wave を検索したり、wave の未読・既読などのプロパティを変更できます。データ API はロボットプロトコルを利用して実現されていて、Active API と少し似ていますが、Active API がロボットが参加している wave だけしか操作できないのに対し、データ API を使用するとユーザーが許す限りにおいて、そのユーザーの閲覧できる wave を全て操作することができます。

4.5.2 DataRequestHandler クラス

データ API を使うにはロボットライブラリの `waveservice` モジュールが利用できます。ただし、このモジュールを直接利用すると OAuth のトークンの管理[*49]を自分で実装しなければいけません。幸い Google Wave API チームが汎用的に使える `DataRequestHandler` クラスと、それを利用したデータ API のサンプルを公開してくれているので[*50]、本節ではそのサンプルを元に説明を進めます。なお、本来データ API は特定の言語やプラットフォームに縛られるものではありませんが、上記の都合上、本節での説明は Python のロボットライブラリを使って GAE 上で動かすためのものになります。

4.5.3 コンシューマ・キーとコンシューマ・シークレットの取得

データ API はユーザーの代理として動作するために OAuth プロトコルを使用します。デフォルトの匿名キーとシークレットを利用しても構いませんが、公開サービスを作るつもりであれば、ユーザーに不安を与えないためにも事前に Google にドメインを登録してコンシューマ・キーとコンシューマ・シークレットを取得しておいた方がいいでしょう[*51]。

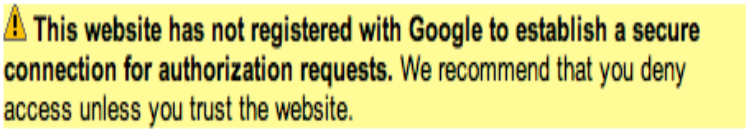
キーとシークレットを取得するには次のサイトにアクセスしてください。

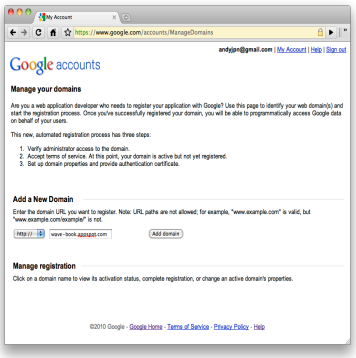
- https://www.google.com/accounts/ManageDomains

[*49] リクエストトークンを取得して、認証して、アクセストークンにアップグレードして・・・といった処理

[*50] http://wave-samples-gallery.appspot.com/about_app?app_id=266001

[*51] 登録していないドメインの OAuth 確認画面では次のような注意が表示されます。





データ API を利用するドメイン名を入力して「Add domain」ボタンをクリックします。

ドメインの所有権の確認画面が開きますので、好きな確認方法を選択して、準備ができたら「確認」ボタンをクリックします。確認方法はいくつかありますが「メタタグ」を選択するのが一番簡単でしょう。指定されたメタタグを含む index.html を作成して、ドメインのドキュメントルートに置いてください。

「確認」ボタンをクリックするとコンシューマ・キーとコンシューマ・シークレットが取得できます。なお、今回の例ではコンシューマ・キーがドメイン名になっていますが、これは Google がたまたまドメイン名をコンシューマ・キーとして使用しているだけで、OAuth プロバイダによっては不規則な英数字の並びになっていることもあります。

4.5.4　Wave 検索サービス

それでは OAuth の準備もできたところで、サンプルの作成に移りましょう。なお、先に述べたようにここで使用するサンプルは Google Wave API チームが作ったもの[*52]とほぼ同じですが、ソースコードを掲載する都合上、データ API とは直接関係しない、画面表示の部分をよりシンプルなものに変更にしています。

まずは以下のファイルをダウンロードして作業ディレクトリに保存してください。ここで登録されているクラス群が OAuth 関係の処理を担当します。

- http://google-wave-resources.googlecode.com/svn/trunk/samples/apps/inboxchecker/oauth

デフォルトでは匿名キーとシークレットに設定されているので、先ほど取得したコンシューマ・キーとコンシューマ・シークレットを使用する場合は WaveOAuthHandler クラスのコンストラクタを次のように変更します。匿名のままで構わないのであれば修正は不要です。

```
 1  class WaveOAuthHandler(webapp.RequestHandler):
 2    """OAuth を利用するリクエストハンドラの基底クラス"""
 3    COOKIE = 'wavesid'
 4
 5    def __init__(self):
 6      #self._service = waveservice.WaveService()
 7      # コンシューマ・キーとコンシューマ・シークレットを
 8      # 使用する場合は引数で指定
 9      self._service = waveservice.WaveService(
10        consumer_key='アプリケーション ID.appspot.com',
11        consumer_secret='abcdefghijklmnopqrstuvwx')
```

さらに、app.yaml に/oauth ディレクトリ以下へのアクセスは上記のファイルで処理するように指定します。

```
 1  application: アプリケーション ID
 2  version: 1
 3  runtime: python
 4  api_version: 1
 5
 6  handlers:
 7  - url: /oauth/.*
 8    script: oauth_handler.py
 9  - url: .*
10    script: アプリケーション ID.py
```

以上でデータ API を利用する準備は完了です。ではデータ API を使用して wave を検索するリクエストハンドラがどのようになるかを見てみましょう。

[*52] http://wave-samples-gallery.appspot.com/about_app?app_id=266001

```
class InboxHandler(oauth_handler.DataRequestHandler):
  """wave を検索するリクエストハンドラ"""

  def get(self):
    # OAuth で認可済みかどうかを確認
    # 認可済みでなければ認可画面にリダイレクト
    if not self.get_token():
      return

    # パラメータから検索条件を取得
    query = self.request.get('query', 'in:inbox')

    # WaveService インスタンスを使用して wave を検索
    search_results = self._service.search(query=query, num_results=10)

    # 結果を表示
    path = os.path.join(os.path.dirname(__file__), 'inbox.html')
    self.response.out.write(template.render(path, {
      'query': query,
      'digests': search_results.digests
    }))
```

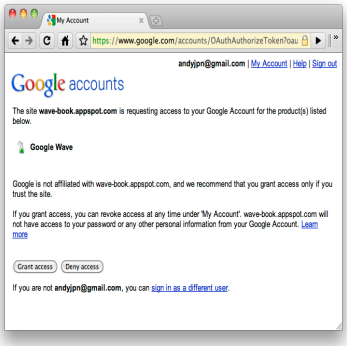
データ API を利用するには oauth_handler.DataRequestHandler クラスを継承したリクエストハンドラを定義します。DataRequestHandler クラスには OAuth を簡単に利用するための get_token メソッドが定義されていて、上記のようにして get/put メソッドの最初に get_token メソッドを呼び出しておけば、以降は_service プロパティ（waveservice.WaveService クラスのインスタンス）を通じてデータ API を使用できます。なお、WaveService#search メソッドの返り値は search.Results クラスのインスタンスです。詳細は 4.3.9.14 節を参照してください。

サンプルにはデータ API を使用するリクエストハンドラとしてもうひとつ FetchWaveHandler クラスが定義されていますが、InboxHandler クラスとほとんど同じなので省略します。ソースコードの全文は本節の最後にありますので参照してください。

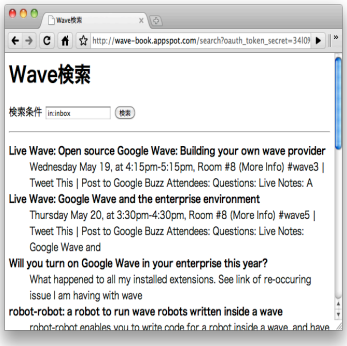
4.5.4.1 実行

アプリケーションが完成したら、デプロイするかローカル開発環境で実行して、ブラウザでアクセスしてください。

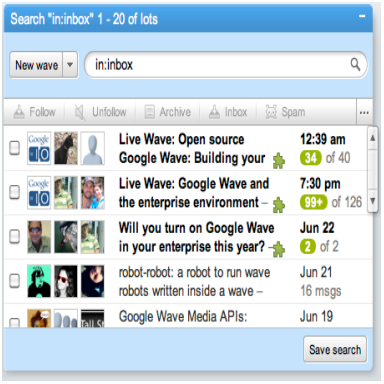
初回アクセスの時にだけ OAuth の認証画面にリダイレクトされます。アカウント選択画面が表示されたら、Wave のデータにアクセスするアカウントを選択します。



確認画面では「Grant access」をクリックしてアクセスを許可してください。



wave の検索ページが開くので検索条件を入力して「検索」をクリックすると検索結果が表示されます。



Google Wave を開いて内容を比較すると同じ結果であることが確認できるでしょう。

4.5.5 オペレーション

データ API はロボットプロトコルを利用しているので、ライブラリの内側では JSON-RPC が実行されています。データ API で利用できる JSON-RPC は現在のところ次の 3 つです。

メソッド	説明
wave.robot.fetchWave(waveId, waveletId)	本オペレーションは wavelet を取得します
wave.robot.search(query, index, numResults)	本オペレーションはユーザーの wave を検索します。`query` は検索条件で、`"in:inbox"`などの、Google Wave の検索パネルと同じ条件式が利用できます。`numResults` は検索結果の数で、デフォルトでは 10 件のデータが返されます
wave.robot.folderAction(modifyHow, waveId)	本オペレーションは `waveId` で指定された wave の状態を変更します。第一引数の `modifyHow` は次のような値を取ります: `markAsRead`, `markAsUnread`, `mute`, `active`（デフォルト）

表 4.6 データ API メソッド一覧

なお、`wave.robot.folderAction` メソッドは執筆時点ではまだロボットライブラリには実装されていないようです。そのため、このメソッドを利用するには、リクエストに使用する JSON を自分で組み立てる必要があります。

4.5.6 全ソースコード

本節で使用したサンプルのソースコード全文を掲載します。Google Wave API チームによる次のサンプルとほぼ同じです。

- http://wave-samples-gallery.appspot.com/about_app?app_id=266001

```
1  # -*- coding: utf-8 -*-
2  import wsgiref.handlers
3  import os
```

```
4
5  from google.appengine.ext import webapp
6  from google.appengine.ext.webapp import template
7
8  from waveapi import search
9  import oauth_handler
10
11 class InboxHandler(oauth_handler.DataRequestHandler):
12   """wave を検索するリクエストハンドラ"""
13
14   def get(self):
15     # OAuth で認可済みかどうかを確認
16     # 認可済みでなければ認可画面にリダイレクト
17     if not self.get_token():
18       return
19
20     # パラメータから検索条件を取得
21     query = self.request.get('query', 'in:inbox')
22
23     # WaveService インスタンスを使用して wave を検索
24     search_results = self._service.search(query=query, num_results=10)
25
26     # 結果を表示
27     path = os.path.join(os.path.dirname(__file__), 'inbox.html')
28     self.response.out.write(template.render(path, {
29       'query': query,
30       'digests': search_results.digests
31     }))
32
33   def post(self):
34     self.get()
35
36 class FetchWaveHandler(oauth_handler.DataRequestHandler):
37   """指定された ID を持つ wave を取得するリクエストハンドラ"""
38
39   def get(self):
40     # OAuth で認可済みかどうかを確認
41     # 認可済みでなければ認可画面にリダイレクト
42     if not self.get_token():
43       return
44
45     # パラメータから wave ID を取得
```

```
46      DIGEST_WAVE_ID = 'googlewave.com!w+SgYrEnoLE'
47      wave_id = self.request.get('wave_id', DIGEST_WAVE_ID)
48      wave_id = wave_id.replace('%2B', '+')
49
50      # WaveService インスタンスを使用して wave を取得
51      wavelet = self._service.fetch_wavelet(wave_id)
52
53      # 結果を表示
54      path = os.path.join(os.path.dirname(__file__), 'fetchwave.html')
55      self.response.out.write(template.render(path, {'wavelet':wavelet}))
56
57  def main():
58    application = webapp.WSGIApplication([
59      ('/inbox', InboxHandler),
60      ('/fetchwave', FetchWaveHandler),
61    ], debug=True)
62    wsgiref.handlers.CGIHandler().run(application)
63
64  if __name__ == '__main__':
65    main()
```

```
1  <html>
2  <head><title>Wave 検索</title></head>
3  <body>
4  <h1>Wave 検索</h1>
5  <form method="post" action="/inbox">
6    <label for="name">検索条件</label>
7    <input id="query" name="query" value="{{ query }}" />
8    <input type="submit" value="検索" />
9  </form>
10 <hr />
11 <dl>
12 {% for digest in digests %}
13   <dt><a href="/fetchwave?wave_id={{ digest.wave_id|urlencode }}">
14     {{ digest.title }}</a></dt>
15   <dd>{{ digest.snippet }}</dd>
16 {% endfor %}
17 </dl>
18 </body>
19 </html>
```

```
1  <html>
2  <head><title>Wave</title></head>
3  <body>
```

```
 4  <a href="/inbox">戻る</a>
 5  <h1>{{ wavelet.title }}</h1>
 6  <ul>
 7  {% for blip in wavelet.root_thread.blips %}
 8    <li>{{ blip.text }}</li>
 9  {% endfor %}
10  </ul>
11  </body>
12  </html>
 1  # -*- coding: utf-8 -*-
 2  import wsgiref.handlers
 3  import pickle
 4  import uuid
 5  import httplib
 6
 7  from google.appengine.ext import webapp
 8  from google.appengine.ext import db
 9
10  from waveapi import oauth
11  from waveapi import wavelet
12  from waveapi import waveservice
13
14  class OAuthRequestToken(db.Model):
15    """OAuth のリクエストトークンを保持"""
16    token_key = db.StringProperty(required=True)
17    token_secret = db.StringProperty()
18    created = db.DateTimeProperty(auto_now_add=True)
19    token_ser = db.TextProperty()
20
21  class OAuthAccessToken(db.Model):
22    """OAuth のアクセストークンを保持"""
23    token_ser = db.TextProperty(required=True)
24    token_key = db.StringProperty(required=True)
25    token_secret = db.StringProperty(required=True)
26    created = db.DateTimeProperty(auto_now_add=True)
27    session = db.StringProperty(required=True)
28
29  class WaveOAuthHandler(webapp.RequestHandler):
30    """OAuth を利用するリクエストハンドラの基底クラス"""
31    COOKIE = 'wavesid'
32
33    def __init__(self):
```

```
34      #self._service = waveservice.WaveService()
35      # コンシューマ・キーとコンシューマ・シークレットを
36      # 使用する場合は引数で指定
37      self._service = waveservice.WaveService(
38        consumer_key='アプリケーション ID.appspot.com',
39        consumer_secret='abcdefghijklmnopqrstuvwx')
40
41  class LoginHandler(WaveOAuthHandler):
42
43    def get(self):
44      # ステップ 1: コールバック URL を指定してリクエストトークンを取得
45      callback_url = "%s/oauth/verify" % self.request.host_url
46      request_token = self._service.fetch_request_token(
47        callback=callback_url)
48
49      # 認証ページから戻ってきたときに OAuthToken オブジェクトを
50      # 再生成するために永続化しておく
51      db_token = OAuthRequestToken(token_key=request_token.key,
52        token_ser=request_token.to_string())
53      db_token.put()
54
55      # ステップ 2: 認証 URL を生成してリダイレクト
56      auth_url = self._service.generate_authorization_url()
57      return self.redirect(auth_url)
58
59  class VerifyHandler(WaveOAuthHandler):
60
61    def get(self):
62      token_key = self.request.get('oauth_token')
63      verifier = self.request.get('oauth_verifier')
64      # 保存したリクエストトークンを取得
65      db_token = OAuthRequestToken.all().filter(
66        'token_key =', token_key).fetch(1)[0]
67      token = oauth.OAuthToken.from_string(db_token.token_ser)
68
69      # ステップ 3: 承認済みリクエストトークンをアクセストークンと交換
70      access_token = self._service.upgrade_to_access_token(
71        request_token=token, verifier=verifier)
72
73      # セッション ID と関連するアクセストークンをデータストアに保存
74      session_id = str(uuid.uuid1())
75      db_token = OAuthAccessToken(session=session_id,
```

```
76        token_key=access_token.key, token_secret=access_token.secret,
77        token_ser=access_token.to_string())
78      db_token.put()
79
80      self.response.headers.add_header('Set-Cookie', '%s=%s; path=/;'
%
81        (WaveOAuthHandler.COOKIE, session_id))
82      return self.redirect('/app/inbox')
83
84  class DataRequestHandler(WaveOAuthHandler):
85    """OAuth で保護されるリクエストハンドラの基底クラス"""
86
87    def get_token(self):
88      session_id = self.request.cookies.get(WaveOAuthHandler.COOKIE)
89      if not session_id:
90        return self.redirect('/oauth/login')
91      db_query = OAuthAccessToken.all().filter('session =', session_id)
92      db_token = db_query.get()
93      if db_token is None:
94        return self.redirect('/oauth/login')
95      self._service.set_access_token(db_token.token_ser)
96      return True
97
98  def main():
99    application = webapp.WSGIApplication([
100     ('/oauth/login', LoginHandler),
101     ('/oauth/verify', VerifyHandler),
102   ], debug=True)
103   wsgiref.handlers.CGIHandler().run(application)
104
105 if __name__ == '__main__':
106   main()
```

4.6 エンベッド API

4.6.1 はじめに

エンベッド API は Wave の持つ 4 種類の API の中で最小の API です。ドキュメントを見てみてもそこには `WavePanel` と `EmbedOptions`、たった 2 つの項目しかありません。しかも `WavePanel` クラスにはコンストラクタの他にたった一つしかメソッドがなく、`EmbedOptions` に至ってはクラスですらなく、単なる JavaScript のオブジェクトです。

しかし小さな API だからといってそれ以外の API と比べて重要性が劣るわけではありません。

例えば上の図では Google Wave とは何の関係もないブログに、エンベッド API によって wave が埋め込まれています。この埋め込まれた wave ももちろん Google Wave 上の wave と同じ機能を持っていて、Google Wave で編集した内容はブログ上の wave にもすぐ反映され、ブログから返信したり複数人でリアルタイムに編集することもできます。さらに、エンベッド API を使用すれば、Google Wave と外部サイトだけではなく、Wave とは無関係なサイト同士が Wave を介してリアルタイムにコラボレーションすることさえ可能になります。

また、wave を誰にでも見られるようにしようと public にしても、Google Wave のアカウントがない人はそもそも Google Wave に入ることができないため、その wave を見ることはできません。しかし、エンベッド API を使用して外部サイトに貼り付けることで Google Wave のアカウントが無い人にも public な wave の内容が閲覧可能になります。これはエンベッド API の大きな特徴です。

エンベッド API の目的が他のサイトに気軽に組み込んで使うことだとすると、シンプルさは短所ではなくむしろ長所といえるでしょう。wave 外部のさまざまなサイトに組み込むと普段先進的なサービスを自分から使うことのないユーザーにも wave を使ってもらうことができます。もしかすると Wave の普及に最も重要な役目を果たすのはエンベッド API なのかもしれません。

4.6.2 ウェブサイトに wave を組み込む

それではさっそくエンベッド API のコード例を見てみましょう。

```
 1  <html>
 2    <head>
 3      <meta http-equiv="Content-Type"
 4        content="text/html; charset=UTF-8">
 5      <title>wave の組み込み</title>
 6      <script src="http://www.google.com/jsapi"></script>
 7      <script type="text/javascript">
 8      google.load('wave', '1');
 9
10      function initialize() {
11        var embedOptions = {
12          target: document.getElementById('waveframe')
```

```
13          };
14          var wavePanel = new google.wave.WavePanel(embedOptions);
15          wavePanel.loadWave('googlewave.com!w+OAfwpOUQA');
16        }
17
18        google.setOnLoadCallback(initialize);
19        </script>
20      </head>
21      <body>
22        この線の下に wave が表示されます。
23        <hr />
24        <div id="waveframe" style="width:500px; height:300px"></div>
25      </body>
26    </html>
```

HTML のロードが終わると intialize 関数が実行され、waveframe という ID を持つ DIV 内に googlewave.com!w+OAfwpOUQA という ID を持つ wave が表示されます。Google AJAX API ローダーを使っている部分を除けば、WavePanel インスタンスを生成して loadWave 関数を呼び出す実質 2 行だけです。ブラウザで表示すると次のようになります。

それではソースコードを順に見ていきましょう。

4.6.2.1　Google AJAX API ローダー

エンベッド API は Google AJAX API ローダーを使用してロードします。Goole AJAX API は次のようなサーバーサイドの処理を伴わない Google の API の総称です。

- Google MAPS API
- Google AJAX Search API
- Google AJAX Feed API
- Google AJAX Language API
- Google Data API
- Google Earth API
- Google Visualization API

- Google Friend Connect API
- Orkut API

AJAX API を使う手順はいずれも同じで、次のようになります。

1. AJAX API ライブラリをロードします。API によっては API キーをパラメータとして渡さなければいけませんが、エンベッド API の場合は API キーは不要です。

```
1  <script type="text/javascript" src="http://www.google.com/jsapi"></script>
```

2. 使用する Google API を google.load 関数（AJAX API ローダー）でロードします。google.load 関数は第一引数に API 名、第二引数に API のバージョンを取り、エンベッド API の場合は第一引数が"wave"、バージョンが"1"になります。

```
1  <script type="text/javascript">
2    google.load('wave', '1');
```

3. ライブラリロード後に初期化が必要であれば、初期化関数（ここでは initialize）を定義します。（詳細は次項）
4. 初期化関数をコールバックに設定します。コールバックの設定には google.setOnLoadCallback 関数を使用します。ライブラリを google.load 関数で読み込んでいるため、通常の window.onLoad などを使用すると期待したタイミングで初期化関数が実行されません。注意してください。

```
1  google.setOnLoadCallback(initialize);
```

4.6.2.2 エンベッド API

それではここからエンベッド API を使用している initialize 関数の中を見ていきます。

```
1  function initialize() {
2    var embedOptions = {
3      target: document.getElementById('waveframe')
4    };
5    var wavePanel = new google.wave.WavePanel(embedOptions);
6    wavePanel.loadWave('googlewave.com!w+OAfwpOUQA');
7  }
```

embedOptions は初期設定を保持するためのオブジェクトです。wave を組み込む DOM 要素を指定する target プロパティだけが必須項目になっています。その他のプロパティに関しては次項で説明します。

エンベッド API を使用しているのはその後の 2 行で、設定オブジェクトを渡して google.wave.WavePanel クラスをインスタンス化し、wave ID を引数にそのインスタンスの loadWave メソッドを呼び出しています。

デフォルトでは WavePanel は通常の Google Wave から wave を取得します。もしサンドボックスの wave を読み込みたければ、WavePanel インスタンス作成時に rootUrl

オプションでサーバーインスタンスを指定してください。指定できる Wave サーバーインスタンスは、現在のところは次のいずれかです。なおインスタンスの**最後の'/' は省略はできません。**注意してください。

サーバー	Wave サーバーインスタンス
Wave プレビュー	http://wave.google.com/wave/
Wave サンドボックス	http://wave.google.com/a/wavesandbox.com/

表 4.7　Wave サーバーインスタンス

4.6.2.3　wave ID の取得

`loadWave` 関数に渡す wave ID は、サンドボックスでは Debug メニューで「Get current wave ID」というアイテムをクリックすると小さなダイアログが開いて簡単に確認できます。プレビューでは Debug メニューは利用できないので、URL から自分で wave ID を抜き出してください。

https://wave.google.com/wave/#restored:search:with%253Apublic+api,
restored:wave:googlewave.com!w%252BaZqycFPkQ

例えば、ある wave を表示している時の URL が上記のようになっていたとすると、「`restored:wave:`」に続く部分、「`googlewave.com!w%252BaZqycFPkQ`」がその wave の URL エンコードされた wave ID です。デコードするには「`%252B`」を「`+`」と置き換えてください。つまり、この例では「`googlewave.com!w+aZqycFPkQ`」が目的の wave ID ということになります。

4.6.3　WavePanel をカスタマイズする

`WavePanel` はコンストラクタのオプションでさまざまにカスタマイズできます。例えば、参加者リストとツールバーが表示されている、サンドボックスの wave を貼り付けるには次のようにオプションを指定します。

```
1  var embedOptions = {
2    target: document.getElementById('waveframe'),
3    rootUrl: 'http://wave.google.com/a/wavesandbox.com/',
4    footer: true,
5    header: true,
6    toolbar: true
7  };
8  var wavePanel = new google.wave.WavePanel(embedOptions);
9  wavePanel.loadWave('wavesandbox.com!w+cblNe4j0A');
```

このようにオプションを指定した `WavePanel` の表示は次のとおりです。

WavePanel に設定できるオプションの一覧は表 4.8 を参照してください。

プロパティ	型	説明
bgcolor	文字列	wave パネルの CSS（background-color）の値
footer	真偽値	wave パネルにフッターを表示するかどうか
header	真偽値	wave パネルにヘッダーを表示するかどうか
height	数値	wave パネルの高さ。なるべくこの属性は指定せず、コンテナとして使用する div 要素にサイズ指定を記述してください
rootUrl	文字列	Wave サーバー URL。値については表 4.7 参照
target	文字列	wave パネルのコンテナになる div 要素の ID
toolber	真偽値	wave パネルにツールバーを表示するかどうか。このフラグは header の値が true の場合にだけ有効です
width	数値	wave パネルの幅。なるべくこの属性は指定せず、コンテナとして使用する div 要素にサイズ指定を記述指定してください

表 4.8　WavePanel 初期化オプション

4.6.4　wave を組み込むためのコードを生成する

4.6.4.1　Link to wave...

ここまで、エンベッド API を説明してきましたが、実は、プログラマ以外のユーザーでも簡単に wave をウェブサイトに貼り付けることができるように、Google Wave には上記で説明したようなコードを生成する機能が組み込まれています。wave のツールバーにある「Link to wave...」ボタンをクリックしましょう。

「Paste HTML to embed in website:」というラベルの下に表示されているのが wave を貼り付けるためのコードです。改行とインデントを追加すると次のようになっています。

```
<div id="waveframe" style="width:500px; height:400px;"></div>
<script src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
google.load("wave", "1");
google.setOnLoadCallback(function() {
  new google.wave.WavePanel({
    target: document.getElementById("waveframe")
  }).loadWave("googlewave.com!w+OAfwpOUQA");
});
</script>
```

コンテナになる DIV タグの id が `waveframe` に固定されているので、一つの HTML 内に複数の wave を貼り付けようとすると、このままコピー&ペーストしたのでは問題がおきるかもしれませんが、そうでなければ非常に簡単に使うことができます。また、自分でオプションを設定した wave を貼りつけたい場合にも、このダイアログからコピーしたものを雛形として使用すると便利でしょう。

4.6.4.2　Google Web Elements

Google Web Elements[*53]は様々な Google サービスをウェブサイトに埋め込むことを支援するサービスで、このサービスを利用して wave を埋め込むためのコードを簡単に生成することができます。次のサイトにアクセスしてください。

- http://www.google.com/webelements/wave/

Wave URL、サイズ、ヘッダー・フッターの表示を指定してエンベッド API を利用するコードを生成できます。

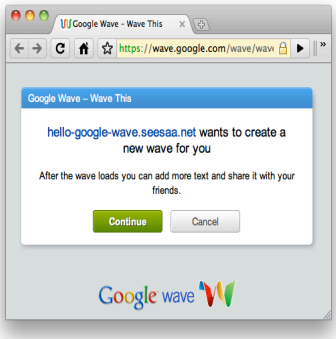
[*53] http://www.google.com/webelements/

4.7 WaveThis ボタン

4.7.1 はじめに

WaveThis はあるウェブサイトに関する議論を Google Wave で始めるためのボタンです。ニュースサイトやブログなどで twit ボタンやソーシャルブックマークボタンを見ることがあるでしょう。その Google Wave 版だと考えてください。API と呼べるものではありませんが、ここで紹介しておきます。

ブログ記事の下に適切はパラメータと共にボタンを用意しておくと、そのボタンをクリックするだけで、Google Wave 上で元記事へのリンクを含む新しい wave が作成されます。試しに上記のサイトの WaveThis ボタンをクリックしてみましょう。



確認画面が開くので、「Continue」を選択します。

すると、このように元記事へのリンクを含む新規 wave が開きました[*54]。参加者を追加してウェブサイトを紹介するのに使用する他、単に自分用のメモとして使用してもいいでしょう。

4.7.2　リンクでの WaveThis

WaveThis ボタンがどのようにして作られているのか、先ほどのボタンのソースを見てみましょう。

```
1  <a href="https://wave.google.com/wave/wavethis?u=http%3A%2F%2Fhello-google-wa
target="_blank">
2    <img src="http://wave.google.com/wavethis/icon16.png" alt="Wave
This" title="Wave This" border="0" hspace="1">
3  </a>
```

ここから分かるように、実は WaveThis とはいくつかの URL パラメータを伴った次のサイトへのリンクです。

- https://wave.google.com/wave/wavethis

4.7.2.1　パラメータ

WaveThis に使用できるパラメータには次のようなものがあります。

t（別名:title）　新規 wave のタイトルです。省略すると URL がタイトルとして使用されます

u（別名:url）　新規 wave に記述されるリンク URL です。省略するとリクエストのリファラが使用されます

c（別名:message）　新規 wave 本文に記述されるテキストです。省略すると u で渡した URL が使用されます

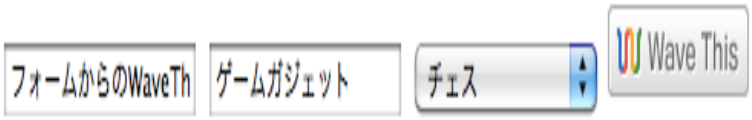
g　新規 wave に貼り付けるガジェットの URL です。ガジェットには自動的に `wavethis_referer` というキーを持つ共有状態が設定され、その値に応じて表示を変えることができます

[*54] wave の内容はボタンのパラメータで指定されます

4.7.3 フォームでの WaveThis

WaveThis のパラメータは POST メソッドを使用して送信することもできます。パラメータをユーザーに入力させる必要がある場合にはこちらの方がいいでしょう。例えば、ユーザーが選択したガジェットを含む wave を作成する WaveThis ボタンは次のようになります。

```
<form name="wavethis" method="post"
  action="https://wave.google.com/wave/wavethis" target="_blank">
  <input type="text" name="t" value="フォームからの WaveThis"/>
  <input type="text" name="c" value="ゲームガジェット"/>
  <select name="g">
    <option value="http://wave-chess.appspot.com/wavechess/com.google.wave.chess.client.ChessGadget.
チェス</option>
    <option value="http://apps.technohippy.net/animal-shogi/gadget.xml">
どうつぶしょうぎ</option>
    <option value="http://wave-cards.googlecode.com/svn/trunk/cards.xml">
トランプ</option>
  </select>
  <img src="http://wave.google.com/wavethis/button24.png"
    alt="WaveThis" onclick="javascript:document.wavethis.submit()">
</form>
```



ガジェットとして「チェス」を選択してクリックした結果は次のようになります。

4.7.4 アイコン

ここまでの例でも利用しましたが、WaveThis リンクに使用するための画像を Google 社が提供しています。使用する場合は画像を別のサーバーにコピーするのではなく、画像タグの src の値として下記の URL を直接使用してください。

高さ	アイコン	URL
16	 	http://wave.google.com/wavethis/icon16.png http://wave.google.com/wavethis/button16.png
24	 	http://wave.google.com/wavethis/icon24.png http://wave.google.com/wavethis/button24.png
32	 	http://wave.google.com/wavethis/icon32.png http://wave.google.com/wavethis/button32.png

表 4.9　WaveThis アイコン

第 5 章

プロトコル

5.1 はじめに

本章では wave を構成する 3 つの P の最後の一つ、プロトコルについて説明します。

まず「wave がプロトコルである」とはどういう意味でしょうか？ここでの「プロトコル」とはコンピュータ同士がネットワークを介して情報をやり取りする手順をさす用語です。始めにプロトコルを定義して、それに沿ったやりとりをすることで、通信相手のことをなにも知らなくてもお互いに通信することができるるようになります。

例えば、一般に最もよく目にするプロトコルの例として、主に Web サイトの閲覧に利用される HTTP（Hyper Text Transfer Protocol）があります。これも HTTP として定義された手順でやりとりすることで、クライアントがどのような OS・ブラウザを使っていたとしても、そしてサーバーがどのような OS・Web サーバーを使用したとしても、お互いがお互いを全く気にせずにクライアントはサーバーからファイルを取得することが可能になります。

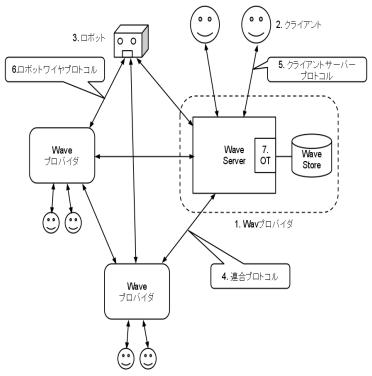
つまり「wave がプロトコルである」とは、「wave」という言葉が Google に閉じた一つの実装を指すものではなく、公開された仕様を指すものだという意味です。Web サーバーに Apache HTTP サーバーや IIS・nginx など様々な実装があるように、将来的には wave サーバーも Google Wave サーバー以外に Apache Wave サーバー・MS Wave サーバーなどの実装が現れるかもしれません。Google 以外の会社や個人が作ったものであっても、wave プロトコルを正しく実装したものであればそれを wave サーバーと呼んで構わないのです。

また、wave プロトコルの全てを実装して wave サーバーを作るばかりではなく、2.5 節で紹介した例のようにプロプライエタリなシステムに wave プロトコルの一部を実装して、外部の wave サーバーとやりとりをすることも可能です。

Google は実際にそういった独自の wave を公開・実装しようとする人たちのために、さまざまな技術資料や実際に動作可能な wave サーバーのプロトタイプ実装を公開しています。本章ではそれらの公開資料に基づいて、wave の全体的なシステム構成、wave で使用されている特徴的な技術、プロトタイプ実装の使用方法などについて駆け足で説明します。wave に関わる仕様は膨大であることに加え、著者の力量不足もあり、網羅的な説明にはなりませんでしたが、少なくともみなさんが独自に調査を進める際の足がかりになればと思います。

5.2　システム概観

では、初めに本章の対象になるシステムの全体像を眺めてみましょう。



wave は E メールシステムと同様に複数の wave プロバイダが協働して動作し、ユーザーがどのプロバイダに属していてもお互いにやりとりすることが可能です[*1]。上図はそういった様子を示しています。以下で図の中で番号の振られているそれぞれの要素について簡単に説明します。

wave プロバイダ (1)　wave システムの中心になるのはもちろん wave プロバイダです。wave プロバイダはその名の通り wave を提供（provide）するサーバーで、その役割は大きく 2 つあります。一つ目は自ドメインに所属する wave に対する同時多数からの編集を矛盾を起こさないように処理すること、二つ目はその wave を他の wave プロバイダと協力してクライアントの元まで届けることです。本章では主にこの wave プロバイダの動作に付いて説明しています。

クライアント (2)　ここでのクライアントは wave のユーザー自身と、そのユーザーが wave にアクセスするために利用しているソフトウェアを含むものと考えてください。wave のユーザーは特定の wave プロバイダに所属していて、 [ユーザー ID]@[所属する wave プロバイダのドメイン] という E メールと同じ形式の一意な wave アドレスで識別されます。ユーザーが直接やり取りするのは自分が所属する wave プロバイダだけで、他のドメインに属する wave が必要な場合にも自分が所属している wave プロバイダを経由して取得します。なお、ユーザーが所属する wave プロバイダはユーザー自身の情報に加え、そのユーザーが作成した wave の管理も担当することになります。

ロボット (3)　ロボットは wave プロバイダにアクセスするもうひとつの存在です。クライアントとは違いロボットは特定の wave プロバイダに所属しておらず、wave の情報が必要なときには、その wave を管理している wave プロバイダに直接アクセスします。

プロトコル　wave プロバイダが通信する相手は wave プロバイダとクライアントとロボットの三種類あります。そのためプロトコルも通信相手ごとに三種類あり、それ

[*1] ただし現在は wave プロバイダは Google だけです。また Google が提供する 2 つの wave プロバイダ、wave プレビューと wave サンドボックスはお互いに接続されてはいません

ぞれ連合（Federation）プロトコル、クライアントサーバープロトコル、ロボットワイヤプロトコルと呼ばれています。

連合プロトコル (4) wave プロバイダが管理する wave は、自ドメインに属するユーザーが作成した wave だけです。それ以外のユーザーが作成した wave を自分のクライアントから要求されたときには、その wave を管理している wave プロバイダに要求して wave のコピーを手に入れる必要があります。さらに、その wave がクライアントによって編集されたときにも編集内容を適切な wave プロバイダに届けなければいけません。もちろん自分が管理する wave を他の wave プロバイダに属するユーザーが閲覧・編集した場合には、反対にそのユーザーが所属する wave プロバイダに wave のコピーを送り、ユーザーの操作が届いた場合にはその操作を wave に適用する必要があります。このような wave に関する情報を wave プロバイダ間でやり取りするためのプロトコルが連合プロトコルです

クライアントサーバープロトコル (5) wave プロバイダとクライアントが通信する際に使用されるプロトコルです。公式のオープンソースクライアントはまだ公開が始まったばかりで、執筆時点ではドキュメントもまだ書きかけといっていい状態です。wave サーバーとブラウザとの通信を個人的に解析した情報を公開しているウェブサイトもありますが、将来的に大きく変更される可能性もあるため、本書ではあまり深くは触れません

ロボットワイヤプロトコル (6) wave プロバイダとクライアントが通信する際に使用されるプロトコルです。ロボットサーバーが wave サーバーから受け取るイベント情報を含む JSON メッセージと、ロボットサーバーが wave を操作するために wave サーバーへ送る JSON-RPC が定義されています。

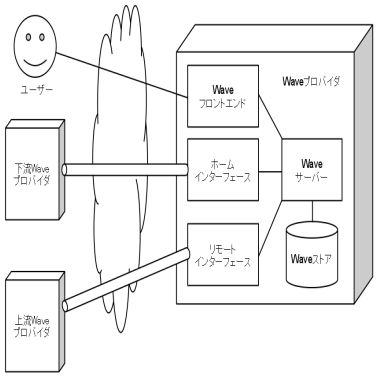
オペレーション変換 (7) wave プロバイダの重要な役割の一つが同時多数からの wave に対する変更を矛盾なく処理することです。この wave の神髄とも言える機能を実現しているのがオペレーション変換（Operation Transformation: OT）と呼ばれるアルゴリズムで、wave を理解するにはこのアルゴリズムの理解が欠かせません。簡単に説明すると wave ではドキュメントは適用された操作の配列として管理していて、操作の適用時に他のユーザーの変更と衝突が発生しそうなときには操作を変換して矛盾を解消してから元のドキュメントに適用することで、衝突を回避します。詳細については 6.2 節を参照してください

5.3 wave システムアーキテクチャ

wave の個々の要素やプロトコルの説明に入る前に、wave プロバイダがシステム全体の中でどのように動作しているのかを確認しておきましょう。

5.3.1 システム構成

まずは静的な構造です。次の図を見てください。



wave プロバイダは担当する機能ごとに 5 つのコンポーネントに分けて考えることができます。

wave サーバー　wave プロバイダの全てのコンポーネントを統合して wave サービスを提供します。wave の編集の競合を解消するためのオペレーション変換もここで行われます

wave ストア　永続化された wavelet[*2]を保持します。保持されるのは、自ドメインに所属するユーザーが作成した wavelet と、自ドメインに所属するユーザーが参照した他ドメイン所属の wavelet のコピーで、前者はローカル wavelet、後者はリモート wavelet と呼ばれます。なお、使用されるストレージについての規定は特になく、Google Wave では Google 製 KVS である BigTable が使用されているでしょうし、プロトタイプ実装である FedOne ではオンメモリで保持されています[*3]

連合ホスト　ローカル wavelet に対する情報を他の wave プロバイダとやり取りをするためのコンポーネントです。具体的には

- 要求に応じてローカル wavelet の内容を、その他の wave プロバイダに送信する
- ローカル wavelet の変更を、その wavelet の参加者が所属する wave プロバイダに通知する
- ローカル wavelet を他の wave プロバイダに所属する参加者が変更した場合に、その変更内容を受け取り適切に反映する

という 3 つの処理を担当します。連合プロトコルはこの連合ホストと次に紹介する連合リモートとの間で使用されるプロトコルです

連合リモート　リモート wavelet に対する情報を、実際にその wavelet を管理している wave プロバイダとやり取りをするためのコンポーネントです。wavelet を管理している wave プロバイダの連合ホストとやり取りをすることになるため、具体的な役割としては前項と対応する

- 他の wave プロバイダに属する wavelet をクライアントから要求され、そのコピーを保持していないときに、適切な wave プロバイダに最新の wavelet の内容を要求し、受け取る

[*2] wavelet のスナップショットを保持しているわけではなく、wavelet へのオペレーションの集合を保持しています

[*3] FedOne を、MongoDB を使うように拡張した人もいます

- 他の wave プロバイダに属する wavelet に自身のクライアントが参加しているとき、その wavelet を管理する wave プロバイダから他の参加者がなした変更を受け取る
- 他の wave プロバイダに属する wavelet に自身のクライアントが参加していて、クライアントが wavelet に変更を加えたとき、その wavelet を管理する wave プロバイダに変更を送信する

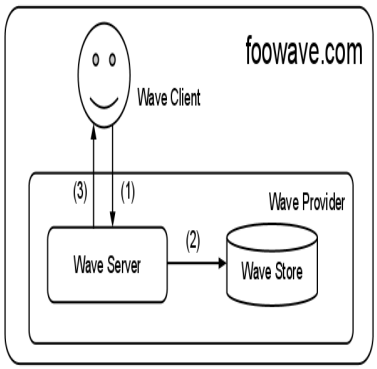
という 3 つの処理を担当します。連合プロトコルはこの連合リモートと先に紹介した連合ホストとの間で使用されるプロトコルです

クライアントフロントエンド 自分のドメインに属するユーザーとのやり取りを担当します。クライアントサーバープロトコルは、このコンポーネントとクライアントの間で使用されるプロトコルです

5.3.2 データフロー

wave プロバイダの静的な構造は前項で把握できたことと思います。ここではその構造の上をデータがどのように流れるかを眺めてみましょう。ユーザーは常に自分が所属する wave プロバイダと通信しますが、前節のホーム／連合リモートの項から分かるように、ユーザーの所属する wave プロバイダと操作対象の wavelet の所属する wave プロバイダが同じか否かで、データの流れが大きく異なります。それぞれの場合について、wavelet の取得と更新がどのようになるかみてみることにしましょう。なお、ここでの説明は基本的に連合プロトコルの実験用実装である FedOne の動作に基づいています。FedOne と実際の Google Wave サーバーの違いについては 5.5.1 節を参照してください。

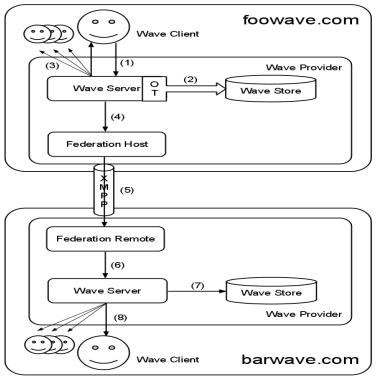
5.3.2.1 クライアントと所属が同じ wavelet を取得



ローカル wavelet の取得については特に変わったことはありません。

1. クライアントが wave プロバイダに wavlet を要求します
2. wave サーバーは wave ストアから要求のあった wavelet を取り出します
3. クライアントに wavelet を返信します

5.3.2.2　クライアントと所属が同じ wavelet を編集



ローカル wavelet の変更は次のようになります。

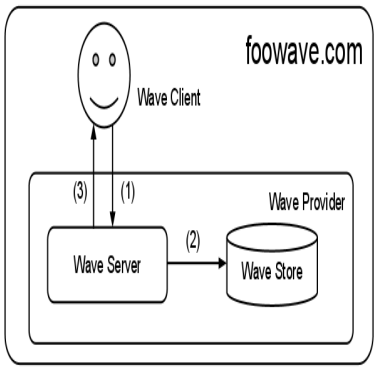
1. クライアントが変更内容（オペレーション）を wave プロバイダに送信します
2. wave サーバーはクライアントから送られてきた変更内容の元バージョンと、現在 wave ストアに保存されている wavelet のバージョンを比較します。両者が違っていたら読み出し後に他の参加者によって編集され、変更内容が衝突しているということなので、オペレーション変換（OT:6.2 節参照）を実行し、衝突を解消してから wave ストアに保存します
3. 変更内容を自ドメインに所属する参加者に送信します。衝突が発生していた場合は、最初にクライアントから受け取った変更内容と、ここで送り返す内容が異なっていることに注意してください
4. wavelet の参加者が全員 foowave.com ドメイン所属の場合は先の項で完了ですが、他のドメインに所属する参加者がいると、さらにそのドメインの wave プロバイダにも変更を通知しなければいけません。図では barwave.com に所属する参加者に対して通知する例になっていますが、この処理は wavelet の参加者が存在する全ての wave プロバイダに対して行われます。
5. wave サーバーはローカル wavelet への変更を連合ホスト経由で他の wave プロバイダに通知します。wave プロバイダ間で使用される連合プロトコルは XMPP[*4]の拡張です。通知先の wave プロバイダにとってはリモート wavelet に関する通知なので、連合リモート経由で情報を送ります
6. wave サーバーが連合リモート経由で変更を受け取ります
7. wave ストアに最新のリモート wavelet を保存します。リモート wavelet は本体を所有している wave プロバイダによってのみ変更されます。そのため衝突を考慮する必要はなく、単純に発生順にオペレーションが適用されます
8. 最新の wavelet の内容を自ドメインに所属する参加者に送信します

[*4] Extensible Messaging and Presence Protocol。Jabber というインスタントメッセージングサービス用に開発されたプロトコルがオープンになったものです

5.3.2.3 クライアントと所属が違う wavelet を取得

リモート wavelet の取得は、wavelet のコピーがクライアントの所属する wave プロバイダに存在するかどうかで流れが異なります。

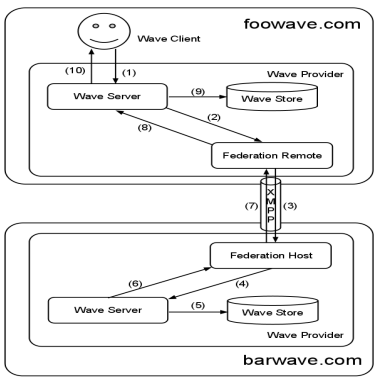
リモート wavelet のコピーが存在する



事前に対象 wavelet にアクセスしたことがあり、wavelet のコピーが wave プロバイダに存在した場合はリモート wavelet の取得はローカル wavelet を取得する場合と違いはありません。

1. クライアントが wave プロバイダに wavlet を要求します
2. wave サーバーは wave ストアから要求のあったリモート wavelet を取り出します
3. クライアントに wavelet を返信します

リモート wavelet のコピーが存在しない



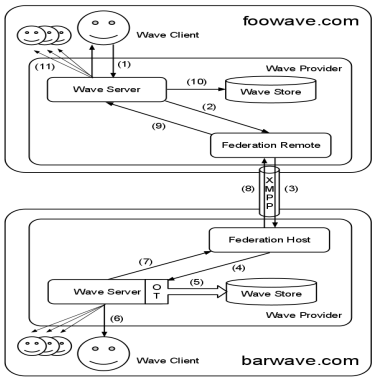
例えば新しく wavelet の参加者に追加されたときなどは、リモート wavelet のコピーがユーザーの所属する wave プロバイダに存在しないことがありえます。その場合は wavelet が所属する wave プロバイダから取ってこなければいけません。

1. クライアントが wave プロバイダに wavlet を要求します
2. wave サーバーは wave ストアに要求のあったリモート wavelet が存在しないので、

連合リモート経由で wavelet が所属している wave プロバイダに要求を投げます

3. リモート wavelet はそれを管理する wave プロバイダに取ってはローカル wavelet なので、連合リモートの通信相手は連合ホストになります。wave プロバイダ間の通信には XMPP が使用されます
4. 連合ホスト経由で wave サーバーが要求を受け取ります
5. 要求された wavelet を wave ストアから取得します
6. 自分の管理する wavelet の送信なので連合ホスト経由で返します
7. 要求した wave プロバイダは連合リモートで wavelet を受け取って wave サーバーに渡します
8. 取得した wavelet は次回以降の問い合わせに備えて wave ストアに保持します
9. クライアントに wavelet を返信します

5.3.2.4　クライアントと所属が違う wavelet を編集

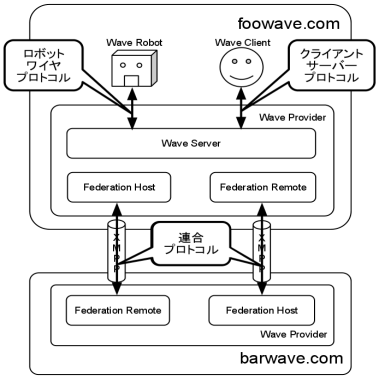


リモート wavelet の編集は、これまでにみたリモート wavelet の表示とローカル wavelet の編集を組み合わせたような流れになります。

1. クライアントが変更内容（オペレーション）を wave プロバイダに送信します
2. wave サーバーは連合リモート経由で wavelet が所属している wave プロバイダに変更内容を送信します
3. wavelet が所属する wave プロバイダの連合ホストに変更内容を送信します
4. 連合ホスト経由で wave サーバーが変更内容を受け取ります
5. wave サーバーは送られてきた変更内容の元バージョンと、現在 wave ストアに保存されている wavelet のバージョンを比較します。両者が違っていたら変更内容が衝突しているので、オペレーション変換（OT:6.2 節参照）を実行し、衝突を解消してから wave ストアに保存します
6. 変更内容を自ドメインに所属する参加者に送信します
7. wavelet 参加者が所属する全ての wave プロバイダに変更を通知します。図では wave を編集したユーザーが所属する foowave.com だけに変更内容を返信しているようですが、実際にはそれ以外の wave プロバイダにも変更を通知しています
8. wave サーバーはローカル wavelet への変更を連合ホスト経由で他の wave プロバイダに通知します
9. wave サーバーが連合リモート経由で変更を受け取ります

10. wave ストアに最新のリモート wavelet を保存します
11. 最新の wavelet の内容を自ドメインに所属する参加者に送信します

5.4 プロトコル



本章の最初に紹介しましたが、wave プロバイダが使用するプロトコルはその通信相手によって 3 種類あります。

プロトコル	通信相手
連合プロトコル	他の wave プロバイダ
クライアントサーバープロトコル	クライアント
ロボットワイヤプロトコル	ロボット

表 5.1 Wave プロトコル一覧

ここまでで全体的な流れは理解できたと思うので、本項ではそれぞれのプロトコルで実際にどういったメッセージがやり取りされているのかを眺めてみましょう。ただしここではいくつかの実例を簡単に見てみるだけで、メッセージ内容の詳細にはあまり触れません。詳細に関しては公式ドキュメント[*5][*6][*7]を参照してください。

5.4.1 連合プロトコル

連合プロトコルは XEP-114[*8]という仕様に基づいた XMPP の拡張として実装されています。XMPP のメッセージは XML 形式で、セッションのルート要素は`<stream>`要素ですが、クライアントとサーバーの一連のやりとりが終了するまでは`<stream>`要素は閉じられることがありません。その`<stream>`要素の直接の子要素、つまりセッション上を流れる一つ一つのメッセージのことを**スタンザ（stanza）**と呼びます。

連合プロトコルが使用するスタンザの大きく分けて次の二種類です。

[*5] 連合プロトコル:http://www.waveprotocol.org/draft-protocol-specs/draft-protocol-spec
[*6] クライアントサーバープロトコル:http://www.waveprotocol.org/whitepapers/internal-client-server-protocol
[*7] ロボットワイヤプロトコル:http://code.google.com/intl/ja/apis/wave/extensions/robots/protocol.html
[*8] http://xmpp.org/extensions/xep-0114.html

- アップデートスタンザ
- サービススタンザ

全てのスタンザは共通の属性として wavelet-name、つまり wave ID と wavelet ID を組み合わせたものを持ち、どの wavelet に関係しているかを判断できます。

5.4.1.1 アップデートスタンザ

アップデートスタンザはその名の通りローカル wavelet の更新を通知する際に使用されるスタンザです。従ってホスト（対象 wavelet を管理している wave プロバイダ）から開始されて、リモート（対象 wavelet のコピーを持つ wave プロバイダ）へ送られることになります。アップデートスタンザに属するスタンザは次のひとつだけです。

- <wavelet-update/>

アップデートスタンザは Message スタンザ[*9]を使用して送信され、実際には次のような形式になります[*10]。

ホストからリモートへの更新通知

```
<message type="normal"
  from="wave.initech-corp.com"
  id="1-1" to="wave.acmewave.com">
  <request xmlns="urn:xmpp:receipts"/>
  <event xmlns="http://jabber.org/protocol/pubsub#event">
    <items>
      <item>
        <wavelet-update
          xmlns="http://waveprotocol.org/protocol/0.2/waveserver"
          wavelet-name="acmewave.com/initech-corp.com!a/b">
          <applied-delta><![CDATA[CiI...MwE]]></applied-delta>
        </wavelet-update>
      </item>
    </items>
  </event>
</message>
```

リモートからホストへの ACK

```
<message id="1-1"
  from="wave.acmewave.com"
  to="wave.initech-corp.com">
  <received
    xmlns="urn:xmpp:receipts"/>
</message>
```

[*9] プッシュで情報を通知するためのスタンザ
[*10] 公式ドキュメントより引用

上記例を見ると、更新通知という割には更新内容がどこにも含まれていないと思うかもしれません。実は連合プロトコルは XMPP の上に乗せるために XML の皮を被っていますが、その中身はプロトコルバッファ（6.1 節参照）というバイナリフォーマットです。更新通知の<applied-delta>タグの本文が省略されていますが、そこに更新内容を持つプロトコルバッファを BASE64 エンコーディングしたものが格納されています。

5.4.1.2 サービススタンザ

サービススタンザはアップデートスタンザ**以外**の全てのスタンザを指し、オペレーションのサブミットや、ヒストリの要求に使用されます。サービススタンザの一覧は次のとおりです。

<submit-request/> オペレーションをサブミットします。全てのスタンザでオペレーションは BASE64 エンコードしたプロトコルバッファ（6.1 節参照）で表されています

<submit-responose/> <submit-request/>の返信で、適用されたオペレーションや、タイムスタンプ、バージョン番号などを返します

<history-request/> バージョンの範囲を指定して、オペレーションを要求します

<history-response/> <history-request/>の返信で、ゼロ個以上のオペレーションを返します

<signer-get-request/> 署名者が不明な wavelet のための証明書を要求します

<signer-get-response/> <signer-get-reqeust/>の返信で、BASE64 エンコードされた証明書のチェーンを返します

<signer-post-request/> 初めて wavelet の編集内容をサブミットする前に、証明書のチェーンをその wavelet をホストしている wave サーバーに送ります

<signer-post-response/> <signer-post-request/>の ACK です

サービススタンザは全てリモートから開始され、子要素はそれぞれ異なりますが、いずれも XEP-0060 に基づいた PubSub イベントを使用して送信されます。一例として submit-request、submit-response を挙げると、次のような形式です[*11]。

リモートからホストへの更新のサブミット

```
 1  <iq type="set" id="1-1" from="wave.initech-corp.com" to="wave.acmewave.com">
 2    <pubsub xmlns="http://jabber.org/protocol/pubsub">
 3      <publish node="wavelet">
 4        <item>
 5          <submit-request
 6            xmlns="http://waveprotocol.org/protocol/0.2/waveserver">
 7            <delta wavelet-name="acmewave.com/initech-corp.com!a/b">
 8              <![CDATA[CiA...NvbQ==]]>
 9            </delta>
10          </submit-request>
11        </item>
```

[*11] 公式ドキュメントより引用

```
12      </publish>
13    </pubsub>
14  </iq>
```

ホストからリモートへの ACK

```
 1  <iq type="result" id="1-1" from="wave.acmewave.com" to="wave.initech-corp.com
 2    <pubsub xmlns="http://jabber.org/protocol/pubsub">
 3      <publish>
 4        <item>
 5          <submit-response
 6            xmlns="http://waveprotocol.org/protocol/0.2/waveserver"
 7            application-timestamp="1234567890"
 8            operations-applied="2">
 9            <hashed-version
10              history-hash=""
11              version="1234"/>
12          </submit-response>
13        </item>
14      </publish>
15    </pubsub>
16  </iq>
```

こちらも一見 XML のようですが、<delta>タグにはプロトコルバッファが収められていることが分かります。

5.4.2 クライアントサーバープロトコル

現在、公式の wave クライアントは 2 種類あり、一つが 3 章で説明した Google Wave のクライアント、もう一つがプロトタイプ実装 FedOne（5.5 節を参照）の Simple Web Client です。残念ながら前者の使用するプロトコルについては Google 社からまとまった情報がほとんど出てきていませんので、ここでの説明は全て FedOne に寄っています。

FedOne でのクライアントサーバープロトコルは WebSockets 上を流れる JSON メッセージとして定義されています。wavelet はオペレーションのシーケンスから成り、クライアントとサーバーはクライアントサーバープロトコルを使用して常にオペレーションをやり取りしています。クライアントサーバープロトコルでやり取りされるメッセージで主だったものは次の 4 つです。

ProtocolOpenRequest　クライアントが wavelet を開こうとすると送られるメッセージです

ProtocolWaveletUpdate　ProtocolOpenRequest を受けて、サーバーからクライアントへの変更を通知するためのメッセージです。オープンリクエストがスナップショットを要求していればその時点での wavelet のスナップショットを送り、そうでなければオープンリクエストによって示されるバージョンからの差分だけを送ります。ProtocolWaveletUpdate は一度の ProtocolOpenRequest に対して、連続して

何度送っても構いません

ProtocolSubmitRequest クライアントによる wavelet の変更をサーバーへ通知するためのメッセージです。このメッセージは ProtocolWaveletUpdate とは異なり連続して送信することはできず、サーバーからの確認応答を受け取るまで変更はクライアント側に蓄積されます。ProtocolWaveletDelta は一回の変更を表す ProtocolWaveletDelta を複数もつことができます。一つの ProtocolWaveletDelta に含まれる ProtocolWaveletDelta の適用はアトミックに行われ、一部の delta だけが適用されることはありません

ProtocolSubmitResponse ProtocolSubmitRequest の確認応答です

それでは、クライアントサーバープロトコルのメッセージの例として ProtocolWaveletUpdate を見てみましょう。

```
1 {
2   "version":1,
3   "sequenceNumber":20,
4   "messageType":"ProtocolWaveletUpdate",
5   "messageJson":"{\"1\":\"wave://ando-2.local/indexwave!indexwave/w+fl86pkp2st77A\",\"2\":[{\"1\":{\
\"}]}}}],\"4\":[],\"5\":{\"1\":4,\"2\":\"\"}}],\"4\":{\"1\":4,\"2\":\"\"},\"6\":false}"
6 }
```

wavelet のアップデートの内容は messageJson に JSON 形式で入っています。JSON の値として JSON が入っているためエスケープが多く分かりにくくなっています。messageJson の値だけを抜き出してインデントを追加してみましょう。

```
1 {
2   "1":"wave://ando-2.local/indexwave!indexwave/w+fl86pkp2st77A",
3   "2":[{
4     "1":{"1":3,"2":""},
5     "2":"digest-author",
6     "3":[{
7       "3":{
8         "1":"digest",
9         "2":{
10          "1":[
11            {"5":6},
12            {"2":"world "}
13          ]
14        }
15      }
16    }],
17    "4":[],
18    "5":{
19      "1":4,
20      "2":""
```

```
21       }
22     }],
23     "4":{"1":4,"2":""},
24     "6":false
25   }
```

キーが全て数字になっています。実はこれは連合プロトコルでも出てきたプロトコルバッファを JSON にシリアライズしたもので、プロトコルバッファの IDL を見れば JSON のキーになっている数字の意味が分かります。つまり、先ほどの連合プロトコルは XML の中に BASE64 エンコードされたプロトコルバッファが入っていましたが、クライアントサーバープロトコルは JSON の中に JSON にシリアライズされたプロトコルバッファが入っているのです。wave のメッセージやデータモデルを理解するのにプロトコルバッファの IDL の理解は必須です。プロトコルバッファについては 6.1 節で説明しています。

5.4.3　ロボットワイヤプロトコル

ロボットワイヤプロトコルはより簡単にロボットプロトコルとも呼ばれます。ロボットプロトコルは 4.3 節で紹介したロボットライブラリよって隠蔽されているので直接使用することはほとんどありませんが、ロボットライブラリの裏側を知っておくことでデバッグが容易になったり、独自のロボットライブラリを作成することができるようになるでしょう。例えば本書では GAE 以外のサーバーでロボットを動かす 4.3.7 節の例でロボットプロトコルを直接使用しています。

ロボットプロトコルは HTTP 経由でやり取りされる次の 2 つからなります。

- JSON メッセージバンドル
- JSON オペレーションバンドル（JSON-RPC）

5.4.3.1　JSON メッセージバンドル

メッセージバンドルは wave サーバーからロボットサーバーへのイベントの通知に使用され、次のような情報を含んでいます。

- 発生したイベントの情報
- イベントに関連する blips や wavelet の内容
- ロボットアドレスや ProxyFor ID などのロボットに関するメタ情報

メッセージバンドルで通知されるイベントの一覧は表 4.4 を参照してください。結局のところ、このメッセージバンドルをデシリアライズしたものがロボット API のイベントハンドラの引数になっています。メッセージバンドルの内容はイベントの種類やロボットの設定によって異なりますが、例えば `BLIP_SUBMITTED` イベントの通知は次のようになります。

```
1  {
2    "events":[{
3      "modifiedBy": "user@example.com",
```

```
4       "timestamp": 1255935016481,
5       "type": "BLIP_SUBMITTED",
6       "properties": { "blipId": "b+ja8F_Hw4J" }
7     }],
8     "wavelet": {
9       "creationTime": 1255934856713,
10      "creator": "user@example.com",
11      "lastModifiedTime": 1255935016481,
12      "participants": [ "user@example.com","user2@example.com" ],
13      "rootBlipId": "b+ja8F_Hw4J",
14      "title": "",
15      "version": 11,
16      "waveId": "example.com!w+ja8F_Hw4I",
17      "waveletId": "example.com!conv+root",
18      "dataDocuments": { }
19    },
20    "blips": {
21      "b+ja8F_Hw4J": {
22        "annotations": [{
23          "range": { "start": 0, "end": 1 },
24          "name": "conv/title",
25          "value": ""
26        }],
27        "elements": {},
28        "blipId": "b+ja8F_Hw4J",
29        "childBlipIds": [],
30        "contributors": ["user@example.com"],
31        "creator": "user@example.com",
32        "content": "\n",
33        "lastModifiedTime": 1255934856708,
34        "version": 6,
35        "waveId": "google.com!w+ja8F_Hw4I",
36        "waveletId": "example.com!conv+root"
37      }
38    },
39    "robotAddress": "myrobot@example.com"
40  }
```

上記の例では、メッセージバンドルは events 要素、wavelet 要素、blips 要素、robotAddress 要素という 4 つの要素を持っています。

events 要素はイベントを区別するための要素で、type 属性の値からこのイベントが BLIP_SUBMITTED に関するものであることが分かります。また、events 要素の値は配列

になっていますが、これはメッセージバンドルが一つの wavelet に関する複数のイベント情報をまとめて送ることがあるからです。

イベント発生元の wavelet に関する情報は `wavelet` 要素に入っています。

その次の `blips` 要素はイベントに関連する blip の情報を持っています。この要素は要素名が複数形であることから分かるように、blip ID をキーにして複数の blip の情報を保持することができます。今回のイベントは `BLIP_SUBMITTED` なので、本来であればここには少なくともルート blip とサブミットされた blip という 2 つの blip に関する情報が含まれます。ただ今回はたまたまルート blip をサブミットするものであったため[*12]、一つしか blip が含まれていません。この `blips` 要素にどの blip が含まれるかは、ロボット API のコンテキスト（168 ページ参照）で指定できます。

メッセージバンドルの要素の詳細についてはドキュメント[*13]を参照して下さい。

5.4.3.2 JSON オペレーションバンドル

オペレーションバンドルは、イベント（メッセージバンドル）への反応、または Active API を使用した主体的な働きかけなどで、ロボットサーバーから wave サーバーへオペレーションを送信するために使用されます。オペレーションバンドルは JSON-RPC メッセージをまとめたもので、例えば wavelet のタイトルを設定し、blip を作成して、"Hello World"と書き込むオペレーションバンドルの内容は次のようになります。

```
 1 [{
 2   "params": {
 3     "capabilitiesHash": "b31821e"
 4   },
 5   "id": "0",
 6   "method": "robot.notifyCapabilitiesHash"
 7 },
 8 {
 9   "params": {
10     "waveletId": "google.com!conv+root",
11     "waveId": "google.com!w+ja8F_Hw4g",
12     "waveletTitle": "A wavelet title"
13   },
14   "id": "op1",
15   "method": "wavelet.setTitle"
16 },
17 {
18   "params": {
19     "blipId": "b+ja8F_Hw4h",
20     "waveletId": "google.com!conv+root",
21     "waveId": "google.com!w+ja8F_Hw4g",
```

[*12] `events/properties/blipId` の値と、`wavelet/rootBlipId` の値が等しいことからわかります

[*13] http://code.google.com/intl/ja/apis/wave/extensions/robots/protocol.html

```
22      "blipData": {
23        "blipId": "TBD_google.com!conv+root_1",
24        "waveletId": "google.com!conv+root",
25        "waveId": "google.com!w+ja8F_Hw4g"
26      }
27    },
28    "id": "op2",
29    "method": "blip.createChild"
30  },
31  {
32    "params": {
33      "blipId": "TBD_google.com!conv+root_1",
34      "how": 0,
35      "waveId": "google.com!w+ja8F_Hw4g",
36      "text": "Hello World",
37      "waveletId": "google.com!conv+root"
38    },
39    "id": "op3",
40    "method": "document.modify"
41  }]
```

このメッセージは次の 4 つの RPC からなっています。

robot.notifyCapabilitiesHash　capabilities.xml に変更があったときにリロードするため、全てのオペレーションバンドルに自動的に付加されます

wavelet.setTitle　wavelet のタイトルを設定します

blip.createChild　現在の blip に返信する blip を作成します

document.modify　ドキュメントを変更します

オペレーションの名前空間は上に登場した 4 種類で、それぞれ次のようなオペレーションを持ちます。

5.4.3.3 robot オペレーション

robot 名前空間は特定の wavelet に紐づいていないオペレーションを保持しています。

robot.notifyCapabilitiesHash(capabilitiesHash)　このオペレーションはサーバーへ送信する全てのオペレーションバンドルに自動的に追加され、現在のロボットの capabilities.xml ファイルのハッシュを送ります。ハッシュの値がサーバーが現在保持している capabilities.xml と異なる場合には capabilities.xml を再読み込みします

robot.createWavelet(waveletData, message)　新しい wave と wavelet を作成します。waveletData は作成される wavelet の詳細を示すもので、ID として"TBD_"というプレフィクスを持つ仮の値が必要です。サーバーで wavelet が作成されるとすぐに wavelet_created イベントが発生します

robot.fetchWavelet(waveId, waveletId, message)　wavelet を取得します

5.4.3.4　wavelet オペレーション

wavelet オペレーションは特定の wavelet に関係するオペレーションです。

wavelet.appendBlip(waveId, waveletId, blipData)　wave に blip を追加します
wavelet.setTitle(waveId, waveletId, title)　wave のタイトルを設定します
wavelet.addParticipant(waveId, waveletId, participantId)　wave に参加者を追加します。`participantId` は参加者のアドレスです
wavelet.setDatadoc(waveId, waveletId, datadocName, datadocValue)　wavelet にデータドキュメントを追加します。データドキュメントは通常のドキュメント（blip）と互換ですが、多くの場合画面には表示されません

5.4.3.5　blip オペレーション

wavelet の中の特定の blip に関係するオペレーションです。

blip.createChild(waveId, waveletId, blipdId, blipData)　現在の blip に対するリプライを作成します。`blipData` は新しく作成される blip のデータです
blip.delete(waveId, waveletId, blipId)　指定された `blipId` を持つ blip を削除します

5.4.3.6　document オペレーション

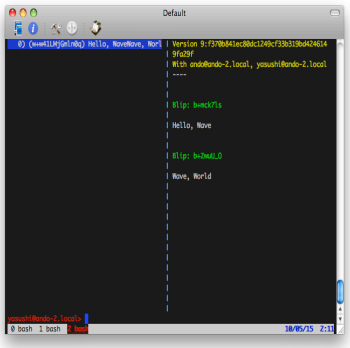
（blip を含む）特定のドキュメントに関係するオペレーションです。

document.appendMarkup(waveId, waveletId, blipdId, content)　現在の wave に `content` パラメータで与えられる HTML（または XHTML）マークアップを追加します。Google Wave 組み込みのコピー＆ペースト機能で使用されている変換がここでも使用されます
document.modify(range, index, modifyQuery, modifyAction)　このオペレーションはドキュメントとエンコードを変更します。変更を適用する場所の指定には `range`・`index`・`modifyQuery` のいずれかひとつだけを使用して指定し、変更の内容は `modifyAction` で指定します

5.5　プロトタイプ wave サーバー

プロトコルの説明の集大成として、最後に Google が公開しているプロトタイプ wave サーバー、FedOne を使って wave サーバーの基本的な動作を確認してみましょう。

5.5.1 FedOne とは



FedOne は Federation One の略で、その名の通り連合（Federation）プロトコルの確認を主な目的として Google が公開し、オープンソースで開発が勧められているプロトタイプ wave サーバーです。wave の機能の内、FedOne に含まれている機能を次に挙げます。

- 連合プロトコル
- オペレーション変換
- 基礎的な wave モデル
- 基本的なクライアント／サーバープロトタイプ
- エージェント

最後の「エージェント」については、これまでの説明には出て来ていませんが、「wave サーバー上で動く特殊なロボット」だと考えてください。エージェントは FedOne 専用のエンティティではなく、Google Wave 上でも Linky[*14]や Spelly[*15]といったエージェントが動作しています。

また、FedOne は FedOne 同士でやりとりするだけではなく、wave サンドボックス（wavesandbox.com）に接続し[*16]wave や操作をやり取りすることもできます。サンドボックスとの接続に関する説明は本書には含まれませんが、興味のある人はぜひ挑戦してみてください。

反対に FedOne に含まれないものとしては次のようなものがあります。

XMPP サーバー 連合プロトコルは XMPP[*17]を利用しますが、XMPP サーバーの機能は FedOne には含まれません。そのかわりに FedOne は XEP-0114 という仕様に基づいて既存の XMPP サーバーに連合プロトコルのための機能を追加します

wave の永続化 現在のところ FedOne 上で作成された wave や操作はすべてオンメモリに保存されサーバー終了と同時に消えてしまいます。ただし、独自に FedOne に永続化機構を組み込もうとしている開発者もいますので、近い将来 FedOne にも wave の永続化が組み込まれるものと思われます

[*14] ユーザーが URL を入力すると自動的にリンクを貼ります
[*15] スペルチェックをして置換候補を提案します
[*16] 執筆時点（2010 年 6 月）では wave プレビュー（wave.google.com）には接続できません
[*17] eXtensible Messaging and Presentation Protocol

各種 API（ロボット・ガジェット・エンベッド）　FedOne は基本的には連合プロトコルの動作確認を目的としているので 4 章で説明したような API は含まれてていません。FedOne で利用できる拡張機能はエージェントのみです

5.5.2　FedOne のセットアップ

5.5.2.1　環境の確認

FedOne のインストールと実行には次のものが必要になります。

- XMPP サーバー[*18]
- Java6: http://java.com
- Ant: http://ant.apache.org
- Python: http://www.python.org
- Mercurial: http://mercurial.selenic.com

これより先に進む前に上記をインストールしておいてください。XMPP サーバーについては XEP-0114 に対応しているものであればなんでも構いません。FedOne のサイト[*19]では次の 2 つの XMPP サーバーのインストール方法が紹介されています。

- OpenFire: http://www.igniterealtime.org/projects/openfire/index.jsp
- Prosody: http://prosody.im

OpenFire は Java 製で本格的な用途にも利用可能な XMPP サーバー、Prosody は Lua 製の軽量 XMPP サーバーです。どちらでも好きな方を使用するといいでしょう。

5.5.2.2　証明書の作成

wave サーバーはドキュメントの認証のために証明書を必要とします。ここでは自己署名証明書を作成しますが、wavesandbox.com と接続するときには自己署名証明書ではなく認証局によって発行された証明書が要求されます。その場合は本節は参考にせず、適切な認証局から証明書を取得しておいてください。

証明書の作成については Google によって簡単なスクリプトが公開されています[*20]。次の内容を持つファイルを作成し、make-cert.sh と名前を付けておいてください。

```
1  #!/bin/bash
2
3  NAME=$1
4
5  if [ "$NAME" == '' ]
6  then
7    echo "$0 <certificate name>" 1>&2
8    exit 1
```

[*18] XMPP サーバーがなくても OT の動作確認は可能ですが、サーバー本来の目的である連合プロトコルは確認できません
[*19] http://code.google.com/p/wave-protocol/wiki/Installation
[*20] http://code.google.com/p/wave-protocol/wiki/Certificates

```
 9  fi
10  openssl genrsa 2048 | openssl pkcs8 -topk8 -nocrypt -out $NAME.key
11  openssl req -new -x509 -nodes -sha1 -days 365 -key $NAME.key -out
$NAME.crt
```

実行権限を与えて、スクリプトを実行します。

```
 1  $ chmod +x make-cert.sh
 2  $ ./make-cert.sh ando.local
 3  Country Name (2 letter code) [AU]:
 4  State or Province Name (full name) [Some-State]:
 5  Locality Name (eg, city) []:
 6  Organization Name (eg, company) [Internet Widgits Pty Ltd]:
 7  Organizational Unit Name (eg, section) []:
 8  Common Name (eg, YOUR name) []:
 9  Email Address []:
```

make-cert.sh の引数は証明書ファイル名に使用されます。FedOne を動作させるドメイン名を指定しておくと、この後の設定が少しだけ楽になります。スクリプト実行中の質問にはどのように答えても構いませんが、Common Name **だけはサーバーのドメイン名を答えるようにしてください**。成功すると.key ファイルと.crt ファイル、今回であれば ando.local.key と ando.local.crt が生成されます。

5.5.2.3 FedOne のインストール

FedOne のソースコードは Python で実装された Mercurial（マーキュリアル）というバージョン管理システムで管理されています。そして FedOne のコンパイルには ant が利用されます。Python と Mercurial、Ant をまだインストールしていない場合はインストールしておいてください。先の 3 つがインストールされていれば次のコマンドを順に実行するだけで FedOne のコンパイルとインストールは完了します。

```
1  $ hg clone https://wave-protocol.googlecode.com/hg/ wave-protocol
2  $ cd wave-protocol
3  $ ant
```

最後の ant の実行に少し時間がかかります（下のログでは 1 分 50 秒ほどですが、5 分ほど待たされたこともあります)。お茶でも淹れて一服するといいでしょう。終了後に次のように表示されていればコンパイルは成功しています。

```
1  BUILD SUCCESSFUL
2  Total time: 1 minute 51 seconds
```

なお、FedOne サーバーは実験サーバーと言う位置づけで、全世界の開発者により非常に頻繁に更新されています。先ほど ant を実行したディレクトリ上で次のコマンドを実行することで、チェックアウトした FedOne を最新版にアップデートできます。

```
1  $ hg pull
2  $ hg update
```

5.5.2.4 FedOne の設定

FedOne をコンパイルしただけでは、まだ実行に必要な設定ファイルがないので、試してみることはできません。まず次のようにしてデフォルトの設定ファイルをコピーしてください。

```
1 $ cp run-config.sh.example run-config.sh
```

この新しく作成した run-config.sh を編集します。執筆時点での全設定項目は表 5.2 のようになっています。

WAVE_SERVER_DOMAIN_NAME	要編集	wave サーバーのドメイン名
WAVE_SERVER_HOSTNAME		wave サーバーのクライアン
WAVE_SERVER_PORT		wave サーバーのクライアン
WEBSOCKET_SERVER_HOSTNAME		サーバーの WebSocket フロ
WEBSOCKET_SERVER_PORT		サーバーの WebSocket フロ
FEDONE_VERSION		FedOne のバージョン（bui
XMPP_SERVER_SECRET	要編集	XMPP サーバーと XEP-01
PRIVATE_KEY_FILENAME		秘密鍵ファイル名
CERTIFICATE_FILENAME_LIST		証明書ファイル名リスト
CERTIFICATE_DOMAIN_NAME		証明書のドメイン名
XMPP_SERVER_HOSTNAME		XMPP サーバーのホスト名
XMPP_SERVER_PORT		XMPP サーバーが使用する
XMPP_SERVER_PING		-
XMPP_SERVER_IP		XMPP サーバーの IP（Fed
WAVESERVER_DISABLE_VERIFICATION		証明書付デルタの検証を無効
WAVESERVER_DISABLE_SIGNER_VERIFICATION		認証局の妥当性を検証するか

表 5.2 FedOne 設定一覧

項目がたくさんあって面食らうかもしれませんが、XMPP サーバーと FedOne が同一ホスト上で動作していて、先に作成した自己署名証明書を使用する場合は次の 3 ヶ所を変更するだけです。

1. 最初の `echo` をコメントアウト
2. `WAVE_SERVER_DOMAIN_NAME` にドメイン名を設定
3. `XMPP_SERVER_SECRET` に XMPP サーバーと XEP-01114 コンポーネントが共有するパスフレーズを設定

もし証明書のファイル名をドメイン名と合わせていなかった場合は、次の 2 つも編集してください。

- `PRIVATE_KEY_FILENAME`
- `CERTIFICATE_FILENAME_LIST`

5.5.3 FedOne を使ってみる

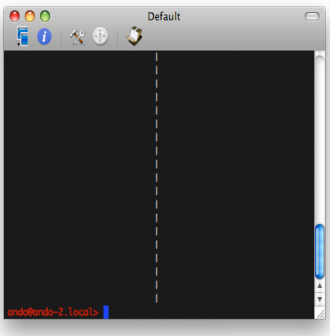
設定ファイルの編集が完了したら FedOne サーバーが起動できます（XMPP サーバーは事前に起動しているものとします)。run-server.sh を実行してください。

```
1  $ ./run-server.sh
2  May 15, 2010 7:48:58 PM org.waveprotocol.wave.examples.fedone.waveserver.WaveServerImpl
<init>
3  INFO: Wave Server configured to host local domains: [ando.local]
4  May 15, 2010 7:48:59 PM org.waveprotocol.wave.federation.xmpp.ComponentPacketTransport
initialize
5  INFO: Initializing with JID: wave.ando-2.local
6  May 15, 2010 7:48:59 PM org.waveprotocol.wave.federation.xmpp.ComponentPacketTransport
start
7  INFO: Connected to XMPP server with JID: wave.ando.local
8  May 15, 2010 7:48:59 PM org.waveprotocol.wave.examples.fedone.ServerMain
run
9  INFO: Starting server
10  2010-05-15 19:48:59.863:INFO::Logging to StdErrLog::DEBUG=false
via org.eclipse.jetty.util.log.StdErrLog
11  2010-05-15 19:48:59.979:INFO::jetty-0.2
12  2010-05-15 19:49:00.090:INFO::Started SelectChannelConnector@localhost:9898
```

FedOne のバージョンによって実行後の細かい内容は異なるかもしれませんが、上のように表示されると FedOne は起動しています。では早速 FedOne サーバーにクライアントを接続してみましょう。クライアントの設定はサーバーと共通なので特に準備は必要ありません。サーバーとは別にターミナルを立ち上げて run-client-console.sh を実行してください。なお引数はクライアントを使用しているユーザー名に使用されます。

```
1  $ ./run-client-console.sh ando
```

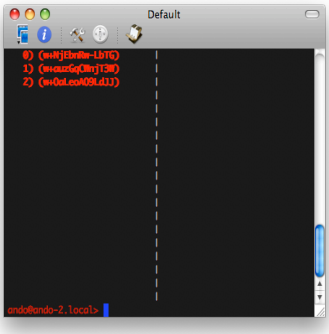
クライアントコンソールを起動すると画面が次のように大きく 3 つのパートに分かれます。



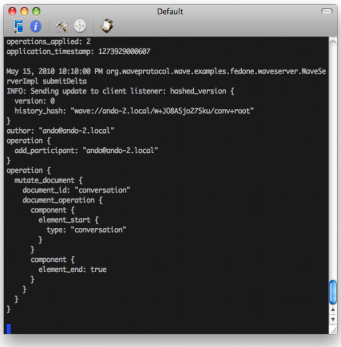
最下部に入力プロンプトがあり、上部は左側が Inbox パネル、右側が wave パネルになります。FedOne で作成したデータは永続化されないので、サーバー起動直後は Inbox

パネルも wave パネルも常に空白です。まず最初に新しく wave を作成してみましょう。wave を作成するには/new コマンドを使用します。

```
1  ando@ando.local> /new
2  ando@ando.local> /new
3  ando@ando.local> /new
```



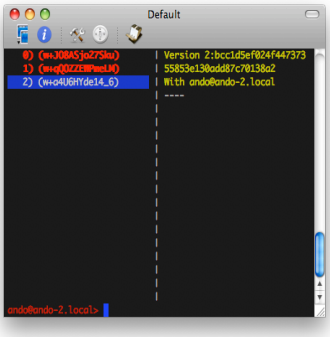
/new コマンドを実行するたびに左側の Inbox パネルに連番と wave ID が追加されます。これだけだとあまりおもしろくもありませんが、ここでサーバーを起動しているターミナルを見てください。



ターミナルにサーバーとクライアントの間でやりとりしているデータの内容が全て表示されています。このように実際にプロトコルを確認することができるのが FedOne のおもしろいところです。

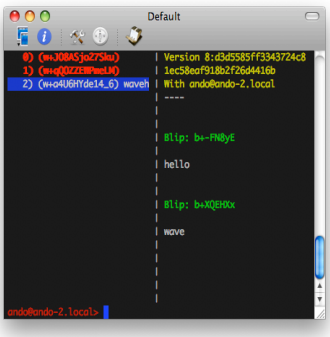
それではこのままもう少しクライアントから FedOne 上の wave を操作してみましょう。次に好きな wave を選んで参加してみます。wave への参加は/open <wave 番号>というコマンドを使用してください。<wave 番号>は Inbox で wave に振られている連番のことです。

```
1  ando@ando.local> /open 2
```



wave に参加すると右側の wave パネルに自分の ID が表示されました。これ以降のコマンドではない（/で始まっていない）入力は全て参加している wave への入力とみなされます。

```
1  ando@ando.local> hello
2  ando@ando.local> wave
```

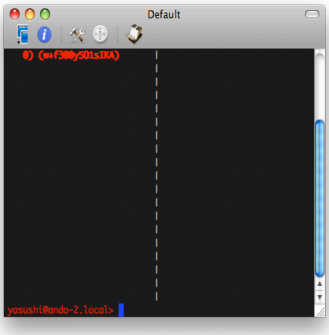


さて、このまま一人で wave の編集を続けていても仕方ありませんので、他のユーザーをこの wave の参加者に追加しましょう。ユーザーの追加には/add <ユーザー ID>コマンドを使用します。追加するユーザー名は任意で構いませんが、ドメインは自分と同じにしておいてください。

```
1  ando@ando.local> /add yasushi@ando.local
```

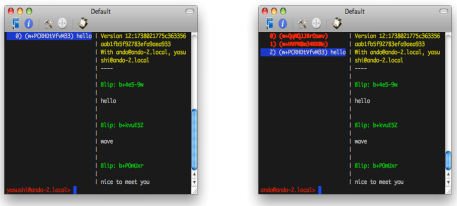
ユーザーを追加したら、ターミナルをもう一つ立ち上げ、先ほど追加したユーザー名でコンソールクライアントを起動します。

```
1  $ ./run-client-console.sh yasushi
```



すると今度は最初から Inbox に wave が一つ表示されていることが分かるでしょう。さっそく/open <wave 番号>[*21]コマンドを使用して wave をオープンして、メッセージを追加してみます。

```
1  yasushi@ando.local> /open 0
2  yasushi@ando.local> nice to meet you
```



ユーザー yasushi の wave にメッセージが表示されるだけではなく、ユーザー ando の wave にも同じメッセージが追加されます。どうでしょう？非常に素朴ではありますが、本質的には FedOne は Google wave と同じものだということが分かったのではないでしょうか？

なおコンソールクライアントではここで使用したコマンド以外にも次のようなコマンドが利用できます。

コマンド:

connect	user@domain server port	user@domain として server:port に接続する
open	entry	与えられた inbox エントリを持つ wave を開く
new		新規 wave を作成する
add	user@domain	wave にユーザーを追加する
remove	user@domain	wave からユーザーを削除する
read		全ての wave を既読にする
scroll	lines	{と}によってスクロールされる行数を設定する
view	mode	開いている wavelet の表示モード

[*21] ユーザーごとの連番なので同じ wave を指す場合でも wave 番号が他のユーザーと異なる場合があります。なお、wave 番号の横に表示されている wave ID（"w+.."という形式の ID）はユーザーによらず wave ごとに固定です

(normal, xml) を変更する

log	**ログを画面にダンプする**
dumplog file	**ログをファイルにダンプする**
clearlog	**ログをクリアする**
quit	**クライアントを終了する**

スクロール:

{ **開いている** wave **を上にスクロールする**

} **開いている** wave **を下にスクロールする**

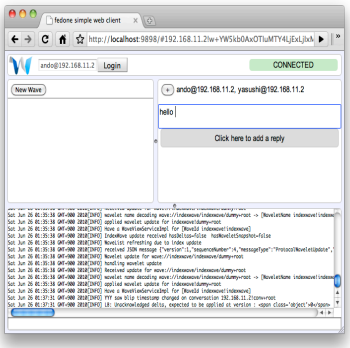
5.5.4 ウェブクライアントを使用する

執筆時点ではまだ trunk ディレクトリにはありませんが、FedOne の io2010 ブランチに簡単なウェブクライアントが付属しています。CUI クライアントではできないクライアントサーバープロトコルの確認もウェブクライアントなら可能です。実行の手順は trunk と何もかわりませんので、ウェブクライアントを一足先に試したい人は次のリポジトリから取得して試してみましょう。

```
1 $ hg clone https://io2010.wave-protocol.googlecode.com/hg/ wave-protocol-io2010
```

コンパイルや設定、サーバーの起動は前項と全く同じです。サーバーが起動したらブラウザを開き、次のアドレスにアクセスしてください。なお、ウェブクライアントはクライアントーサーバー間の通信に WebSockets を使用しています。そのため比較的新しいブラウザでしか動作しません。Chrome・Safari5 または WebKit の Nightly build を使用してアクセスしてください。

- http://localhost:9898/



ログインアカウントは [ユーザー名]@[ドメイン] という形式になります。CUI クライアントと相互にやり取りすることもできますし、クライアントサーバープロトコルの内容を画面下方のパネルで確認することもできます。クライアントサーバープロトコルについては現時点では文書化された情報がほとんどありませんので、興味がある人はここでいろいろと試してみるといいでしょう。

第 6 章

付録

6.1 プロトコルバッファ

6.1.1 はじめに

プロトコルバッファは Google がサーバー間のデータ交換に利用している構造化データをシリアライズするためのツールで、連合プロトコルやクライアントサーバープロトコルの例にも見られたように、Google Wave でも内部的に使用されています。wave をプロダクトやプラットフォームとして使うだけであればプロトコルバッファの知識は不要ですが、プロトコル、特に FedOne の内部実装を理解するには知っておくと便利ですので、ここで簡単に紹介しておきます。詳細については公式ドキュメント[*1]を参照してください。

6.1.2 特徴

構造化データをシリアライズする方法は一つではありません。プロトコルバッファ以外のフォーマットもいくつも存在します。その中でも最もよく使われているものといえばやはり XML でしょう。XML と比較するとプロトコルバッファには次のような特徴があります。

定義ファイル プロトコルバッファはデータ構造を記述するためのシンプルな言語を持ちます。シリアライズされたデータは、定義ファイルの何番目の要素がどのようなバイト列を値として持つかを示すだけで、その構造に関する情報を持っていません。そのためデータのシリアライズ・デシリアライズには定義ファイルまたはそこから生成されたオブジェクトを必要とします。[*2]

バイナリフォーマット プロトコルバッファはバイナリフォーマットです。すべて文字列に変換してしまう XML とは違い、データ型に応じたフォーマットでシリアライズされいます

データサイズが小さい 上記の 2 点の項目の当然の帰結として、XML よりもデータサイズが大幅に小さくなります。例えば整数を一つシリアライズする場合を考えてみましょう。XML は仮に`"<int>150</int>"`とすると、データサイズは文字コードに

[*1] http://code.google.com/intl/en/apis/protocolbuffers/

[*2] ネットワーク経由でプロトコルバッファを受け渡す場合は、エンドポイント同士で事前に定義ファイルを共有しておくほか、定義ファイル自体をプロトコルバッファでシリアライズして送ることもできます

もよりますが UTF-8 であれば 14 バイトです**もちろん XML 宣言を追加するとさらに大きくなります**。これをプロトコルバッファで表せば 08 96 01 のたった 3 バイトで済みます。ドキュメントによると XML と比較してデータサイズが 3-10 倍ほど小さくなるようです

パースが高速 データサイズが小さく、また定義ファイルから処理用のコードを生成できることによりシリアライズされたデータのパースが高速です。XML と比較すると 20-100 倍は高速であるとドキュメントには書かれています

スキーマ変更に強い プロトコルバッファは不明なフィールドを単に無視します。そのため新しいバージョンのメッセージに新しいフィールドを追加したとしても、旧バージョンメッセージ用のプログラムを変更する必要はありません。また拡張用の領域を事前に指定しておくこともできます

コード生成 先に少し書きましたが、プロトコルバッファでは定義ファイルからそのデータを処理するためのコードを生成することができます。公式のライブラリは C++・Java・Python の 3 言語だけですが、C・Objective-C・Perl・Ruby など多くの言語用のライブラリが有志により開発されています[*3]

RPC プロトコルバッファはシリアライズのためのコードだけではなく、それらを利用して RPC を実現するコードの雛形も生成できます。FedOne はこれにより生成されたコードを利用しています

6.1.3 .proto ファイル

プロトコルバッファの定義ファイルは通常.proto という拡張子を持ちます。FedOne にも.proto ファイルが含まれていますので、プロトコルバッファの定義ファイルがどのようなものなのか、FedOne で実際に使用されている定義を例に見てみましょう。

（src/org/waveprotocol/wave/examples/fedone/waveserver/waveclient-rpc.proto より一部抜粋）

```
1 message ProtocolWaveletUpdate {
2   required string wavelet_name = 1;
3   repeated federation.ProtocolWaveletDelta applied_delta = 2;
4   optional federation.ProtocolHashedVersion commit_notice = 3;
5   optional federation.ProtocolHashedVersion resulting_version = 4;
6   optional WaveletSnapshot snapshot = 5;
7   optional bool marker = 6 [default=false];
8   optional string channel_id = 7;
9 }
```

プロトコルバッファのメッセージ定義は次のような形式になります。

```
1 message メッセージ名 {
2   修飾子 型 フィールド名 = タグ [オプション];
3 }
```

[*3] http://code.google.com/p/protobuf/wiki/ThirdPartyAddOns

メッセージ名 大文字始まりのキャメルケースにします

フィールドの修飾子 required、repeated、optional の 3 つが使用でき、それぞれ次のような意味を持ちます

required 整形式のメッセージは必ず一つこのフィールドを持たなければいけません

optional 整形式のメッセージはこのフィールドを省略するか、一つだけ持つことができます

repeated 整形式のメッセージはこのフィールドを任意の個数（ゼロ個を含む）持つことができます。フィールドの順序は維持されます

フィールドの型 string や bool のような定義済みの型の他に、federation.ProtocolWaveletDelta のように自分で定義したメッセージが指定できます

フィールド名 小文字アルファベットとアンダースコアを使用してください

タグ タグは各フィールドに設定するメッセージ内で一意な数値です。バイナリエンコードされたデータ内でフィールドはこのタグの値によって識別されます

オプション デフォルト値などのいくつかのオプションをフィールドに設定できます

従って ProtocolWaveletUpdate メッセージは、wavelet_name フィールドが必須で、applied_delta フィールドはゼロ個以上複数含まれる可能性があり、それ以外のフィールドは全て任意です。このようにプロトコルバッファの定義ファイルを見れば wave で使用されるデータモデルの詳細が分かります。

また、定義ファイルにはメッセージ定義だけではなく、それらを使用した RPC も定義されています。

（src/org/waveprotocol/wave/examples/fedone/waveserver/waveclient-rpc.proto より一部抜粋）

```
1  service ProtocolWaveClientRpc {
2    rpc Open (ProtocolOpenRequest) returns (ProtocolWaveletUpdate) {
3      option (rpc.is_streaming_rpc) = true;
4    };
5  }
```

上記は FedOne で使用されている RPC 定義の例です。ここでは ProtocolOpenRequest メッセージを送信して、その結果を ProtocolWaveletUpdate メッセージとして受け取る Open RPC が定義されています。Open RPC は wave クライアントが wavelet を開くときに実行される RPC で、一度のリクエストに対して更新内容 ProtocolWaveletUpdate を複数受け取る場合があるので、オプションとして rpc.is_streaming_rpc が true に設定されています。この RPC は 5.4.2 節で例として取り上げたものですが、見比べると JSON の内容がこのメッセージ定義と対応していることが分かるでしょう。

proto ファイルで定義されるのはあくまでもインターフェースだけで、実際の処理はこの定義ファイルから生成される雛形を元に開発者が実装することになります。しかし、それでもコメントや変数名を通して定義ファイルだけでもかなりの情報が読み取れます。FedOne の src ディレクトリにはいくつかの proto ファイルがありますので、wave プロトコルに興味がある方は一通り目を通しておくといいでしょう。

6.2　オペレーション変換

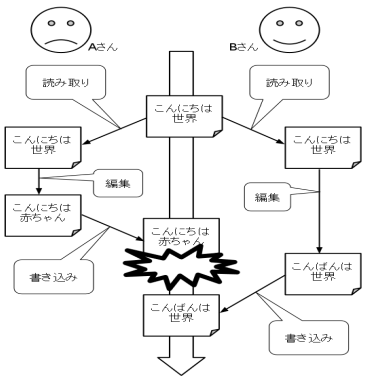
オペレーション変換（Operation Transformation, OT）は誤解を恐れず簡単にまとめると「編集の衝突を解消するためのアルゴリズム」で、wave 専用のものではありませんが、その根幹を成しているアルゴリズムだと言っても全く過言ではありません。OT を理解すると本章でこれまで説明してきたプロトコルがなぜそのようになっているのかが理解できるでしょう。本節ではまず一般的な OT について説明し、その後で wave において拡張された部分について説明します。

6.2.1　基本的な OT

OT はインターネットなどの低帯域で遅延のある環境でドキュメントを共同編集するために開発された技術です。その歴史は意外と古く 20 年ほど前にはすでに基本的な部分は確立されていました。本節ではまずそのベーシックな OT について説明します。

6.2.1.1　解決すべき問題

まず初めに OT が解決しようとしている問題について考えてみましょう。遅延のある環境で、一つのドキュメントを同時に複数人で編集した場合に起きる問題、それは（おそらく容易に想像がつくと思いますが）変更の衝突です。下の図を見てください。



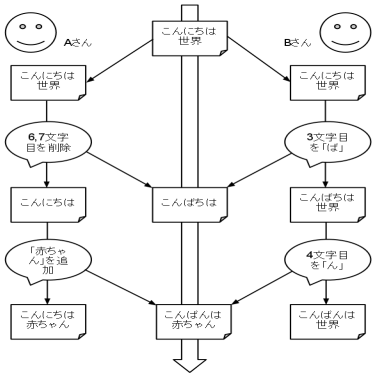
A さんと B さんが同時にドキュメントを編集した結果、最終的に A さんの編集した内容が消えてしまっています。

このような事態を防ぐために真っ先に思いつく対策は変更前にドキュメントにロックをかけてしまうことでしょう。つまり、A さんはファイルを変更しようと思いついたときにドキュメントをロックし、他の人がドキュメントを読み書きできないようにします。そして変更が終わったらロックを解除して他の人にドキュメントを開放するのです。しかしこの方法では常に一人しかドキュメントにアクセスできず、共同編集と言いながらも作業のそこかしこで遅延が生じてしまいます。

ドキュメントをロックする方式の問題は、排他的にロックを掛け合うことで、まるで共同編集者たちの間でドキュメントそのものがやり取りされているような状況を発生させていたことです。これでは回覧板を回しながら各自が順にコメントを追加しているのとたいして変わりません。そうではなく、まるで編集者全員がホワイトボードを囲んで同時に書

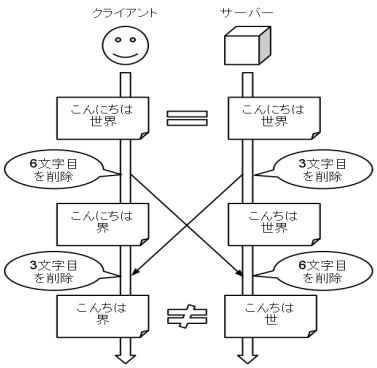
き込んでいるような状態にするにはどうすればいいのでしょう。

6.2.1.2 オペレーション変換

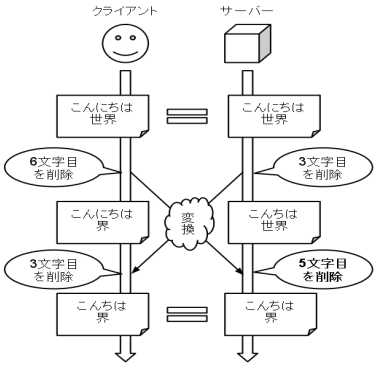


OT ではドキュメント（のロック）を奪い合うのではなく、上の図のようにドキュメントの変更方法（Operation）をやりとりします。図の左右で A さんと B さんが独自に自分用のコピーを編集していますが、中央の共有ドキュメントには両者の変更が同時に反映されている様子がわかるでしょう。

しかし実はロックする代わりに操作をやり取りするだけではまだ問題の解決にはならず、状況が振り出しに戻っただけです。先の図では共有ドキュメントの同期だけを考えて、それぞれのユーザーが見ているドキュメントの同期は考えていませんでした。今度はサーバー上の共有ドキュメントとクライアント上のドキュメントの同期も含めて考えてみましょう。

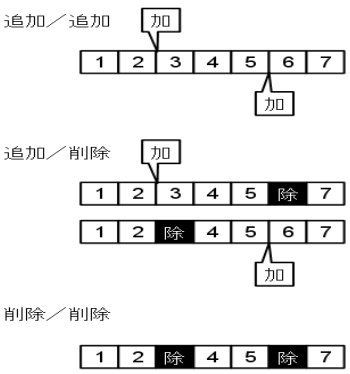


上図ではドキュメントが全く同時に編集された結果、処理が交差してしまい、最終的にドキュメントの内容が異なってしまっています。このような状態に陥ることを防ぐため OT ではドキュメントに操作を適用する前に、その操作を変換（Transformation）します。



例えばクライアント上で 6 文字目を削除して通知するのと同時に、サーバーは他のクライアントから 3 文字目を削除すると言う操作を受けてそれをクライアントに送るとします。その場合、クライアントはサーバーから送られてきた操作をそのまま適用し、サーバー側では 3 文字目がすでに消えていることを検討にいれて 5 文字目を削除すれば、お互いのドキュメントに齟齬は生じません。このようにドキュメントへの操作 (Operation) をやり取りして、それぞれの操作を必要に応じて変換 (Transformation) してからドキュメントに適用すると言うのが、OT の基本的なアイデアです。

操作の変換についてもう少し詳細に見てみましょう。説明を単純にするためにドキュメントへの操作は一文字ずつの追加と削除という 2 種類しかないものと考えまると、操作の組み合わせは、追加/追加、追加/削除、削除/削除の 3 種類だけになり、それぞれのパターンについて適用順序が入れ替わったとき（つまり衝突が発生したとき）に行うべき変換は次のようになります。



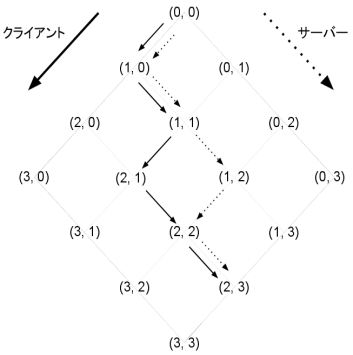
6.2.1.3　状態空間

さて、必要に応じて上のように操作を変換すれば複数のクライアントが待ち時間なく編集でき、かつドキュメントの不整合も発生しないことが分かりました。残る問題は「変換が必要かどうかをどのように知るか」という問題です。

実は OT においてクライアント／サーバー間でやり取りされるのは操作だけではありません。それに加えて「現在送信されている操作の対象になったドキュメントは、サーバーで発生した操作とクライアントで発生した操作をそれぞれ何回ずつ適用されたものか」という情報が付随しています。これはドキュメントの状態空間を考えると分かりやすいでしょう。図に現れる数字の組は (クライアントからの操作の適用回数, サーバーから

追加／追加	
追加位置が異なる	後ろの操作の追加位置を一つ後ろにずらす
追加位置が同じ	任意の順序で追加を実行する
追加／削除	
追加位置が削除位置の前	削除位置を一つ後ろにずらす
削除位置が追加位置の前	追加位置を一つ前にずらす
追加位置と削除位置が同じ	操作位置は変更しないが、削除を先に実行する
削除／削除	
削除位置が異なる	後ろの操作の削除位置を一つ前にずらす
削除位置が同じ	一度だけ削除を実行する

の操作の適用回数) を表しています。



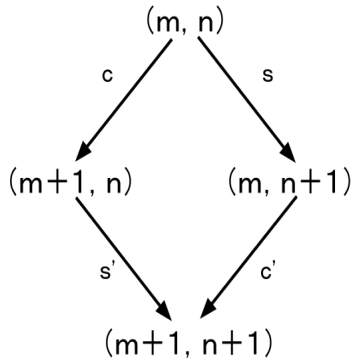
*4

クライアントからの操作が適用されるとドキュメントの状態は左下に進み、サーバーからの操作が適用されるとドキュメントの状態は右下に進みます。クライアントとサーバーは同じ状態（座標）にいるときはそれぞれが見ているドキュメントの内容は一致していますが、クライアントとサーバーの状態（座標）が異なればドキュメントの内容も異なります。つまりクライアントとサーバーが同じ経路を進んでいるときは同期が保たれていて、分岐点があればそこで変更が衝突したと言うことです。先の図であれば、状態 (1,1) までは同期が取れていましたが、そこでクライアントとサーバーで同時に違う操作が発生してしまい、それぞれ (2,1) と (1,2) に状態が分離しています。

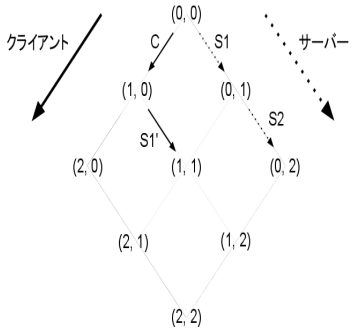
この状態（座標）が操作と共に送信されるのです。操作を変換する必要があるのは、操作を受け取ったときにその操作が対象としているドキュメントと、これからその操作を適用しようとしているドキュメントの状態が異なるときです。

まず操作を変換すると言うことが、状態空間のどういう遷移を表してるのか確認してみましょう。クライアントで発生した操作を c、サーバーで発生した操作を s、変換された操作をそれぞれ c', s' とすると、クライアントとサーバーそれぞれの状態空間上の経路は次のようになります。

*4 High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System, Fig. 4



状態 (m, n) に操作 c と s' を順に適用しても、操作 s と c' を適用しても、どちらも状態 (m+1, n+1) へ遷移しています。つまり c と s を操作変換して得られる c' と s' は、操作 c, s によって分岐してしまった経路を再び統合するための操作になっているのです。ただし、c'、s' で衝突が解消できるのは衝突の直後だけです。衝突に気づかずにいくつも操作を送ってしまった場合には単純に c', s' を使用して解消することはできません。次の図を見てください。



*5

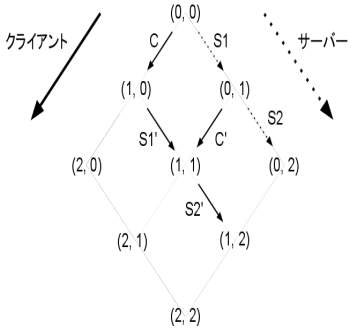
上の図では操作 c と操作 s1 が衝突し、クライアントとサーバーで状態が異なってしまったまま、さらにサーバー上のドキュメントに操作 s2 を適用しています。クライアント上で一度、サーバー上でニ度、ドキュメントを操作しているので、最終的には状態 (1,2) にたどり着く必要があります。しかし、クライアント上のドキュメントを状態 (1,1) から状態 (1,2) に遷移するための操作 s2' も、サーバー上のドキュメントを状態 (0,2) から状態 (1,2) に遷移するための操作 c" も、c と s2 を単純に OT して求めることはできません。

ここでクライアントの視点に立って状況を考えてみましょう。クライアントがわかっていることは次の 3 つです。

1. 状態 (0,0) のドキュメントに対して、クライアントで発生した操作 c1 を適用した
2. 状態 (0,0) のドキュメントに対して、サーバーで発生した操作 s1 を受け取った
3. 状態 (0,1) のドキュメントに対して、サーバーで発生した操作 s2 を受け取った

*5 High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System, Fig. 5(a)

1 と 2 から衝突が発生していることは分かりますし、3 からその衝突に気付かずサーバー上のドキュメントにさらに操作が適用されたと言うこともわかります。要するにクライアントはこの状態空間を完全に把握しています。そしてその前提があれば状態 (1,1) から状態 (1,2) に遷移する操作 s2' を求めることも簡単です。



*6

図から分かるように操作 s2' は操作 c' と操作 s2 に対して OT を適用した操作です。そして操作 c' は操作 c と操作 s1 に対して OT を適用した結果得られる操作です。従ってクライアントが操作 c を送信後にサーバーから s1, s2 を受け取ったとき、ドキュメントに正しく操作を反映する手順は次のようになります。

1. 状態 (0,0) に操作 c を適用
   （a）クライアントの状態 (1,0) に遷移
   （b）サーバーに c を送信
2. s1, s2 を受信
3. s1 が状態 (0,0) に対する操作であることから衝突が発生したことを把握
4. s1 の衝突を解消
   （a）c と s1 から c' と s1' を生成
   （b）s1' を使用して状態 (1,1) に遷移
5. s2 の衝突を解消
   （a）4-1 で生成した c' と s2 から s2' を生成
   （b）s2' を使用して状態 (1,2) に遷移

サーバーもクライアントと全く同様にしてドキュメントを正しい状態に遷移させることができます。

6.2.1.4 擬似コード

先項の手順は一般化でき、操作（と操作対象の状態）を正しく受信できていればどのような衝突でも解消できます。実際にどのようなアルゴリズムになるか、疑似コードで見てみましょう。

```
1  送信した操作の数 = 0
2  受信した操作の数 = 0
```

*6 High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System, Fig. 5(b)

```
 3  相手に適用されていない操作のキュー = []
 4
 5  def 発生(操作):
 6    操作をローカルドキュメントに適用
 7    送信(操作，送信した操作の数，受信した操作の数)
 8    相手に適用されていない操作のキュー.追加(操作，送信した操作の数)
 9    送信した操作の数 = 送信した操作の数 + 1
10
11  def 受信(受信メッセージ):
12    # 受信を確認したメッセージをキューから削除
13    for キュー内のメッセージ in 相手に適用されていない操作のキュー:
14      if (キュー内のメッセージ.送信した操作の数 < 受信メッセージ.受信した操
作の数):
15        相手に適用されていない操作のキュー.除去(キュー内のメッセージ)
16
17    # ASSERT 受信メッセージ.送信した操作の数 == 受信した操作の数
18    for i in [1..長さ(相手に適用されていない操作のキュー)]:
19      # 受信したメッセージとキュー内のメッセージを操作変換(OT)する
20      [受信したメッセージ，相手に適用されていない操作のキュー[i]] = \
21        操作変換(受信したメッセージ，相手に適用されていない操作のキュー[i])
22
23    受信メッセージ.操作 をローカルドキュメントに適用
24    受信した操作の数 = 受信した操作の数 + 1
```

6.2.1.5　まとめ

以上が、最も標準的な OT の説明です。OT を利用すればクライアントに遅延を感じさせずクライアント・サーバー間でドキュメントの整合性を保つことができることが分かって頂けたと思います。なお、ここまでの説明は Google Wave の参考文献にも上がっている「High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System」[*7]という論文に因りました。詳細については当論文または Wikipedia[*8]などを参照してください。

6.2.2　Google による OT の拡張

wave で使用されている OT は前項で説明したそのままのものではなく、独自に拡張されています。本節ではその拡張点をいくつか紹介します。

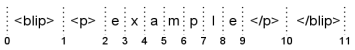
[*7] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping: High-latency, low-bandwidth windowing in the Jupiter collaboration system, UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology, pp.111-120. ACM, 1995.

[*8] http://en.wikipedia.org/wiki/Operational_transformation

6.2.2.1 オペレーションの拡張

wave のドキュメントはアノテーションが付加された XML ドキュメントですから、OT も当然それらを扱うことができるように拡張されています。例えば XML 要素を追加／削除すると言った操作や、アノテーションを設定・更新するという操作が追加されています。

ドキュメントの操作においてタグはアトミックに扱われ、操作のポジションをタグの内部に設定することはできません。



また、アノテーションはドキュメントの一定の範囲に設定できるメタデータですが、アノテーションの範囲も必要であれば OT によって適切に変換されます。

6.2.2.2 サーバーでの状態管理の簡略化

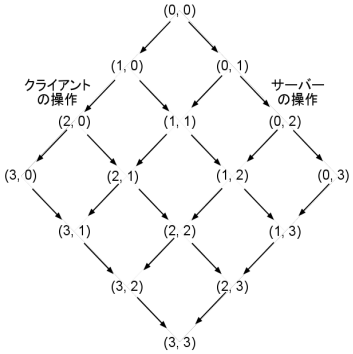
前節の説明の通り、通常の OT ではサーバーはクライアントごとに状態空間を管理します。つまり 1000 クライアントを処理するサーバーであれば、1000 個の状態空間を管理して、それぞれの状態空間で受け取った操作を変換しなければいけません。さらにクライアントの数が増えれば増えただけ、際限なくサーバーの負荷は増大していきます。イントラであれば最大ユーザー数も予想がつき、事前に十分な能力のサーバーを用意することもできるでしょうが、wave はインターネット上に公開され、ゆくゆくはメールを置き換えて全世界で利用されることを目指しています。本質的にスケールしない方式を採用するのは得策ではありません。では、どうするのか。クライアントが増えたことで必要になる処理なら、その増えたクライアントに処理を任せればいいのです。

wave のクライアントで発生した操作はまずクライアント内のキューに蓄えられ、以前に送信した操作の確認応答（ACK）をサーバーから受け取るまで、サーバーには送信されません。つまり、サーバー側のドキュメントに未適用な操作はそれぞれのクライアントでただだかひとつです。これによってクライアントは、サーバーのドキュメントに現在適用されている操作が何であるか、つまりサーバードキュメントの状態空間内での位置を把握できます。サーバードキュメントの状態と、その状態のサーバードキュメントに適用したい操作があれば、操作変換をサーバー上で行う必要はありません。それらを利用して、クライアントは事前に必要な変換を済ませてからサーバーに操作を送信することができます。

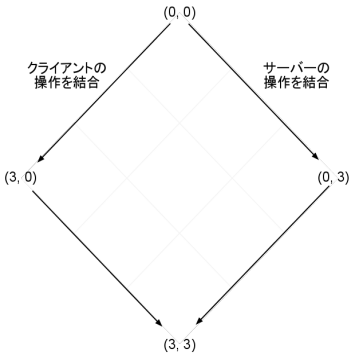
つまりサーバーの OT パスに沿った操作をクライアントが送ってくるのでサーバーはクライアントごとに異なる状態空間を管理する必要がなくなり、オペレーションのキューがひとつだけで済むようになります。

6.2.2.3 操作の結合

サーバー上とクライアント上で独立して複数の操作が行われると、それらを操作変換してドキュメントを正しい状態に遷移させるのは非常に重たい処理になります。次の図を見てください。

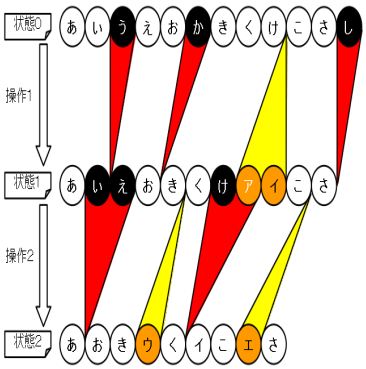


もしクライアント上で n 回の操作が行われ、サーバー上で m 回の操作が行われたとすると、ドキュメントを同期するためには nm 回の操作変換が必要です。しかし連続した操作を一つにまとめることができるとどうでしょう。



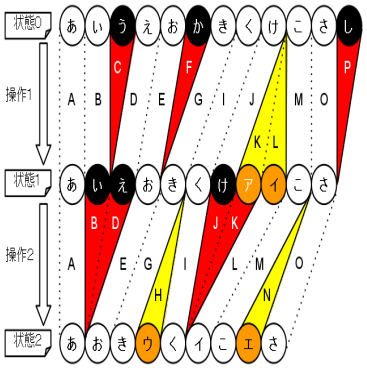
それよりもずっと少ない回数の変換でドキュメントを同期することができました。ただし、これにより処理全体が高速化するには操作変換に比べて操作の結合が効率的であることが前提です。操作変換の回数が少なくなっても操作の結合に時間がかかって採取的な効率が悪化したのでは本末転倒ですが、wave では操作を結合しなかった場合の計算量 `O(mn)` に対して、操作を結合することにより `O(n*log(n) + m*log(m))` に減らすことに成功しているそうです[*9]。

それでは、操作がどのように結合されるかを簡単な図でみてみましょう。説明を簡単にするため、操作の種類は削除（▽）と挿入（△）のみとしています。



[*9] Google Wave: Under the Hood http://www.youtube.com/watch?v=uOFzWZrsPV0

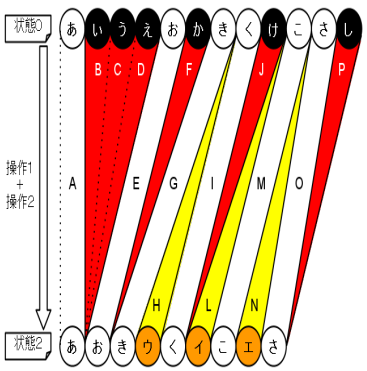
状態 0 のドキュメントに操作 1 を適用した結果が状態 1 のドキュメントで、その状態 1 のドキュメントに操作 2 を適用した結果が状態 2 のドキュメントです。つまり操作を結合すると言うことは、状態 1 を削除して状態 0 から状態 2 に一息に変化できる操作を求めるということになります。それにはまず操作 1 の適用結果が状態 2 のどの位置に対応するか、操作 2 の適用対象が状態 0 のどの位置に対応するかをマークします。



そしてそれらを次のルールに従って結合します。

操作 1	操作 2	結合の結果	例
-	-	-	A,E,G,I,M,O
-	削除	削除	B,D,J
削除	-	削除	C,F,P
-	追加	追加	H,N
追加	-	追加	L
追加	削除	-	K

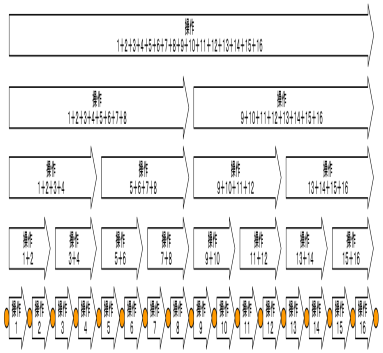
表 6.1　操作の結合



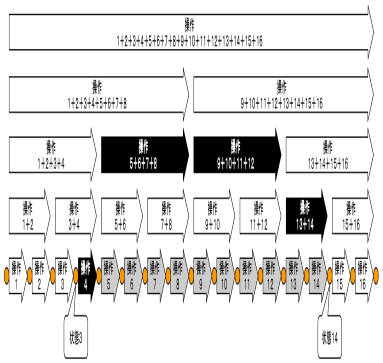
これで状態 1 を取り除いて状態 0 から状態 2 に遷移する操作を得ることができました。同様の手順を繰り返すことで操作がいくつ連続していたとしても一つにまとめることができます。

6.2.2.4　操作のコンポジションツリー

wave のドキュメントの実体は操作のシーケンスになっていて操作の履歴がすべて残っているので、編集の履歴を自由に遡ったり先へ進めることができます。しかし、例えば n 回の操作を遡ってドキュメントを確認したいときに、そのまま n 回操作を逆変換して適用するようなやり方では、n が大きくなればなっただけ待ち時間も大きくなり、いずれ破綻するというのは明らかでしょう。そのような事態を避けるため wave ではドキュメントの持ち方を少し工夫します。



上の図のように wave では一つ一つの操作に加えて、操作をバイナリツリー状に結合したものを同時に保持しています。たとえば操作が 16 あるときには、2 つずつの操作、4 つずつの操作、8 つずつの操作、全ての操作、を事前に結合しておきます。



そして状態を遷移したいときには、事前に結合された操作を利用して状態間の操作数が最小になるパスを選択します。例えば、状態 3 から状態 14 に遷移するには本来であれば操作を 11 回適用する必要がありますが、図のように操作を選べばたった 4 つの操作を適用するだけで遷移を完了できます。

あとがき

Google Wave を初めて見たのは今年の Google I/O[*10]二日目の基調講演だったんですが、あの熱狂は実際大したものでした。プレゼンが終わったときにはあのだだっ広い会場ほぼ総立ちだったし、ノートパソコンぶんぶん振り回して叫んでる人とかいたし。で、翻って今の日本の現状。Twitter みててもだいたいが、重いーとか、バグるーとか、意味分かんねーだとか、いまいちな反応ばかり。その辺がちょっと残念なわけです。

Wave がプロダクトとしてまだちょっと微妙な完成度というのは正直事実だと思います。ただ、Wave の本質はそこじゃありません。Wave で重要なのは「任意の XML ドキュメントをインターネットのような遅延の大きい環境の下でも同時編集できる」こと、つまりそれを実現するプロトコルこそが本質です。

XML と聞くと単なる構造化されたテキストデータだと思う人もいるかもしれませんが、例えば、MS Office のデフォルトの保存形式は XML です。つまり Wave プロトコルを使用すればインターネット越しに Word や Excel のドキュメントを同時編集することが可能になります。 ベクターグラフィックも楽譜も 3D も、実際のところほとんどなんだって XML で表現できます。もちろんパフォーマンスはバイナリフォーマットの独自プロトコルよりも大分劣るでしょうが、「人間が同時に編集する」というシチュエーションで、人間の反応速度を超えるようなパフォーマンスはそもそも不要です。

今の Google Wave の先、そこにはネット上のありとあらゆるドキュメントがリアルタイムでユーザーに編集され正しい姿を目指して常に形を変え続ける[*11]、そういう未来があります。どうでしょう、そう考えるとわくわくしてきませんか？

Google Wave はよく見るベータバージョンではなく、あまり耳にしない preview バージョンとして公開されています。これは Google がいつもの Google プロダクトよりもすこし手前の段階、例えばサービスの方向性やユースケースなどについてもユーザーの意見を求めているということではないでしょうか？

もちろん API を利用してちょっとしたゲームを開発するのも楽しいことです。でも API をそういう用途にしか使わないのは少しもったいない話です。あらゆるものがリアルタイムに変化し続ける Web とはどういうものか。公開されている API やプロトコルを利用して皆さんの考える新しい Web のありようを Google に伝えることができたら、5 年後 10 年後にはそのアイデアが世界を変えていることだってありえます。

Google Wave API Japan グループはそういうアイデアを育てる場であったり、実現を助ける場であれたらいいなと思っています。

以上は「Google-Wave-API-Japan[*12]」のキックオフミーティング翌日に私が書いたブログ記事からの引用ですが、本書への想いも同じです。

[*10] 去年の文章なので Google I/O 2009 のことです
[*11] 個人的に Smalltalk 環境のようなものを妄想してます
[*12] http://groups.google.co.jp/group/google-wave-api-japan

Google Wave の発表から一年が過ぎ、あの時の熱狂はもう跡形もないけれど、その代わりに Google Wave を普通の便利なサービスとして日常的に使う人が少しずつ増えているように感じています。E メールが決して熱狂を持って迎えられるようなツールではないように、それを置き換えようとしている Google Wave も、現在のような緩やかな広がり方の方が当初の熱狂よりむしろ相応しいのかもしれません。

しかし、そんなふうに考えつつも、やはり私はそれだけでは少し残念なのです。革新的なウェブサービスが公開されることはそれほど珍しい話でもないかもしれませんが、そのプロトタイプやプロトコルの仕様までもが同時に公開されるとなるとそうある話ではありません。この本がリアルタイムウェブの広まりを少しでも加速するきっかけになればと思います。

いつか、Google I/O2009 に参加したことを思い出して「Web の世界が変わる瞬間に居合わせた」と言える日が来ることを願いつつ。

2010 年 7 月 多摩川の辺りにてあんどうやすし

著者略歴

あんどうやすし
Google **公認** API Expert（Google Wave）

九州大学大学院 工学研究科（航空宇宙工学専攻） 修了
Bath University, MSc Computer Science
現、シーサー株式会社勤務

オブジェクト指向ならなんでもいいかと思っていたけど Perl **のそれは流石にきつ過ぎませんか？と会社で言い続けていたら、気がつけば** Objective-C **担当に。パワフルな言語だとは思いつつも、異なる文法が混ざりあったソースコードをいじっていると、いっそのこと** Smalltalk **を採用してくれればなぁと思う** Ruby **エンジニア。**