

知的Webのための マッシュアッププログラミング

新谷虎松 大園忠親

(名古屋工業大学大学院情報工学専攻)

本稿では知的なユーザ支援のための知的 Web を実現するための Web アプリケーション開発技法として、マッシュアップに基づく Web プログラミングについて紹介する。ここでは、マッシュアップに基づく Web プログラミングをマッシュアッププログラミングと呼び、マッシュアッププログラミングの背景となる Web プログラミングの基礎、Web API、および、既存のマッシュアッププログラミング支援ツールについて紹介する。さらに、知的な Web アプリケーションを構築するための、マッシュアッププログラミングの具体例を示すことで、マッシュアッププログラミングにおける課題とその解決方法を説明する。また、マッシュアッププログラミングにおけるセキュリティ上の注意点と対策方法について述べる。最後に、今後のマッシュアッププログラミングに関する展望について紹介する。

知的 Web とマッシュアップ

最近、Web を利用した知的なユーザ支援機能を実現するために、知的 Web 技術と Web プログラミングとしてのマッシュアップ手法が着目されている。知的 Web (Web Intelligence) というキーワードには、明確な定義はないが、その対象とする範囲は、Web2.0 技術やセマンティック Web 技術に関連して多く、最新のテーマを包含している。たとえば、知的 Web の応用分野として、Web サービス、Web 電子商取引、Web エージェント技術、XML、次世代オントロジー、および、情報共有技術などがあり、既存の人工知能応用技術、マルチエージェントシステム、電子商取引技術を融合した新しい分野として確立されつつある。Web に関連した人工知

能の会議の1つとして、IEEE/WIC/ACM International Conference on Web Intelligence がある。この会議は WIC (Web Intelligence Consortium^{☆1})を母体としている。

一般的に、知的なユーザ支援機能の実現に関連して、効果的なユーザインタフェースを実装するためには、その開発に多くの工数が必要とされる。一方、既存の Web ブラウザを利用する Web アプリケーションを導入することにより、このような知的なユーザインタフェースの効率的かつ効果的な実現が可能である。最近では、Web ビジネスを展開する多くの企業により、有用な Web API が数多く公開され、知的な Web アプリケーション (知的 Web) を実現するための効果的なマッシュアップ手法の実現が強く要望されている。

マッシュアップとは複数の Web サービスを組み合わせ、1つの Web サービスを構築することを意味する。開発者が複数の Web サービスを組み合わせるためには、それらの Web サービスを呼び出すための Web API を適切に組み合わせる必要がある。開発者が複数の Web API を組み合わせる最も単純な方法は、複数の Web サービスを同一の Web ページ上で表示するように HTML を編集することである。もしくは、開発者が複数の Web サービスを連携させるには、専用のマッシュアップ開発環境を利用するか、Web サービスを連携させるためのプログラムが必要である。本稿で解説するマッシュアッププログラミングは、Web API を組み合わせて新たな Web サービスを作成する手法である。具体的には、既存のブラウザで機能する JavaScript を用いてマッシュアッププログラミングの概要を説明する。本稿で示すプログラム等は、便宜上、正確さよりも記述の簡潔さを重視しているので、実際のプログラミングでは、公式の仕様に従うよう注意が必要である。

☆1 <http://wi-consortium.org/>

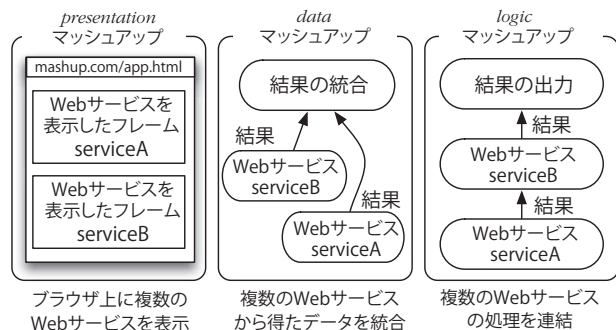


図-1 マッシュアップの分類

マッシュアップの分類

文献3)では、図-1に示すように、マッシュアップを *presentation*, *data* そして *logic* の3種類に分類している。*presentation* マッシュアップでは、同一Webページ内に複数のコンテンツを組み合わせて表示する。iGoogleやMy Yahoo!のようなポータルサイトが例として挙げられる。もっとも単純なマッシュアップで、HTMLの編集でも実現できる。

data マッシュアップでは、より高度で、複数の情報源から得たデータを合成してサービスを提供する。地図上に別のデータを重ね合わせる例が挙げられる。マッシュアップ開発環境を用いることで、プログラムなしでも実現可能である。

logic マッシュアップでは、さらに高度で、複数のWebサービスの入出力を連結させる。一般的に、プログラミングが必要である。旅行支援アプリケーションなどで、最安値のフライトを発見し予約するような、ワークフローを含むアプリケーションが挙げられる。

マッシュアップの実装方法

複数のWebサービスをどこで組み合わせるかという観点からマッシュアップを分類すると、サーバサイドマッシュアップとクライアントサイドマッシュアップの2種類に分類できる。

サーバサイドマッシュアップは、Webサーバ上で実行されるプログラムにより、複数のWebサービスを合成してからブラウザに送信する方法である。サーバ上の計算資源を利用できるため、多様なライブラリやシステムとの連携が容易であり、高度なWebアプリの構築が可能である。その反面、Webサーバへ負荷が集中する欠点がある。

クライアントサイドマッシュアップは、ブラウザ上で動作するプログラム (JavaScript) により、複数のWebサ

ービスを合成する方法である。ここでの処理の利点は、ブラウザ上で実行されるため、サーバへの負荷の集中がない点である。一方、ブラウザ上でのプログラム実行環境に依存するため、パソコン上のブラウザでは可能なことも携帯情報端末上ではできないこともある。JavaScriptは、一般的なコンパイラ言語と比べると高速ではないが、最近のブラウザではJavaScriptの実行速度の高速化が進んでおり、実用的なアプリケーション構築のための有効な手段としてJavaScriptが注目を集めている。

マッシュアップ支援環境

本章では、プログラミングの知識がないユーザがマッシュアップを行うための開発支援環境を紹介する。複数のWebサービスを組み合わせるためには、それぞれのWebサービスの入出力の整合性をチェックする必要があるが、ユーザにとって手間や知識が必要であり、プログラム開発経験のない一般的なユーザには敷居が高い。そのようなユーザでもマッシュアップを可能にする支援環境として、Yahoo!Pipes^{☆2}とMicrosoft Popfly^{☆3}を紹介する。これらの支援環境では、図式化されたWebサービスを、グラフィカルなエディタで編集する機能を提供することで、プログラミング不要のマッシュアップを実現している。

Yahoo!Pipesは、Yahoo!社が提供するマッシュアップ支援環境である。Yahoo!Pipesでは、Web APIの提供する機能が図式化されたモジュールとして提供されている。ユーザは、“Pipes Editor”と呼ばれるグラフィカルなエディタを用いて、使いたい機能を持つモジュールをパイプ^{☆4}で接続することでマッシュアップを行う。ユーザは、プログラムを書く必要がなく、モジュールをドラッグ&ドロップしていただくだけで、マッシュアップを行える。図-2は、Yahoo!Pipesを用いて2つのモジュールをパイプで接続する例である。

Microsoft Popflyは、Microsoft社が提供するマッシュアップ支援環境である。図-3は、その実行画面である。Yahoo!Pipesと同様、グラフィカルなエディタ上でモジュールを配置し、それらのモジュールを接続するだけでマッシュアップが完成する。Popflyで作成したマッシュアップはSilverlight^{☆5}アプリケーションとして実行可能である。

☆2 <http://pipes.yahoo.com/>

☆3 <http://www.popfly.com/>

☆4 Yahoo!Pipesのパイプは、UNIXのパイプと類似している。UNIXでは2つのプログラムpとqを“p|q”のように“| (パイプ)”で接続することで、pの出力をqに入力できる。

☆5 Webブラウザ上でのマルチメディアコンテンツ再生環境。http://www.microsoft.com/silverlight/を参照。

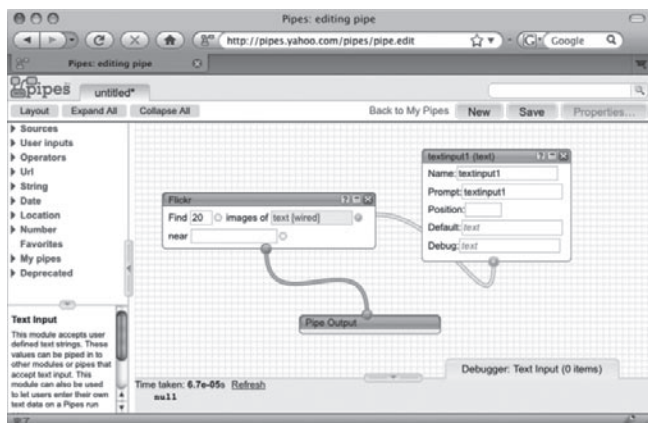


図-2 Yahoo!Pipesの実行情例

これらの支援システムでは、システムに登録済みの Web API のみを利用可能であり、ユーザが任意の Web API を利用することができない。誰でも簡単に Web API を登録できるような仕組みの開発が必要である。

マッシュアップの基礎知識

マッシュアップを始めるための基礎知識として、Web API の探し方、および、簡単な利用方法を紹介する。

適切な Web API の発見

Google, Amazon, そして、Yahoo! などの Web 関連企業が、自社の Web サービスを提供するための手段として、Web API を公開している。開発者は、これらの Web API を使用することにより、インターネット上のさまざまなサービスを自分の Web サイトに組み込むことが可能となる。インターネット上には、Google の提供する地図サービスである Google Maps^{☆6}, Amazon が所有する書籍データにアクセスするための Amazon Web サービス^{☆7}, Yahoo! の日本語テキスト解析のための Web API^{☆8}, そして、YouTube で配信されている動画の情報を取得するための Web API など、多くの Web

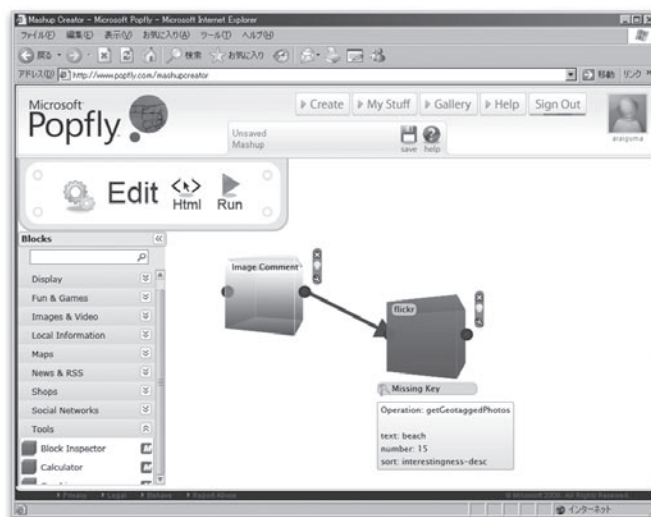


図-3 Microsoft Popflyの実行情例

API が提供されている。

開発者は、必要な機能を提供する Web API を見つける必要がある。ProgrammableWeb^{☆9}は、有名なマッシュアップポータルである。2009年2月現在で1,154個の Web API、および、3,718個のマッシュアップが登録されている。ここでは、Web API のカテゴリ分類や詳細な検索条件が利用可能であり、非常に使いやすい。日本の Web API の検索には、Web API 比較・マッチングサービス^{☆10}がある。ここでは、2つの Web API の情報を比較する機能が提供されている。

Web API では、主に2種類のプロトコル REST (Representational State Transfer) および SOAP (Simple Object Access Protocol) が利用される。公開されている Web API の多くは REST である。SOAP で提供の Web API でも、REST と併用している場合が多い。

REST では、HTTP の GET メソッドを使用して Web API の URL にアクセスすると、結果が取得できる。REST の実行結果は、Web API に依存し、形式として、XML および JSON が一般的に使われている。

SOAP では、HTTP の POST メソッドを利用してクライアント・サーバ間でメッセージ通信を行う^{☆11}。SOAP におけるメッセージは XML で記述される。

Web API の利用

REST 形式の Web API は、ブラウザだけで動作を確認することが可能である。例として、郵便番号検索 API の使用例を示す。ブラウザを起動して、"http://zip.cgis.biz/xml/zip.php?zn=4668555" にアクセスすると、図-4のコンテンツが表示される。この URL の末尾 "zn=4668555" は、名古屋工業大学の郵便番号 466-8555

☆6 <http://code.google.com/intl/ja/apis/maps/>

☆7 <http://aws.amazon.com/>

☆8 <http://developer.yahoo.co.jp/webapi/jlp/>

☆9 <http://www.programmableweb.com/>

☆10 <http://www.api-match.com/>

☆11 SOAP1.0 では HTTP のみであったが、1.1 以降では SMTP, FTP を用いた送受信も可能である。1.2 以降では REST のような GET メソッドを使用した呼び出しも可能となった。


```
<?xml version="1.0" encoding="utf-8" ?>
<ZIP_result>
<result name="ZipSearchXML" />
<result version="1.01" />
<result request_url="http%3A%2F%2Fzip.cgis.biz%2Fxml..." />
<result request_zip_num="4668555" />
:
<value company_kana="ナゴヤコウギョウダイガク" />
<value state="愛知県" />
<value city="名古屋市昭和区" />
<value address="御器所町" />
<value company="名古屋工業大学" />
</ADDRESS_value>
</ZIP_result>
```

図-4 郵便番号検索 API の実行結果

を表し、この要求は「郵便番号が 466-8555 の住所」の問合せとなる。図-4 には、名古屋工業大学の住所に関連した文字列である、「愛知県」「名古屋市昭和区」などの文字列が含まれていることが分かる。本 Web API では、先の“zn=4668555”中の“4668555”の部分のを他の郵便番号に変更すれば、その郵便番号に対応した住所情報を得ることができる。

Web プログラミングの概要

本章では、JavaScript を用いた Web プログラミングについて説明する。最近、デスクトップアプリケーションのような Web アプリケーションが数多く見られるようになった。このような Web アプリケーションは、ブラウザ上で動作するスクリプト言語として JavaScript を利用している。JavaScript を用いて、ブラウザにおけるオブジェクト操作 API である DOM (Document Object Model)^{☆12} を利用することで、ブラウザ上に表示された Web ページを自由に变化させることが可能である。さらに、ブラウザにおける非同期通信のためのオブジェクトである XMLHttpRequest^{☆13} を用いることで、読み込み済みのページ上で、ページ遷移なしにデータを送受信することが可能になる。これらの複合技術は、Ajax と呼ばれ、多くの入門書がある。本稿では、マッシュアップに関連した要素技術の概要を紹介する。

単純な Web アプリケーション

まずは <form> タグを用いた、単純な Web アプリケ

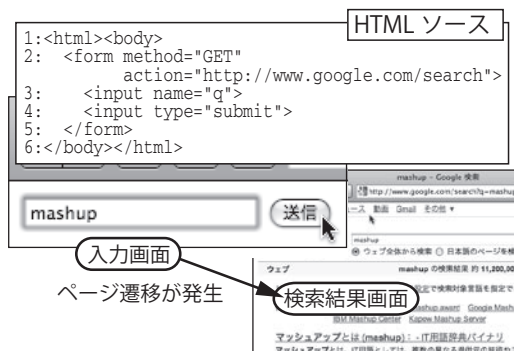


図-5 form を用いた Web アプリケーション

関数名	機能
document.createElement(tag)	要素 tag を生成
x.appendChild(y)	要素 x に要素 y を追加
x.removeChild(y)	要素 x から要素 y を除去
document.getElementById(id)	ID が id である要素を取得

表-1 DOM 操作関数の一部

ーションを示す。この Web アプリケーションは、検索語を入力してボタンを押すと、検索結果を返す。図-5 は、<form> タグを用いた検索システムのための HTML ソースと、その実行画面である。図中の上部の枠で囲まれた HTML ソース中 2～5 行目の <form> タグにより、入力画面で入力された検索語を Google に問い合わせ、検索結果画面を得ている。

図で示すように、本アプリケーションは、このままでは、検索結果の出力にページ遷移を伴い不便である。一方、JavaScript や DOM を有効利用することで、ページ遷移なしに検索結果を出力することが可能になる。次にページ遷移が不要な Web アプリケーションの実装方法を示す。

効果的 Web アプリケーション

ブラウザは、読み込んだ HTML の内部表現として木構造を用いている。DOM は、この木構造を操作するための標準的な API である。DOM には、表-1 で示されるような、関数が標準的に含まれている。

図-6 は、表-1^{☆14} で示した DOM を用いて、ページ遷移なしに検索結果を表示する効果的な Web アプリケーションの実装例である。本アプリケーションも、先に示した単純な Web アプリケーションと同様に、入力さ

☆12 <http://www.w3.org/DOM/>

☆13 <http://www.w3.org/TR/XMLHttpRequest/>

☆14 興味のある読者は removeChild 関数を使って検索結果を消すプログラムを作成してみるとよい。

```

HTML&JavaScript ソース
1:<html><body>
2:<script type="text/javascript">
3:function search() {
4:  var q = document.getElementById("q");
5:  var x = document.createElement("iframe");
6:  x.width = "200"; x.height="250";
7:  x.src =
8:    "http://www.google.com/search?q=" + q.value;
9:}
10:</script>
11:<input id="q">
12:<input onclick="search();"
13:</body></html>

```

① “mashup” と入力して検索



“mashup” の検索結果を
ページ遷移なく表示

② そのまま “javascript” と入力して検索



①の検索結果
はそのまま
②の検索結果
が追加表示

図-6 DOMを用いた Web アプリケーション

れた文字列を検索する Web アプリケーションであるが、ページ遷移が不要である点異なる。

本例では *presentation* マッシュアップの例として、`<iframe>` タグのブラウザ上での表現である `iframe` 要素を JavaScript により生成することで、複数の Web コンテンツをマッシュアップする例を示す。図-6 上部の枠で囲まれた部分が、HTML と JavaScript のソースである。ここでは、12 行目で定義された検索ボタンが押されたときに、3～9 行目で定義された関数 `search` が呼ばれる。`search` では、ID が “q” である要素を取得し (4 行目)、幅が 200 (6 行目) で Google に問い合わせる (7 行目) ための `iframe` 要素を生成し (5～7 行目)、そして、Web ページに `iframe` を追加している (8 行目)^{☆15}。

DOM を利用することで、ページ遷移を伴わない、動作が機敏な Web アプリケーションが実装可能になり、先に示した単純な Web アプリケーションに比べてより良いユーザ体験を実現できる。図-6 の下部は、本アプリケーションの実行例である。本アプリケーションでは、検索語を入力して、検索ボタンを押すたびにページ遷移

なしに検索結果がページ内に追加される。すなわち、図の①と②のようにページ遷移のない、俊敏な検索サービスを提供することが可能である。

実用的マッシュアップのための要素技術

JavaScript の興味深い点は、文字列を式として評価する `eval` 関数である。`eval` 関数は非常に強力な関数であり、後に説明する JSON (JavaScript Object Notation) ^{☆16} は、`eval` 関数を有効利用した例である。`eval` 関数はセキュリティ上の問題の原因にもなり得るので、細心の注意を払って使用する必要がある。

JavaScript から XMLHttpRequest オブジェクト (XHR) を利用することで、ページ遷移することなしにサーバとデータを送受信するプログラムを記述可能になる。XHR は、サーバからのデータを単純な文字列、もしくは、XML 文書として受信する機能を持つ。XHR では、データ読み込みの完了時にコールバック関数が呼ばれるので、非同期な処理が実現できる。

XHR を使ってサーバからデータを送信する形式として JSON が便利である。JSON を用いることで、複雑なデータ構造を持つデータを文字列により送受信することが容易になる。JSON は、JavaScript におけるオブジェクトの表記法に基づくデータ記述言語である。JSON で記述されたデータは、JavaScript の `eval` 関数を用いることで、構文解析などのテキスト処理なしに、読み込むことが可能である。

JSON では、キーと値のペア $key_i, value_i$ をオブジェクトの属性と値として、次のように記述する。

$$\{key_1 : value_1, key_2 : value_2, \dots\}$$

key_i は文字列を取り、 $value_i$ は数値、文字列、真理値、オブジェクト、または、配列などの値を取る。

次に JSON の記述例と操作例を示す。

```

var json = '{"name": "Sano", "age": 20}';
var obj = eval("(" + json + ")");
obj.name = "Goto";

```

変数 `json` には、JSON 形式の文字列を代入している。このオブジェクトは、キーと値のペアとして、`name` と `Sano` および `age` と `20` を持つ。`eval` 関数により JSON 形式で記述された文字列が評価され、得られたオブジェクトが、変数 `obj` に代入される。キーの持つ値へのアクセスは、オブジェクトの属性へのアクセスとして記述すればよい。たとえば、キー `name` の値の変更は、`obj.name = "Goto"` のように記述する。

☆15 4, 5 行目の `var` は変数宣言を表し、6 行目の `x.width` はオブジェクト `x` の属性 `width` を表す。

☆16 <http://www.ietf.org/rfc/rfc4627.txt>

タグ	メソッド	利用法
<iframe>	GET, POST	非表示状態で生成する。
	GET	画像をサーバに要求すると、画像の読み込み後に、onload イベントが発生するので、これにより受信完了を判定する。サーバは送信したいコンテンツをクッキーとして送り、ブラウザでは JavaScript でクッキーを読むことで送受信する。クッキーのデータ量の制約により送信可能なデータ量も制限される。
<script>	GET	動的に生成しスクリプトをサーバから読み込むことで送受信する。スクリプトの読み込み後、JavaScript が実行されることで受信完了を判定する。

表-2 クロスサイト対応のタグ

マッシュアップにおけるセキュリティ

マッシュアップによりアプリケーションを簡便に作成できる反面、多様な Web サービスを組み合わせることにはセキュリティ上の問題が含まれる可能性がある。マッシュアップで利用している Web サービス中に、悪意のあるコンテンツ、もしくは、何らかの脆弱性のあるコンテンツが含まれていると、他の Web サービスが提供するコンテンツに対して攻撃することが可能になる。たとえば、重要な情報を盗んだり、悪意のあるサイトに誘導したりすることも可能である。

ブラウザのセキュリティモデル

ブラウザにおける従来のセキュリティモデルである同一生成元ポリシー（Same origin policy）では、異なるドメイン間^{☆17}のコンテンツ間でのインタラクションを禁止していた。XMLHttpRequest にも同一生成元ポリシーが適用されている。このため、マッシュアップのように、複数のドメインから得られたコンテンツを合成する場合に問題があった。

同一生成元ポリシー下で複数のドメインから得られたコンテンツのマッシュアップを実現するために、表-2 に示される <iframe> タグ、 タグ、もしくは、<script> タグが利用可能である。同一生成元ポリシー下でも、これらのタグにより異なるドメインのコンテンツを読み込むことが可能（クロスサイトが可能）である。これらのタグにより、複数の Web サービスを

同一 Web ページ上で組み合わせることが可能になる。一方、Web サービス間で互いを保護する仕組みがないため、セキュリティ上の問題が発生する可能性が残っている。HTML の次バージョンである HTML5^{☆18}では、異なるドメイン間のコンテンツを安全に扱うための仕組みが検討されているが、現状では、プログラマが脆弱性をなくすための注意を払う必要がある。

クロスサイトスクリプティングへの対策

クロスサイトスクリプティング（XSS : cross-site scripting）とは、悪意を持ったユーザの入力に含まれるスクリプトが別のユーザの閲覧するコンテンツ上に表示され、実行される攻撃手法である²⁾。XSS への対策としては入力データのフィルタリングが一般的である。Web サービスにより取得した情報の安全性は、一般的には保証されない。たとえば、Web サービスから得られた XML に、個人情報を盗聴するスクリプトが埋め込まれている可能性もある。これは、Web サービス提供者が悪意を持っている場合だけでなく、悪意を持った第三者の攻撃により悪意のあるスクリプトがデータに埋め込まれる可能性もある。そのため、大手企業が提供している Web サービスであっても、常に注意を払った方が好ましい。また、データ中の危険性のあるプログラムに対しては無効化や削除などといった対策を行う必要がある。

外部から情報を入力する方法として、HTML フォームもあるが、この際には入力された値に対してサニタイジングを行うことが一般的である。同じように、Web API の情報に関してもサニタイジングを行うことが重要である。ここでのサニタイジングとは、入力データからシステムに影響を与えそうな HTML タグ、JavaScript などのプログラム、SQL 文などを変換もしくは除去することである。

iframe を用いた安全なマッシュアップ

安全なマッシュアップの実現手法を示す。ここでは、文献 4) に基づき、iframe を利用した Web サービス間の分離と、iframe 間通信を用いたブラウザ上での Web サービス間の連携の実現について紹介する。

<iframe> タグを用いることで、安全なマッシュアップを実現できる。<iframe> タグにより異なるドメインのコンテンツを同一 Web ページ上に読み込むことが可能になる。同一生成元ポリシーにより、同一ドメインのコンテンツを含む iframe 間の干渉は可能であるが、異なるドメインのコンテンツを含む iframe 間の干渉は制限されている。たとえば、互いの DOM や変数の読み書きもできないので、Web サービス間を互いに保護す

☆17 厳密には、ホスト名と URI スキームの対であり、ポート番号を含む場合もある。本稿では、単純にドメインと呼ぶ。

☆18 <http://dev.w3.org/html5/spec/>

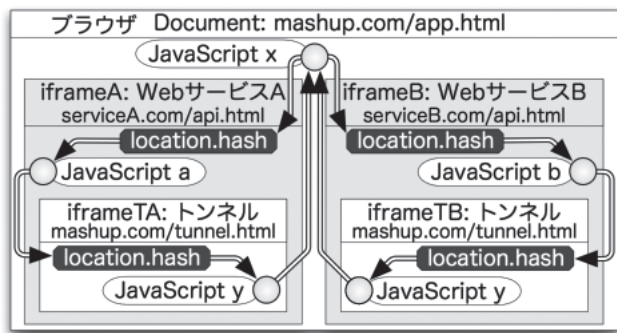


図-7 location属性を利用したiframe間通信

ることが可能である。

iframeにより分離されたWebサービス間での連携手法について、Burkeが紹介した^{☆19}、location属性の“フラグメント識別子 (fragment identifier)”を利用した、ブラウザ上でのiframe間通信を紹介する^{☆20}。フラグメント識別子とは、Webページ内の特定の部分 (フラグメントと呼ぶ) を指定するための識別子であり、“#identifier”の形式で記述される^{☆21}。iframe間通信を実現したJavaScriptライブラリとして、たとえば、Dojo^{☆22}のIFrame Proxyが利用可能である。

フラグメント識別子を表すlocation.hash属性への書き込みによりiframe間通信を実現することが可能である。iframeは、コンテンツのURLを表す属性であるlocation属性を持つ。iframeは、location属性に代入にされたURLを新たなコンテンツとして読み込む機能を持つ。location属性は他の属性とは異なり、他ドメインのiframeからでも書き込みが可能であるという興味深い特徴がある^{☆23}。location属性への書き込みに伴うWebページの読み込みを回避するために、location.hash属性への書き込みを用いる。フラグメント属性はページ内の場所を示すので、location.hash属性のみの変更はWebページの読み込みを伴わない。

図-7はiframe間通信の実現例である。図では、mashup.com/app.html (Document) に、異なるドメインのWebサービス serviceA.com/api.html (iframeA) およ

び serviceB.com/api.html (iframeB) をiframeとしてマッシュアップしている。図中の丸はJavaScriptを表し、JavaScript a, b, x, yは、図中で対応するフレーム内に含まれるJavaScriptである。矢印はiframe間通信に伴うデータの流れを表す。

iframe間通信のための特別なiframe (トンネルと呼ぶ) として mashup.com/tunnel.html (iframeTA および iframeTB) を用いる。iframe Pがiframe Qを含むとき、PをQの親と呼び、PはQのlocation属性に書き込むことが可能である。図-7で示されるように、iframeAおよびiframeBが、トンネルiframeTAおよびiframeTBの親となることで、JavaScript aおよびbはトンネルのlocation属性に書き込むことが可能になる。aおよびbからトンネルのlocation.hash属性への書き込みを、メッセージの送信とする。

JavaScript xとyは同一ドメインでありデータ交換が可能なので、yがJavaScript aおよびbから受信したメッセージをxに渡す。DocumentはiframeAおよびiframeBの親なので、xはこれらのiframeのlocation属性に書き込むことが可能である。xがメッセージをiframeAおよびiframeBのlocation.hash属性に書き込むことで、aおよびbはメッセージを受信できる。location.hashを用いることで、サーバに依存せずブラウザだけでiframe間の通信が可能になる。

知的 Web のための マッシュアッププログラミング例

ここでは、知的Webを実現するための簡単なマッシュアッププログラミングについて説明する。ブラウザ上でのマッシュアッププログラミングと、サーバ上でのマッシュアッププログラミングについてそれぞれ説明する。

クライアントサイドマッシュアップ例

ブラウザ上で地図をマッシュアップする例を示す。地図サービスとしてGoogle Maps API^{☆24}を用いる。図-8は、Google Maps APIの使用例である^{☆25}。

Google Maps APIでは、緯度と経度で指定した場所の地図を表示することが可能である。緯度と経度は図-8の8行目のように、GLatLng型によって表現する。ここでは、緯度が“34.659546”で、経度が“135.005665”である。9行目では、地図を生成している。地図は6行目の<div>タグの位置に生成される。10行目のsetCenter()関数によって、8行目で定義した座標の地図が表示される。ここで、第1引数は座標であり、第2引数は地図の倍率である。Google MapsのAPIをJavaScriptから利用することで、地図を表示するだけで

☆19 <http://tagneto.blogspot.com/2006/06/cross-domainframe-communication-with.html>

☆20 window.name を利用した方法もある。 <http://www.sitepen.com/blog/2008/07/22/windowname-transport/> 参照。

☆21 <http://domain.com/search?q=query#test> の #test 部分。

☆22 <http://www.dojotoolkit.org/>

☆23 ただし、読み込みはできない。

☆24 <http://code.google.com/intl/ja/apis/maps/>

☆25 Google Maps API を利用するためには API キーが必要である。本プログラム中では2行目の「KEY」がAPIキーを表す。

```

1: <head>
2: <script src="http://maps.google.com/maps?file=api&v=2&key=KEY" type="text/javascript">
3: </script>
4: </head>
5: <body>
6: <div id="map" style="width:640px;height:480px"/>
7: <script type="text/javascript">
8: p=new GLatLng(34.659546,135.005665);
9: m=new GMap2(document.getElementById("map"));
10: m.setCenter(p,13);
11: </script>
12: </body>

```

図-8 Google Maps の使用例

```

1: <?xml version="1.0" encoding="UTF-8" ?>
2: <result>
3: <version>1.1</version>
4: <address>東京タワー</address>
5: <coordinate>
6: <lat>35.658587</lat>
7: <lng>139.745425</lng>
8: <lat_dms>35,39,30.913</lat_dms>
9: <lng_dms>139,44,43.53</lng_dms>
10: </coordinate>
11: <url>http://www.geocoding.jp/?q=東京タワー</url>
12: <needs_to_verify>no</needs_to_verify>
13: <google_maps>東京タワー</google_maps>
14: </result>

```

図-10 Geocoding API の返り値

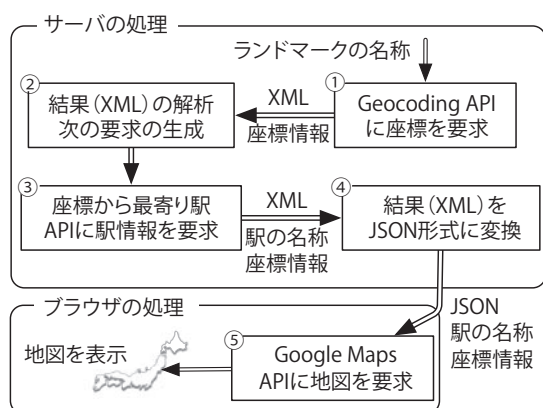


図-9 「最寄り駅マップ」におけるマッシュアップ

なく、地図上に線分を描画したり、ラベルを表示することが可能になり、他のサイトから取得したデータなどを地図上に重ねて表示することができる。

サーバサイドマッシュアップ例

地図と駅の情報を組み合わせた「最寄り駅マップ」をマッシュアップを利用して作成する例を示す。図-9は、本例で使用するWebサービスと、それらの接続関係を表している。図中の①～⑤は実行順序を示しており、①、③、⑤でWebサービスにアクセスしている。

本例におけるマッシュアップでは、ランドマークから位置を検索するサービスと位置から駅の情報を検索するサービスをlogicマッシュアップする例と、駅の情報と地図をdataマッシュアップする例を示す。ランドマークの緯度・経度を取得するためのサービスとして、Geocoding API^{☆26}を用いる。これは、住所やランドマークの名称などから、その地点の緯度・経度を返す

Web APIである。最寄り駅の情報を取得するためのサービスとして、座標から最寄り駅API^{☆27}を用いる。これは、ある地点の緯度・経度を与えると、その周囲の駅情報を返すWeb APIである。地図はGoogle Maps APIを用いて取得する。ここでは、Google Maps APIに渡すための駅の座標データを求めるために、Geocoding APIと座標から最寄り駅APIからXML形式で返されたデータをWebサーバ上で処理する。

Geocoding APIと座標から最寄り駅APIは、REST形式のWeb APIである。ユーザが「東京タワー」を入力としてシステムに与えたとする。Geocoding APIで「東京タワー」の座標を得るために、サーバは次のURLにRESTでアクセスする。

<http://www.geocoding.jp/api/?v=1.1&q=東京タワー>

結果として、図-10で示すXML文書が得られる。<lat>タグと<lng>タグにより、それぞれ緯度と経度が分かる。サーバでは、得られたXML文章を解析し座標データを得る。その後、サーバは、座標データを座標から最寄り駅APIに与え、結果として、最寄りの駅の一覧をXML形式で取得する。本XMLには駅名や駅の座標データが含まれており、このデータをGoogle Mapsに渡すことが次の目標である。今回は、Google MapsをJavaScriptを用いてブラウザ上で制御したいので、座標データをJavaScriptプログラムに渡す必要がある。サーバでは座標データをJSON形式で記述し、クライアントに送信する。クライアントは、JSON形式の座標デー

☆26 <http://www.geocoding.jp/api/>

☆27 http://www.ekidata.jp/tools/api_station_c.html



図-11 最寄駅マップの実行例

タに基づき、図-11のようにJavaScriptを用いてGoogle Maps上にマーカーを設置する。

知的なマッシュアップ支援に向けて

マッシュアッププログラミングにおける課題について述べる。ここでは、Web APIの発見および利用における課題について説明する。

Web APIの発見

開発者が、要求に適したWeb APIを探すときに、汎用の検索エンジンは適していない場合があり、Web APIを集めたポータルサイトを利用するとよい。たとえば、汎用の検索エンジンは、組合せを考慮したWeb APIの検索については、不十分である。マッシュアップにおいて、Web APIの組合せを考慮する必要がある。組合せ可能なWeb APIを探すには、Web API同士の整合性についても考える必要があり、手間と時間がかかる。Web APIを組み合わせるには、組み合わせるべきWeb APIの入出力が一致していなければならない。たとえば、先の例題では、Geocoding APIと座標から最寄駅APIを組み合わせたが、このときは、Geocoding APIの出力である緯度と経度を座標から最寄駅APIが入力値として扱えたので、組み合わせることができた。この組合せ可否の判定は人間には可能であるが、自動化は困難である。なぜならば、Web APIから出力されたXMLの形式が統一されていないからである。たとえば、緯度を表す要素名に関しても、“latitude”や“lat”のような表現の揺れがある。

Web APIは日々増え続けており、類似した機能を持つWeb APIが複数個存在する場合もある。機能が類似

したWeb APIが複数種類ある場合に、より性能の高いWeb APIを選択したい。仕様に基づき、最適なWeb APIを選ぶには時間がかかる。最適なWeb APIを選択するための情報を記述するための仕様や、それらの情報を管理するためのリポジトリが求められる。

Web APIのバージョンアップやサービス中止に追従する必要もある。使用中のWeb APIが、仕様変更やサービス停止などの理由により、ある日突然使えなくなったり、新たな機能が追加されることは珍しくない。このような、Web APIに関する情報の更新に追従するための適切な手段が存在しないので、何らかの支援が必要である。

今後、UDDI (Universal Description, Discovery and Integration) や WSDL (Web Services Description Language) の普及により、自動的に連携可能なWebサービスが発見可能になることが期待される。UDDIは、インターネット上で利用可能なWebサービスを調べるディレクトリである。WSDLは、XMLによりWebサービスを記述するための言語である。WebサービスにセマンティックWebの技術を用いるセマンティックWebサービス¹⁾も研究されている。

Web APIの利用

マッシュアッププログラミングにおいて、Web APIに与えるべきパラメータをプログラムとして実装する作業には手間がかかる。Web APIによっては、100個以上の入出力パラメータを持ち、必要十分な入出力パラメータについて理解するには時間がかかることもある。アプリケーションによっては、すべてのパラメータが必須ではないが、複雑なパラメータの組合せを必要とする場合もある。また、入力パラメータの種類によって出力パラメータが変化するため場合も想定する必要がある。WSDLのような共通的な記述方式が一般的になれば、プログラムによってWeb APIを使用するためのプログラムを自動的に生成できるようになり、利便性や開発効率が高まると期待される。

マッシュアッププログラミングは、試行錯誤的になりがちである。たとえば、Web APIの仕様が記載された文書やWebページとプログラムを見比べながら実装したり、出力値を理解するために実際にWeb APIを試すような作業を、何度も行う必要がある。Web APIによって、XMLの形式がRSS形式であったり、Atom形式であったりと異なっており、名前空間の扱いなどを区別する必要がある。また、REST、SOAP、および、XML-RPCなどさまざまなプロトコルに合わせたプログラムの開発も手間がかかる。

マッシュアップアプリケーションを効率よく動作さ

せるためには、組み合わせる Web API を適切な順序で実行した方がよい。たとえば、依存関係のない Web API 同士を並列に実行することが可能である。利用する Web API の実行速度は、その Web API の提供元に大きく依存する。複数の Web API の最適な実行順序を決定するために、事前のサンプリングによって、その実行速度を見積もる必要もある。使用する Web API の種類が少ない場合は、手動による最適化も可能である。Web API の実行が保証されない環境において、多数の Web API の依存関係を把握しつつ、実行順序を最適に決定するのは容易ではない。

利用制限が設定された Web API も存在する。たとえば、1日のリクエスト数や、リクエストを行う最小限必要な時間間隔が設定されている場合が多い。複数の Web API を利用する場合は、使用している Web API の中で許可されたリクエスト数が最も少なく、可能なリクエスト間隔が長いものに合わせて、全体を調整する必要がある。1つの Web API が使用できなくなると、システム全体が動かなくなるような実装は避けなければならない。

今後の展望

Web API の充実と、それらの情報を統合的に管理する仕組みの普及が求められる。マッシュアップをより簡便にするための HTML の新たな仕様が普及が期待される。また、簡単に安全なマッシュアップを実現するための、システムの実現も望まれる。

知的なプログラムであるエージェント技術により Web API の抽象化が進めば、エージェント間の連携に

基づくマッシュアップが可能になり、より高度で知的な Web アプリケーションをプログラミングなしに実現できる。Web API に関するマッチメイキングのような、エージェントに基づくマッシュアップに関して今後の研究の発展が期待される。

マッシュアップにより誰でも簡単にアイデアを実現できるようになることで、情報技術により、安全で快適な世の中が実現できるようになることを期待する。

参考文献

- 1) McIlraith, S. A. et al. : Semantic Web Services, IEEE Intelligent Systems, pp.46-53 (2001).
- 2) 吉濱, 石田, 浦本 : Web2.0 アプリケーションにおける代表的な攻撃手法とその対策, 情報処理, Vol.50, No.1, pp.44-54 (Jan. 2009).
- 3) Dornan, A. : Mashup Basics : Three for the Money, Network Computing, <http://www.networkcomputing.com/showitem.jhtml?articleID=201804223> (2007).
- 4) Keukelaere, F. D. et al. : SMash : Secure Component Model for Cross-Domain Mashups on Unmodified Browsers, WWW2008, pp.535-544 (2008). (平成 21 年 3 月 2 日受付)

新谷虎松(正会員)

tora@nitech.ac.jp

1982 年富士通(株)国際情報社会科学研究所研究員。1993 年名古屋工業大学、現在、同大学院情報工学専攻教授。Web エージェントの研究に従事。2004 年(株)ウィズダムウェブ設立創業者代表取締役。東京理科大学大学院修士修了。博士(工学)。本会フェロー。

大園忠親(正会員)

ozono@nitech.ac.jp

2000 年名古屋工業大学助手。現在、同大学院情報工学専攻准教授。2004 年(株)ウィズダムウェブ設立創業者最高技術責任者。名古屋工業大学大学院博士修了。博士(工学)。Wisdom Web の実現を目指した知的情報処理技術の研究に従事。

