JavaScript を用いた Web アプリケーション開発支援の研究
– HTML 要素と通信データの対応関係を用いた記述方法の提案 –

2007MI112 近藤 友紀　　2007MI147 村松 雅大
指導教員　蜂巣 吉成

1　はじめに

近年，Web アプリケーションにおいて，操作性が向上した RIA(Rich Internet Application) が注目されている．JavaScript を用いた RIA は一般的に，文書構造を HTML，見栄えを CSS，振舞いを JavaScript と，分離して記述することが推奨されており，jQuery を用いて分離することが可能である [1]．また，分離して記述することで，文書構造とプログラムの開発の分業や可続性の向上，記述の変更が容易といった利点がある．

RIA の通信は JavaScript で記述されており，JavaScript を用いた RIA の開発では，Ajax 技術を利用して JavaScript プログラムが実行時に通信データを取得し，ブラウザにおける HTML の内部表現である DOM 木などを変更する．取得したデータをブラウザの画面上に反映するので，HTML 要素と通信で取得したデータの対応関係をとり，動的に HTML 要素を作成し，ブラウザに反映する必要がある．jQuery では，これらの記述が混在しており，開発が困難である．また，複数の異なる Web アプリケーションの開発において，取得データをブラウザに反映する処理では，類似した記述をおこなう必要がある．

本研究の目的は，Web アプリケーションにおける通信処理の開発支援である．通信で取得したデータと HTML 要素の対応関係と，ブラウザに反映する処理を分離して記述する方法を提案する．また，複数の Web アプリケーションを開発する際に，類似した処理記述が必要となるブラウザへの反映処理の開発支援をおこなう．混在した記述に対して，振舞い部分の HTML 要素と取得したデータの対応関係を用いることで分離をおこなう．類似した処理をライブラリ関数として用意することで開発者の記述量を削減させる．通信におけるエラー処理においては，取得したエラーの種類とエラー処理の対応関係を記述し，変換器を用いてコードを生成することで記述量を削減させる．

本研究では JavaScript を用いたページ遷移を伴わない Web アプリケーションの Ajax 技術を利用した通信を対象とし，取得するデータの形式として JSON 形式を使用する．

2　Web アプリケーション開発の問題点

本研究は Web アプリケーションの通信処理に対して，開発支援をおこなう．通信で取得したデータをブラウザ上に反映する際に，文書構造の HTML 要素に対する取得したデータの対応関係と，ブラウザへの反映処理が混在して記述されている．また，ブラウザへの反映処理では，複数の Web アプリケーションの開発時に，ブラウザへ反映する内容は同じだが記述の異なる類似した記述をする必要がある．**図 1** では，通信処理の流れの中で，混在した記述と類似した処理を行っている部分を示す [2]．

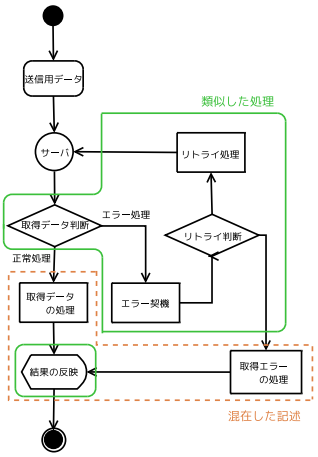

図 1　通信の流れ

2.1　振舞いと文書構造の混在

Web アプリケーションの振舞い部分では，**図 1** で示した通り，取得したデータと結果の表示において，混在した記述をおこなう必要がある．取得したデータの処理として，JavaScript プログラム内に文書構造の HTML 要素の属性名・内容と取得したデータと，ブラウザへの反映処理が依存しているので，混在して記述されている．

```
a = $("#tmp_id").clone(true);
str[n] ="tmp_id"+ n;
a.attr("id",str[n]);
a.find("#avatar").attr("id","avatar"+n);
a.find("#avatar"+n).attr({
  'href': 'http://sample_code.com/{from_user}'
          .replace(/{from_user}/, this.from_user),
  'target': '_blank'
});
  ....      // 複数のデータを記述
$("#TMP").append(a);
n++;
```

図 2　混在した記述例

図 2 の下線部に，通信の結果をブラウザ上に反映させる Web アプリケーションの JavaScript 内で混在した記述の例を示す．文書構造の HTML 要素の id に，取得したデータと表示する処理が同時に記述されており，一つの HTML 要素に複数のメソッドが混在して記述され開発に手間がかかる．

2.2 類似した処理の記述

Web アプリケーションの通信では，取得したデータをブラウザ上に反映する処理をおこなうが，Web アプリケーションごとに取得するデータが異なり，反映する内容をそれぞれを記述しなければならない．

図 3 ではブラウザへの反映処理で，反映する処理方法は類似しているが異なる記述をする例として，2 つの Web アプリケーションをあげる．双方とも，文書構造の HTML 要素の複製し，ブラウザ上に反映する処理をおこなう．**図 3** の (a) は，文書構造の HTML 要素内のデータの削除を，(b) は HTML 要素の複製を，(c) は取得したデータを文書構造の HTML 要素に類似した挿入する処理をおこなう記述である．

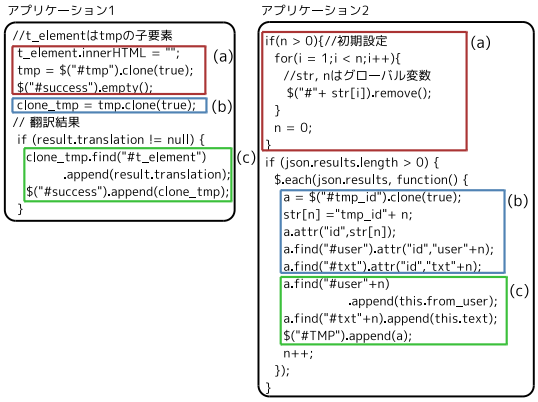

図 3　類似した記述

3　通信処理の記述方法の提案

2 節の問題点から JavaScript プログラム内の振舞い部分のデータを反映する処理と，文書構造の HTML 要素と取得したデータの混在を解消し，パターン化することでブラウザに反映する処理を簡潔に記述できる方法を提案する．

本研究では，文書構造の HTML と取得したデータの対応関係を連想配列で示し，データの反映処理をライブラリ関数として提供する．ライブラリ関数では引数として，対応関係，実際に取得したデータ，テンプレートの id をとる．また，使用するエラーの種類と処理を開発者が簡潔に記述し，変換器を用いることで記述量を削減する．具体的な記述方法は 3.3 節で示す．

図 4 に本研究を用いた Web アプリケーションの開発方法を示す．本研究では，文書構造の HTML に通信結果を表示する際のテンプレートとなる HTML 要素を記述し，id を指定する．また，HTML 要素と取得データの対応関係を JavaScript 内で記述し，ブラウザに反映する処理をおこなうライブラリ関数を用いることで，結果を表示する．なお本研究では，通信結果を表示するための HTML 要素をテンプレートと呼ぶ．テンプレートでは内容や属性が書かれていないが，対応関係を用いて反映処理をおこなうライブラリ関数によって取得データが指定される．またテンプレートに取得データを指定する前は，style を hidden にすることで非表示としておく．エラー処理については，開発者はエラーの種類とその処理を記述し，変換器によりコードを生成する．

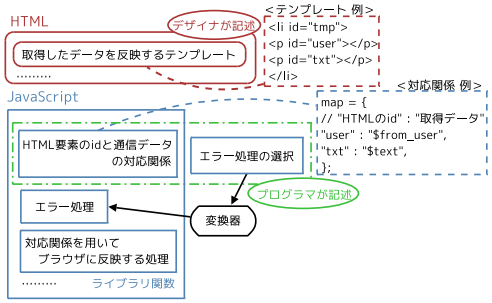

図 4　本研究を用いた Web アプリケーションの開発方法

3.1 混在した記述に対しての開発支援方法

本研究では，2.1 節で記述した問題点と，取得データをブラウザに反映する処理において，データの反映処理と，HTML 要素と取得したデータの対応関係を明示的に記述して分離する．

2.1 節で記述した問題点である混在した記述における取得データの処理は，**図 1** で示すように，正常処理とエラー処理に分けられる．正常処理では，通信本来の機能を実現する処理をおこない，エラー処理では，付随したそれぞれのエラー処理をおこなう [2]．正常処理とエラー処理のそれぞれに対して，対応付けをおこなう必要がある．文書構造の HTML 要素と取得データの対応関係を用いて，反映するデータを作成し，ブラウザへの反映処理との対応関係を記述することで混在を解消する．

3.1.1 通信処理の対応関係

本研究では，プログラマが HTML 要素と JSON 形式で取得したデータの対応関係を連想配列として記述する．また，通信で取得したデータを表示する HTML 要素には，id をつけ，対応関係で指定する．この連想配列を，反映処理を実現したライブラリ関数の実引数で用いる．ブラウザへの反映処理における HTML 要素と取得データの対応関係では，HTML 要素でのデータの種類，取得データに対する処理によって複数の対応関係が考えられる．**図 5** に文書構造の HTML 要素と，取得データの対応付けの記述方法を示す．

- HTML 要素の内容として，データを対応付け
- HTML 要素の属性値として，データを対応付け

取得データを HTML 要素の属性値として反映する際，HTML 要素の id と属性名が必要となる．本研究では，HTML 要素の id と属性名を示す際，**図 5** で記述したように「@」を加える．

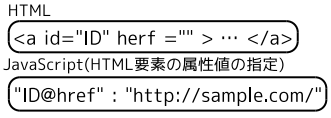

図 5　HTML 要素の属性値の指定

HTML 要素に対応させるデータにおいて，HTML 要素へ反映する際の処理を示す．

- 取得データを直接, HTML 要素に反映する場合
- 取得データを処理した後に，HTML 要素に反映する場合

上記で示したように，HTML 要素に対応させるデータでは，取得したデータを直接反映させる場合とデータに対して何らかの処理をした後に，反映する場合がある．また，JSON 形式で取得したデータのキー名に関しても示す必要がある．本研究では，**図 6** で記述したように，取得したデータに対して何らかの処理を行う際，処理内容を「${}」で囲み，キー名の場合は「$」をキー名の前に記述することで取得データの判別をおこなう．

取得したデータを処理してHTML要素に反映する際の記述方法

```
id : "${'http://sample.com/{from_data}'
                          .replace(/{from_data}/,data)}"
```

取得したデータのキー名の記述方法

```
ID : " $key_name "
```

図 6　HTML 要素へ反映する際の処理

3.2　ブラウザに反映する処理

3.1 節で示した提案により，Web アプリケーションごとに異なる取得データの処理とブラウザへの反映処理を分離した．分離したことにより，類似した処理であるデータをブラウザに反映する処理を関数として事前に用意することが可能となる．プログラマは，連想配列で示した対応関係，実際に取得したデータ，デザイナが示した文書構造内のテンプレートの id を引数として，ブラウザ上にデータを反映する処理の関数を呼び出すことで，ブラウザへの反映が可能となり，記述量が削減できる．

ブラウザに反映する処理を行う関数では，文書構造内のテンプレートを複製し，連想配列で示されている対応関係をもとに，通信で取得したデータを反映する DOM 要素を生成する．その後，生成した DOM 要素を文書構造内のテンプレートの兄弟要素として反映している．3.1.1 節で示した複数の対応関係に対しても，記述形式に合わせた処理をおこなう．

3.3　エラー処理の対応関係

エラー処理に対しても開発支援方法を提案する．**図 1** で示す通り，エラー処理にも，混在した記述と類似した記述がある．本節では，エラー処理の分析をおこない，エラー処理においての対応関係の記述方法を提案する．

エラー処理は下記の 4 つに分類することができる [2]．

1. エラー契機：エラー処理の契機となる処理
2. リトライ判断：リトライを続けるかどうかを判断する処理
3. リトライ処理：エラー発生後，リトライする際におこなう処理
4. 取得エラーの処理：エラー発生後，データを表示する際におこなう処理

エラー契機では，サーバから取得したデータがエラーである場合，どのような処理をおこなうかを判断する．リトライ判断とリトライ処理では，再度サーバに接続するかの判断と処理をおこなう．分類したエラー処理から，複数の Web アプリケーションを開発する際に，類似した記述が存在する．エラーメッセージは，通信データで取得したエラーメッセージを表示する必要があり，リトライ処理は，サーバへの再接続を Web アプリケーション開発時に記述する必要がある．また，用途が異なる Web アプリケーションの開発においても，エラーメッセージの表示とリトライ処理を記述する必要があることから，再利用可能な類似した記述である．

通信で取得したエラーの種類において，ブラウザに反映する内容を示す．

- 典型的なエラーメッセージを反映する場合
- 独自のメッセージを反映する場合
- 独自のエラー処理をおこなう場合

上記に示したように，エラーの種類に対して，典型的なメッセージの反映，独自のメッセージの反映，独自のエラー処理を記述し，これを変換器を用いて，実際のコードを生成し，開発者の記述量を削減する．

開発者は，通信で取得したエラー内容とエラーの表示方法を記述する．表示方法として，典型的なエラーメッセージの表示は**図 7** のように，「msg」を記述することで行えるようにした．また，リトライ処理に対しても Timeout を用いることで，回数を指定して処理を行えるようにした．エラーメッセージとリトライ処理以外を記述したい場合は**図 7** のように，「${}」にコードを記述する．

エラー処理の記述例

```
error{
  Timeout(5) & msg;
  Not_Found & msg;
  Unauthorized & ${ alert("エラーメッセージ") };
}
```

変換後の記述例

```
if(xOptions.status==408){
 msg = "408: Request is Request Time-out."
 timeout(tw(),5); //twは通信をおこなう関数
}else if(xOptions.status==404){
  msg = "404: Request is Not Found";
}else if(xOptions.status==401){
  alert("エラーメッセージ");
}
```

図 7　エラー処理の記述例

4　実装と評価

提案方法を用いて，実装と評価をおこなう．

4.1　実装

開発支援に関しては, 開発者が HTML 要素と表示させるデータの対応関係を示した連想配列を用いて HTML に表示をおこなう関数と追加した DOM 要素を削除する関数の 2 つをライブラリ関数として用意した．また，ライブラリ関数内で 6 つの関数を用い，全体で 300 行程度となった．

4.2 評価

実際に使用されている Web アプリケーションに，作成した変換器を適用し，記述量の削減と HTML と JavaScript プログラムの分離し，混在した記述の解消が可能であるか評価をおこなった．

4.2.1 振舞いと文書構造の分離

評価では，短い投稿を入力して記事を共有する Web サービスの一つである twitter 内の，投稿者を検索する Web アプリケーションを用いる．この Web アプリケーションは，ブラウザに反映をおこなう部分において，HTML 要素と JavaScript が分離できていない．

本研究では，jQuery を用いることで，上記で示した Web アプリケーションの JavaScript プログラム内の HTML 要素の分離をおこなった．2.1 節のデータを反映する処理と取得データの混在した記述に対しては，開発者が記述した対応関係を用いて反映する処理をおこなうことで，混在した記述を解消することができた．**図 8** に本研究の開発支援を適用し，混在する記述の解消を行なった例を示す．

JavaScript(対応関係を用いた例)

```
default_state();    //データの削除
//取得データと文書構造内のHTML要素の対応関係
map = {
 "user" : "$from_user",
 "avatar@href" : "${'http://sample.com/{from_user}'
                    .replace(/{from_user}/,$from_user);}",
 …
 };
/* 取得したデータを対応関係を用いてブラウザに反映する関数 */
/* changehtml(テンプレートid,対応関係を示した連想配列,取得データ) */
changehtml(tmp_id,map,data);
…
```

図 8　本研究の開発支援を適用したコード

従来の Web アプリケーション開発では，開発後に変更する場合，取得したデータと文書構造の HTML 要素を JavaScript プログラムで追加，変更や，ブラウザへの反映処理の変更をおこなう必要がある．本研究で提案した記述方法を用いることで，対応関係を示す JavaScript プログラムの変更のみで追加，変更をすることができる．

4.2.2 類似した処理の削減

評価は，3 つの Web アプリケーションでおこなった．上記で示したアプリケーション (a) と入力した文章を翻訳し結果を表示するアプリケーション (b) と，アカウント検索をおこなうアプリケーション (c) の 3 つである．

本研究を適用したときの記述量の変化を**表 1** に示す．本研究の開発支援を適用した場合，HTML 要素とデータの対応関係を，関数の引数とした連想配列を用いて示す必要がある．行数での記述量の評価が困難であり，本研究では，評価として開発者が記述するメソッド数の変化に着目した．

表 1　記述量の変化

	取得データ処理	対応関係	メソッド数
(a)	40 行→ 30 行	20 行	25 個→ 10 個
(b)	8 行→ 5 行	3 行	7 個→ 2 個
(c)	39 行→ 20 行	13 行	23 個→ 6 個

表 1 では，対応関係を用いることで混在した処理を解消し，ブラウザへの反映処理をライブラリ関数として呼び出すことで，開発者が呼び出し記述するメソッド数を削減している．本研究では，より簡潔に記述することが可能となり，JavaScript プログラムの記述量を削減することができた．また，Web アプリケーションを開発する際に HTML 要素とデータの対応関係を示すことで，ライブラリ関数部分の再利用が可能となった．

4.2.3 JavaScript ライブラリとの比較

JavaScript ライブラリである jQuery を用いることで，HTML と JavaScript を分離できるが，取得したデータを反映する場合などにおいて記述が混在してしまう．また，JavaScript 用テンプレートエンジンである Jarty のような，テンプレートによって分離をおこなうことも可能である [3]．しかし，テンプレートを記述するために独自の記述方法を学習する時間が必要となる．また繰り返しなどの制御コードを記述する必要があり，テンプレート部分が複雑になる．本研究では HTML と JavaScript を分離し，主に JavaScript 内での記述とすることで，テンプレートのように学習時間をほとんど必要としない．また配列形式の取得データを自動判別し，反映処理を繰り返しおこなうライブラリ関数を提供することで，制御コードを記述する必要がない．

5 おわりに

本研究では，JavaScript を用いた Web アプリケーションの通信処理を考察し，文書構造の HTML と振舞いの JavaScript を分離するために，対応関係を用いる JavaScript プログラム内の記述方法を提案した．また，ライブラリ関数を用いることで類似した処理を削減できることを評価した．

本研究のブラウザへの反映処理では，連想配列を用いてデータの対応関係をとった．今回は，取得したデータの形式が JSON 形式である場合の対応関係の記述方法を考えたが，通信で取得したデータの形式は複数存在するので，今後は他のデータの形式を考慮した対応関係をとる必要がある．また，通信に必要なデータを取得するためには，ユーザインターフェース部分の JavaScript プログラムを分析する必要がある．

参考文献

[1] S.Willison,“ *Unobtrusive JavaScript with jQuery*, ” `http://simonwillison.net/static/2008/xtexh/`, 2008.

[2] 寺内敦，山中顕次郎，加藤順, “ 通信サービスにおけるエラー処理の自動生成, ” 情報処理学会全国大会講演論文集，pp.159-160，1992.

[3] kotas,“ *Jarty*, ”`https://guthub.com/kotas/jarty`, 2010.