オフライン実行可能な Web アプリケーションの開発支援に関する研究

2008MI219 千賀 英佑　　2008MI241 武田 純一　　2008MI268 山田 裕人
指導教員　蜂巣 吉成

1　はじめに

現在，Ajax 技術などによりこれまでのデスクトップアプリケーションと同様に使える Web アプリケーションが普及している．スマートフォンなどのモバイルデバイスの普及により，ネットワークから切り離されたときの使用を考慮する必要が出てきた．モバイルデバイスはネットワーク接続の維持が難しく，ネットワーク接続が切れても利用できることが求められている．HTML5 の API の策定と主要な Web ブラウザの実装により，標準的技術を用いてオフライン実行可能な Web アプリケーションが構築可能となった．

HTML5 の API の仕様はオフライン実行可能な Web アプリケーション構築に必要な最低限のものである．例えば API の 1 つである Web Storage はクライアントサイドのデータストレージであるが，キーと値のペアを保存するだけの単純な仕組みである．従ってキャッシュ削除などのキャッシュ管理機能は備えておらず，キャッシュの管理を行うとき不可欠である保存日時やデータ量などの情報は付加されない．開発者はそれぞれ独自にこれらの機能を実現し，構築を行わなければならないので開発にかかるコストが増える．現在，オフライン実行可能な Web アプリケーションは，開発支援が十分になされているとはいえない．

本研究はオフライン実行可能な Web アプリケーションの開発支援を目的とする．Web アプリケーションをオフライン実行可能にするということは 1 つのアプリケーション内にオフライン実行時，オンライン実行時の記述をし，オフライン，オンライン時それぞれに応じた使い分けを可能にすることである．そこで一般的なオンラインで動作する Web アプリケーションを CRUD の観点から分類し，種別毎の構造からオフラインで動作させる時に必要な構造，機能を考察する．また，考察によって得られた結果からオフライン実行可能な Web アプリケーションを構築のためのライブラリの提案を行う．提案するライブラリを用いれば，既存のアプリケーションを大きく変更せずにオフライン実行可能とできる．

2　背景技術

2.1　オフライン実行可能な Web アプリケーション

Web アプリケーションでは，ブラウザは Web サーバから HTML ファイル，CSS，画像データ，JavaScript によるプログラム (以下，静的なファイルと呼ぶ) を取得し，ブラウザの画面にこれらを表示する．現在，Ajax 技術を用いて通信をおこなう Web アプリケーションが普及している．この場合，ブラウザで実行されている JavaScript プログラムが Web サーバとデータを送受信し (以下，動的なデータと呼ぶ)，ブラウザの画面を更新する．これらの Web アプリケーションの例としては，地図アプリケーションやスケジュール管理アプリケーションなどがある．このような Web アプリケーションをネットワークに接続されていないオフライン状態で実行するには次のことが必要となる．

a)　静的なファイルの扱い
  - a1)　静的なファイルをクライアントで保存
  - a2)　保存された静的なファイルをブラウザで読み込み画面表示

b)　動的なデータの扱い
  - b1)　動的なデータをクライアントで保存
    - サーバから受信したデータの保存
    - サーバに送信するデータの保存
  - b2)　保存された動的なデータをブラウザで読み込み画面表示
  - b3)　クライアントに保存されている動的なデータとサーバデータをオンライン時に同期

2.2　関連研究・技術

文献 [1] ではモバイルデバイスを対象にしたオフラインの閲覧環境を提案している．プリフェッチサーバと呼ばれる代理サーバがオリジナルの Web サーバから静的なファイルを事前に取得し，クライアントはプリフェッチサーバから取得したファイルを Google Gears を用いて保存し，オフラインでの Web ページ閲覧を可能にしている．しかし，保存対象は静的なファイルであるので，動的なデータによって動作する Web アプリケーションはオフラインで実行できない．

HTML5 ではオフライン実行可能な Web アプリケーション開発のために Application Cache[2] と Web Storage[3] が策定された．Application Cache は静的なファイル，Web Storage は動的なデータを保存する仕組みである．Application Cache は，静的なファイルをローカルにキャッシュさせ，Web ページの読み込み時間短縮を可能とする．Application Cache は，従来のブラウザのキャッシュ機能とは違い，キャッシュするファイルをマニフェストファイルから指定できる．2.1 節の a はこの API を用いることで解決する．Web Storage は，クライアントサイドにデータを保存する Key-Value 型のデータストレージである．テキストデータを扱うことができ，JavaScript でその操作内容を記述する．Web Storage は動的なデータのキャッシュを実現する．2.1 節の b1，b2 はこの API を用いることで解決する．

2.3 問題点

Application Cache は静的なファイルをローカルにキャッシュできるが，動的なデータはキャッシュ不可能である．Web Storage は動的なデータのキャッシュを実現するがキーと値のキャッシュを行うだけの単純な仕組みであるので，保存した日時，更新した日時，データサイズなど Web アプリケーションの開発に有用とされる情報が付加されておらず，開発者の負担が大きくなる．また，削除しない限りキャッシュを保管し続けるが，保存出来るデータ量に限りがあるので容量超過した時にアプリケーションが正常に動作しなくなる．2.1 節の b3 は現状の API を単純に用いただけでは対処できない．

3 Web アプリケーションの構造分析

Web アプリケーションは多種存在しており，そのすべてが同じ方法の支援でオフライン実行可能とするアプリケーションの構築ができるとは限らない．そこで従来のアプリケーションを CRUD の観点から分類し，分類毎の構造からオフラインで動作させる時に必要な構造，機能を考察する．

3.1 Web アプリケーションの分類

サーバデータの読み込みのみを必要とするアプリケーションは，動作に必要なファイル，動的なデータのローカル保存によりオフライン実行を可能にできる．このアプリケーションではフォームの入力からサーバデータ取得のための問い合わせ用データを作成し，HTTP リクエストを送信する．その後，サーバからのレスポンスデータを基にアプリケーションを動作させる．この場合のユーザの要求は，過去の検索結果をオフライン時でも再度確認できることである．この実現のために，サーバデータ取得のための問い合わせ用データを key，レスポンスデータを value として保存する．これによりオフライン時の動作が可能となる．時刻表検索のアプリケーションの場合は，駅名や日時などが問い合わせ用データ (key) となり，時刻表がサーバからのレスポンスデータ (value) となる．これらを保存することで過去に検索したデータをオフラインでも表示できる．

データの作成，更新，削除などサーバデータの操作も必要とするアプリケーションでは，動作に必要なファイル，動的なデータ以外に，作成，更新，削除などのデータ操作内容もローカルに記録する必要がある．この場合のユーザの要求は，オンライン時に取得したサーバデータをオフライン時にクライアントで編集し，オンライン時にその編集したデータをサーバに送信し，結果を反映させることである．この実現のために，オンライン時にサーバデータを取得し，ローカルに保存する．オフラインでのデータ編集作業をローカルに保存し，オンライン時に編集データをサーバに送信する．タスクリストアプリケーションの場合，オンライン時にサーバからタスク一覧を取得し，ローカルに保存する．オフライン時にタスクを編集した場合は編集内容を保存し，オンライン時に編集後のタスクをサーバに送信し，結果を反映させる．

3.2 サーバデータの読み込みのみを必要とする Web アプリケーション

オフラインで動作させるには，オンライン時に Web アプリケーションの動作に必要なファイル，データをキャッシュしておく必要がある．

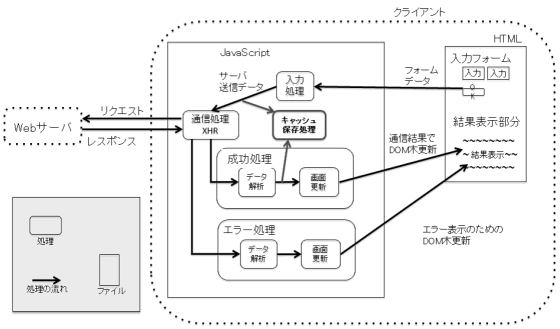

図 1　オフライン時に動作させるためのオンライン時の動作

図 1 はサーバデータの読み込みのみを必要とする Web アプリケーションをオフラインで動作させるためのオンライン時の動作である．キャッシュ保存処理で Web アプリケーションの動作に必要な動的なデータをキャッシュする．

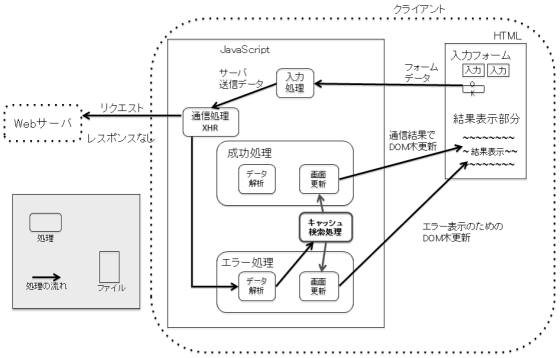

図 2　オフライン時の動作

図 2 はオフライン時の動作である．オフライン実行可能な Web アプリケーションは，データの最新性を重視する Web アプリケーションであるか，一度データを取得してしまえば当面は再取得する必要がない Web アプリケーションであるかでキャッシュを扱う順序が異なる．データの最新性を重視する Web アプリケーションでは，通信状況を確認し，オフライン時や通信エラーが起きた場合にキャッシュの検索処理を行い，キャッシュが存在した場合はキャッシュを基にアプリケーションを動作させる．一方，一度データを取得してしまえば当面は再取得する必要がない Web アプリケーションの場合は，はじめにキャッシュの検索処理を行い，キャッシュが存在しなかった時に通信を行う．

3.3 データの作成，更新，削除などサーバデータの操作も必要とする Web アプリケーション

オフライン時にアプリケーションの動作させるとき，次の 2 つの方法が考えられる．

方法 1 ローカルに保存するデータをサーバ送信用データとサーバから受信するデータに分けて保存し，オフライン時にそれらを基にアプリケーションを動作させる

方法 2 ローカル DB を利用し，その中にサーバ送信用データとサーバから受信するデータを同じデータ形式で保存し，オフライン時にそれらを基にアプリケーションを動作させる

4 ライブラリの設計と実装

4.1 データの読み込みのみを行う Web アプリケーション用ライブラリの設計

Web アプリケーションが新たにキャッシュデータを用いて動作する機能を得るにあたって，次のような問題に対処する必要がある．

- データの最新性が重視されるアプリケーションでは古いキャッシュデータが価値を持たない
- キャッシュできるデータの量は無制限でない

4.1.1 付加情報によるデータの管理

キャッシュデータは一般的にサーバへの問い合わせ用データをキーとして Web Storage に保存される．Web Storage は 1 つのキーに対し 1 つのデータしか記憶できない．この解決策としてキャッシュデータ毎に保存日時, 参照日時, 参照回数, データサイズなど固有の付加情報を記録し，管理する方法を採る．これら付加情報を各データの Web Storage のキーから特定可能な別のキーの示す領域に格納する．このような管理形態によって，これらデータは連携可能な 1 つのデータ単位として振る舞える．例えば，あるキャッシュデータの Web Storage におけるキーが「Key」であった場合,「Key_Save_Date」というキーにアクセスすることで，そのキャッシュデータの保存日時を参照できるようにする．

4.1.2 付加情報により実現されるキャッシュの破棄

キャッシュデータ毎に付加情報を記録することで，有効期限を過ぎたデータの存在を検知，破棄できる．また容量を超過してデータがキャッシュされようとしたとき，既にキャッシュされているデータから最も残す価値の低いデータを特定し，置換できる．残す価値の低いデータの特定方法として，一般的に LRU と LFU というキャッシュ置換ポリシーがある．LRU は最も参照日時の古いデータを，LFU は最も参照頻度の低いデータを残す価値の低いとみなすポリシーである．これらは最も代表的なポリシーであるが，単純な仕組みであるので必ずしも効果的とは言えない場合がある．そこで Proxy サーバの Squid[4] で採用されている LFUDA と GDSF を採用する．

LFUDA(LFU with Dynamic Aging)

LFU では過去に参照頻度の高かったキャッシュデータが後々参照されなくなった時，削除されにくいが，LFUDA はデータの参照頻度だけでなく参照日時も考慮できる．データがキャッシュされた時，またはデータが参照された時に，データ i のキー値 K_i を算出し，値の小さいものから削除する．キャッシュデータ i の参照回数を F_i，最後に削除されたドキュメントのキー値を L とすると，キー値 K_i は次の計算式で求められる．

$$K_i = F_i + L$$

GDSF(Greedy Dual-Size with Frequency)

サイズの大きいキャッシュデータから削除していけば，より多くのデータを保持できヒット率を高められるが，結果的にサイズの大きいデータを Web サーバから得ることになり，ネットワークの負荷を軽減できない．GDSF はデータサイズだけでなく，その参照されやすさも考慮できる．LFUDA と同様にキャッシュデータ i のキー値 K_i を算出し，キー値の小さいものから削除するが，計算式が異なる．キャッシュデータ i のサイズを S_i とすると，キー値 K_i は次の計算式で求められる．

$$K_i = F_i/S_i + L$$

4.1.3 付加情報により実現されるその他の機能

付加情報を用いることで，オフライン実行可能な Web アプリケーション特有の機能である履歴の表示，履歴の保護，履歴の削除の機能を実装できる．

4.2 データの作成，更新，削除を行う Web アプリケーション用ライブラリの設計

4.2 節の 2 つの方法のうち方法 1 は受信と送信のキャッシュを整合させて管理するのが難しいという問題がある．方法 2 は既存のアプリケーションの構造が変わるが，データの整合性を取る必要がない．ゆえに方法 2 を採用する．**図 3** に設計するライブラリの概要を示す．

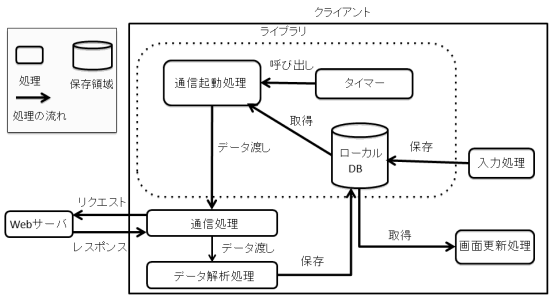

図 3 CRUD 用ライブラリ

オフライン時にクライアントで作業したデータの保存は次のような順序で行う．

1. ブラウザで入力したフォームデータを JavaScript プログラムに渡す

2. ローカルで作業したデータをローカル DB に保存する際，それが作成，更新，削除のどれにあたるかを示す情報を付加する
3. タイマー関数によって通信起動処理を起動し，ローカル DB からサーバと未同期のデータを抽出し，各関数へ渡す
   - データの新規作成用の関数
   - 既存データ更新用の関数
   - 既存データ削除用の関数
4. 通信を行う
   - 通信成功ならばデータをサーバと同期させる
   - 通信失敗ならばデータを未同期にしておく
5. 以前ローカル DB に保存したデータの中で未同期だったものを同期済の状態にする
6. 必要に応じてローカル DB に保存されたデータを基にブラウザの画面表示を更新する

4.3 実装

JavaScript によるライブラリの実装を行い，データの読み込みのみを行う Web アプリケーション用ライブラリは 350 行程度，データの作成，更新，削除を行う Web アプリケーション用ライブラリは 100 行程度となった．

5 考察

実際にライブラリの適用実験を行い，作成したライブラリによる開発作業削減の効果について考察する．

5.1 データの読み込みのみを行う Web アプリケーション用ライブラリ

オフライン時にサーバからの応答を得られない場合には動作しない従来の Web アプリケーションに，オフラインで動作する機能を付加することで，通信失敗時でもキャッシュデータを用いた動作が可能となる．**図 4** は，Web アプリケーションの通信の成否に関する箇所に 10 行程度の記述を追加することで，Web アプリケーションにオフラインで動作する機能を付加した例である．

```
// 通信に成功した場合
success : function (json, textStatus) {
  ...
  oflShowMsg("online");
  oflOnline();
  oflPut(word, JSON.stringify(json));
  displayOnSuccess(word, JSON.stringify(json));
  ...
}
```

```
// 通信に失敗した場合
error : function (xOptions, textStatus, errorThrown) {
  ...
  oflShowMsg("offline: " + msg)
  oflOffline();
  var val = oflGet(word);
  if (val) {
    displayOnSuccess(word, JSON.stringify(val));
  }
  ...
}
```

図 4　Web アプリケーションへ追加された記述の例

また，通信処理を含むライブラリである jQuery-JSONP に本研究のライブラリを適用した場合の変更記述は 30 行程度であり，Web アプリケーションの記述に変更を加える必要はなかった．

考察

本研究は Web アプリケーションをオフライン実行可能化させる場合の変更箇所を明確にした．本研究で提案するライブラリを用いることで，Web アプリケーションにオフラインでも動作をさせる標準的な機能を容易に付与することができた．また jQuery-JSONP のように既存のライブラリに本研究のライブラリを適用することで，Web アプリケーション自体を変更することなくオフラインでの実行を可能にできる．このようにライブラリを用いてオフラインで動作する機能を付加することでデータのキャッシュやキャッシュデータの取り扱いなど Web アプリケーションの記述を複雑化させやすい機能の分離が達成される．

5.2 データの作成，更新，削除を行う Web アプリケーション用ライブラリ

図 5 にデータの作成，更新，削除を行う Web アプリケーションへのライブラリ適用を示した．これによりデータを作成，更新，削除した結果をローカル DB へ保存でき，サーバ通信時にサーバとローカル DB の未同期データの定期的な探索とサーバへの送信を可能とした．

```
// 新規作成のデータをローカルDBに保存
function create_data(){
  var newData = {
              "id": localID,
              "title": document.addplan.title.value,
              "date": document.addplan.date.value,
              "contents":document.addplan.contents.val
ue,
"importance": document.addplan.importance.value
  };
  create_localdb_from_local_data(newData);
  display_todos(localData);
}
```

```
// サーバに新規作成のデータを送信する
function create_server(data) {
  var createurl = encodeURI(url) +
"?mode=create&title="
              + data.title + "&date=" + data.date +
"&contents=" + data.contents
              + "&importance=" + data.importance;
// 通信しているIDを保存
  var connID = data.id;
以下省略
```

図 5　CRUD アプリケーションへのライブラリ適用

考察

本研究で提案する CRUD 用ライブラリを用いることで，構造を変える必要があるが，Web アプリケーションにオフラインでも動作をさせる標準的な機能を容易に付与することができた．また，このようにライブラリを用いて Web アプリケーションにオフラインで動作する機能を付加することで，サーバ受信データとオフライン時のローカル作業データの管理をすることができた．

6 おわりに

本研究はオフライン実行可能な Web アプリケーションの構造を分析することで問題を整理した．またその問題点解消，開発作業削減を実現するライブラリを作成し，適用実験によって効果を確かめた．今後は今回挙げた事例以外についても適用効果を検証する必要がある．

参考文献

[1] Yung-Wei Kao, Ching-Tsorng Tsai, Tung-Hing Chow, and Shyan-Ming Yuan, “An Offline Browsing System for Mobile Devices,” iiWAS ’09, 2009.

[2] W3C, “*Offline Web Applications*, ” http://www.w3.org/TR/offline-webapps/, 2011.

[3] W3C, “*Web Storage*, ” http://www.w3.org/TR/webstorage/, 2011.

[4] Squid, “squid : cache_replacement_policy configuration directive, ”http://www.squid-cache.org/Doc/config/cache_replacement_policy/, 2011.