Vol.2012-DBS-155 No.14
2012/11/19

Webアプリケーションでのページ遷移を伴わない認証認可処理の実装とセキュリティ

新居 雅行[1]

概要：Web アプリケーション開発では，クライアントサイドの JavaScript を利用するスタイルが積極的に行われるようになった．本論文では，クライアントサイドで JavaScript を活用した Web アプリケーションにおける，認証認可処理の実装方法について検討を行う．通常の Web アプリケーションでは，ログインページや認証，認可の処理をすべてサーバ側で行う．一方で JavaScript を活用した Web アプリケーションでは，サーバとクライアントの双方での処理分担を設計する必要がある．本論文では，ページ遷移を引き起こす事なく認証認可の処理を実現するための，クライアントとサーバサイドでの処理割当について考察を行い，その割当に基づき効率的に Web アプリケーションを開発するためのフレームワークを開発した．クライアント側に処理機能があることから，その機能をアプリケーション開発者が意図していない利用をする可能性があり，その点から考えられる対策についても検討した．

The Implementation of Authentication and Authorization and The Security Issues in a Web Application

Abstract: JavaScript is a familiar way to develop web application in these days. In this paper, I addresses the implementation of authentication and authorization of the client-side JavaScript-based web application. A typical Web application might handle authentication, authorization and login on the server side. Otherwise, a JavaScript -based web application has to be designed how the server and client sides handle each function to realize authentication and authorization. The design to devide the function without page reloading is discussed and then the framework to be able to develop efficiently is developed based on the conclusion. Although the framework has an API for the client-side additional program, it can be called from the program the application developer didn't intend. The framework is prepared for unexpected accesses.

1. はじめに

Web アプリケーションの開発では，サーバサイドの処理に加えて，JavaScript を利用したクライアントサイドへの処理の組み込みが積極的に行われている．JavaScript のライブラリでアプリケーションを作成するとページの書き換えがなく，必要に応じた通信処理が背後で稼働することから，見栄えやレスポンスが向上する．ユーザインタフェースを向上させれば，利用の促進や利用者の満足度が高くなることが期待でき，より良いユーザインタフェースを構築するために jQuery[1] などの JavaScript ライブラリが広く使われている．一方，多くの Web アプリケーションでは，利用者を限定したり，操作範囲を限定するために，認証や認可の仕組みが必要となる．従来の Web アプリケーションでは，サーバサイドで利用できるフレームワークでは認証や認可の仕組みが組み込まれており，仕様に則って必要な設定やプログラムを記述すれば，認証の機能は実現できる．

しかしながら，JavaScript を活用した Web アプリケーションにおいて，柔軟なページ再描画を活かしつつ認証認可の機能を提供する事は，一般的に利用されているフレームワークでは取り扱われていない．クライアントサイドでのプログラム構築を進めたとしても，認証や認可の処理はサーバ側で行うため，ログインページの表示といった認証を開始するプロセスではページのリロードが必要になる．たとえば，データの参照は認証が不要，データ更新は認証が必要という仕組みを持ったページを作成するような場合，ページ更新なしで構築するのは容易ではない．

本論文では，JavaScript を利用したアプリケーション開

[1] 自営業 Self-employed, nii@msyk.net

発に利用するフレームワークにおいて，認証と認可を実現する組み込み手法を提案する．JavaScript 内で認証の処理を完結させるために，ページ遷移を利用しない手法が必要となる．そこで，現在のページ上に新たなオブジェクトを表示してログインパネルを表示する手法を採用した．認可の適用においては，確実に指定した通りに適用されるための実装方法を検討した．JavaScript で稼働する部分で判定の処理を行うとクライアント側での改変が予想されるので，連動して稼働するサーバ側で判定を行うようにした．

認証や認可の仕組みは，筆者が開発している Web アプリケーションフレームワーク INTER-Mediator[2][3] で実装した．このフレームワークは，データベースに対する読み書きの処理を行う Web ページをプログラムを行わずに開発できるもので，中小企業など予算規模が小さな組織での業務ソフトを短期間に開発できるような基盤となることを目指して開発されている．なお，提案する手法は一定のルールを決める事で異なるフレームワーク間で連携した動作においても利用できる．

本稿では引き続き，第 2 章で認証認可に必要な機能が代表的なフレームワークでどのように実現されているかを解説する．第 3 章では実装を行ったフレームワークの認証や認可を伴わない場合の動作を解説する．第 4 章では実際に認証や認可を組み込む場合の検討事項とその結果を示す．第 5 章では実装した結果発生した検討事項と解決策を説明する．そして，第 6 章で関連研究についても言及する．

2. Web アプリケーションの認証と認可

Web アプリケーションでの認証や認可において必要な処理と，代表的なフレームワークがどのように処理するのかを最初に解説する．

機能別にフレームワークでの対応状況を表 1 にまとめた．Java フレームワークの Struts[4] はコンテナの Tomcat[5] の仕組みを併用した動作ができる．Ruby on Rails[6] については認証は Devise プラグイン [7]，アクセス権は Declarative Authorization プラグイン [8] を使う状況を想定した．CakePHP は Ver.2 以降の AuthComponent クラス [9] を使う状況を想定した．

2.1 ブラウザのログインパネル

HTTP プロトコル上で WWW-Authenticate ヘッダを利用した認証を行う場合 [10]，通常はブラウザが用意したログインパネルを表示する．この仕組みでの認証を利用するには設定のみでできる．なお，ページロード時に認証した結果は，JavaScript の通信処理でもユーザ名とパスワードは引き継がれて，HTTP プロトコルによる認証は継続されるのが一般的なブラウザの動作である．従って，ページロード時にログインパネルが表示されて認証されると，その結果は Ajax 通信でも引き継がれる．

2.2 HTML ページで作成したログインパネル

ログインパネルのページをフレームワークの機能で作成する．認証処理が失敗するとエラーページ，認証処理が成功すると本来のページを表示するという仕組みを，例えばリダイレクトを利用する．後処理についてはある程度のプログラムを行う．一般的なケースはいずれのフレームワークもドキュメントに解説されている．認証エラーのページもフレームワークの機能を使って作成する．

2.3 ユーザ記録のデータベースとの連携

いずれのフレームワークも，データベースにアカウントを記録して，そのアカウントで認証する機能が利用できる．また，認証を適用する単位のロールあるいはグループの設定もできる．データベース処理についてはモデル化したり，あるいは接続方法を設定するだけで利用できる．ただし，ユーザの追加やパスワードの変更などの仕組みはページを自分で作る必要があるが，Devise プラグインはオンラインサインアップも含めたユーザ管理の機能まで組み込まれている．

2.4 認証と認可の適用

いずれもアスペクト指向的な前処理を利用して，認証や認可の適用を行う．Struts ではアノテーションを利用したインタセプタを定義する方法などが使われる．Ruby on Rails ではフィルタを利用してメソッドの処理に入る前に認証や認可を適用できる．CakePHP ではハンドラという機能を利用する．いずれも実際の処理の前にプログラムを割り込ませるアスペクト指向の手法で，そのプログラムは開発者が組み込む．また，フレームワーク内では認証結果や認可の結果が参照できるので，プログラムを作るという上では任意の位置に認証や認可を適用できる．

2.5 ログイン状態の継続

Web アプリケーションでは複数のページを遷移するのが一般的だが，クッキーやセッションを使って認証トークンを記録するなどして，一定時間内ではユーザが再度ログイン作業をしなくても，一連のページを参照できるような仕組みを備えている．

3. フレームワークの基本動作

認証と認可の仕組みを筆者が開発しているフレームワークの INTER-Mediator 上で実現したが，まず，フレームワークのおおまかな構成と，認証や認可が関わる部分についてこれらの仕組みを組み込む前の動作について解説する．このフレームワークを使うことで，プログラムを追加しないでも基本的なデータベースと Web ページの処理が構築できることを目指した．一方，必要ならばプログラムによる拡張が可能である．

表 1 Web アプリケーションでの認証と認可に必要な機能
Table 1 Authorization and Authentication in Web applications

必要な機能		一般的なフレームワーク			JavaScript による実現の限界	INTER-Mediator での状況	
分類	機能	Struts (Tomcat)	Ruby on Rails	CakePHP Ver.2.x		対応状況	解決方法
認証	ログインパネル (ブラウザ)	設定のみ	設定のみ	設定のみ	JavaScript 単体では利用できないが, ページロード時に認証すれば HTTP 認証の結果がサーバとの通信に引き継がれる	×	-
	ログインパネル (HTML)	ページを作成	用意されている	ページを作成	作成可能	フレームワークが用意	ページ上に要素を追加してログインパネルを実現
	ユーザ記録データベース	JDBC の設定のみ	自動作成	ツールでモデルとコントローラを生成	サーバ連携必要	設定のみ	サーバ側のモジュールでデータベース処理を組み込む
	認証処理・エラー処理	Tomcat で可能	可能	可能 (認証無効時の AJAX 対応処理も含む)	サーバ連携必要	サーバ側で判定	
	ログイン状態の継続	○	○	○	○	○	JavaScript の変数あるいはクッキーで保持
	認証の適用	アノテーションを利用したインタセプタを作成して適用	用意されたフィルタを適用	ハンドラによる事前処理を適用可能	サーバ側での判定が必要	自動	サーバ側で処理前に自動判定
ユーザ管理	新規登録	要作成	用意されている	要作成	-	サンプルで対応	
	パスワード変更	要作成	用意されている	要作成	-	サンプルで対応	
	サインアップ	要作成	用意されている	要作成	-	要作成	
	パスワードリセット	要作成	用意されている	要作成	-	要作成	
アクセス権	グループ (ロール)	設定のみ	設定のみ	ハンドラによる事前処理で適用可能	サーバ連携必要	設定のみ	CRUD 個別あるいは全部に対して設定ができる
	適用範囲	メソッド単位	メソッド単位, ビュー単位, CRUD を含むモデル単位など	メソッド単位	-	ページ, テーブル, フィールド単位	レコードにユーザ名のフィールドを設置すればレコード単位も可能

フレームワークでは，ページのテンプレート処理（ページのひな形とデータベースの内容を統合する処理）をクライアントサイドで JavaScript で実装した．ユーザの応答に対する処理など，ユーザインタフェースに関連する事をすべて JavaScript で実装している．サーバとクライアント間の通信は，テーブルの CRUD に対応したデータのやりとりだけを行う．テンプレートの処理後，ユーザの操作を受けてデータベースを更新したり，さらには JavaScript のプログラムを実装することを考慮した結果，JavaScript で多くの部分を構成する事になった．

図 1 にはクライアントからデータ取得の要求を受けて，サーバと通信する典型的な処理を示した（以下の丸数字は

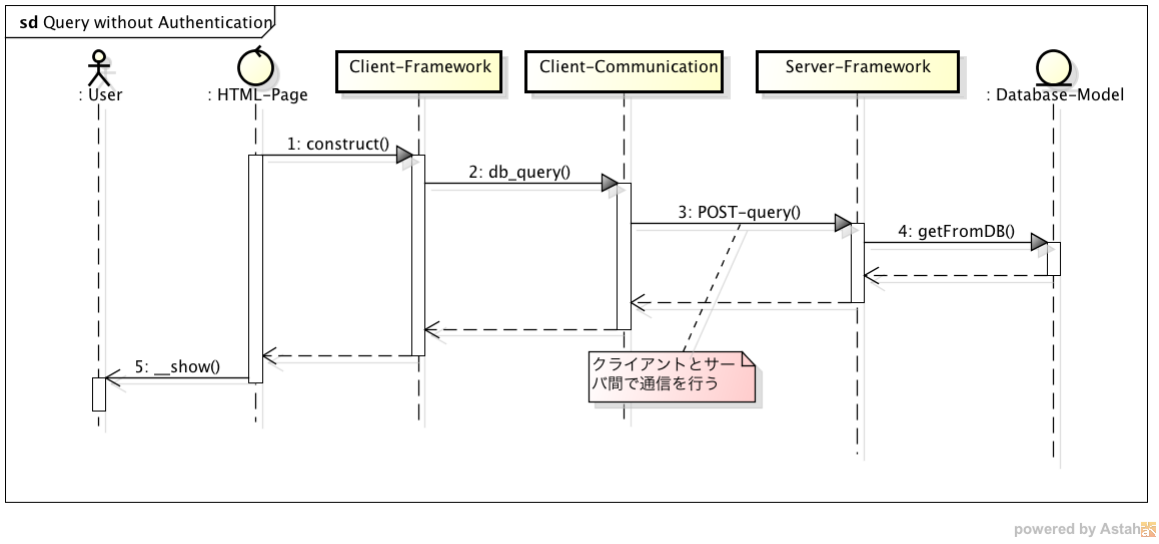

図 1 認証や認可を伴わない場合のデータベースアクセス
Fig. 1 The Database Accessing without Authentication and Authorization.

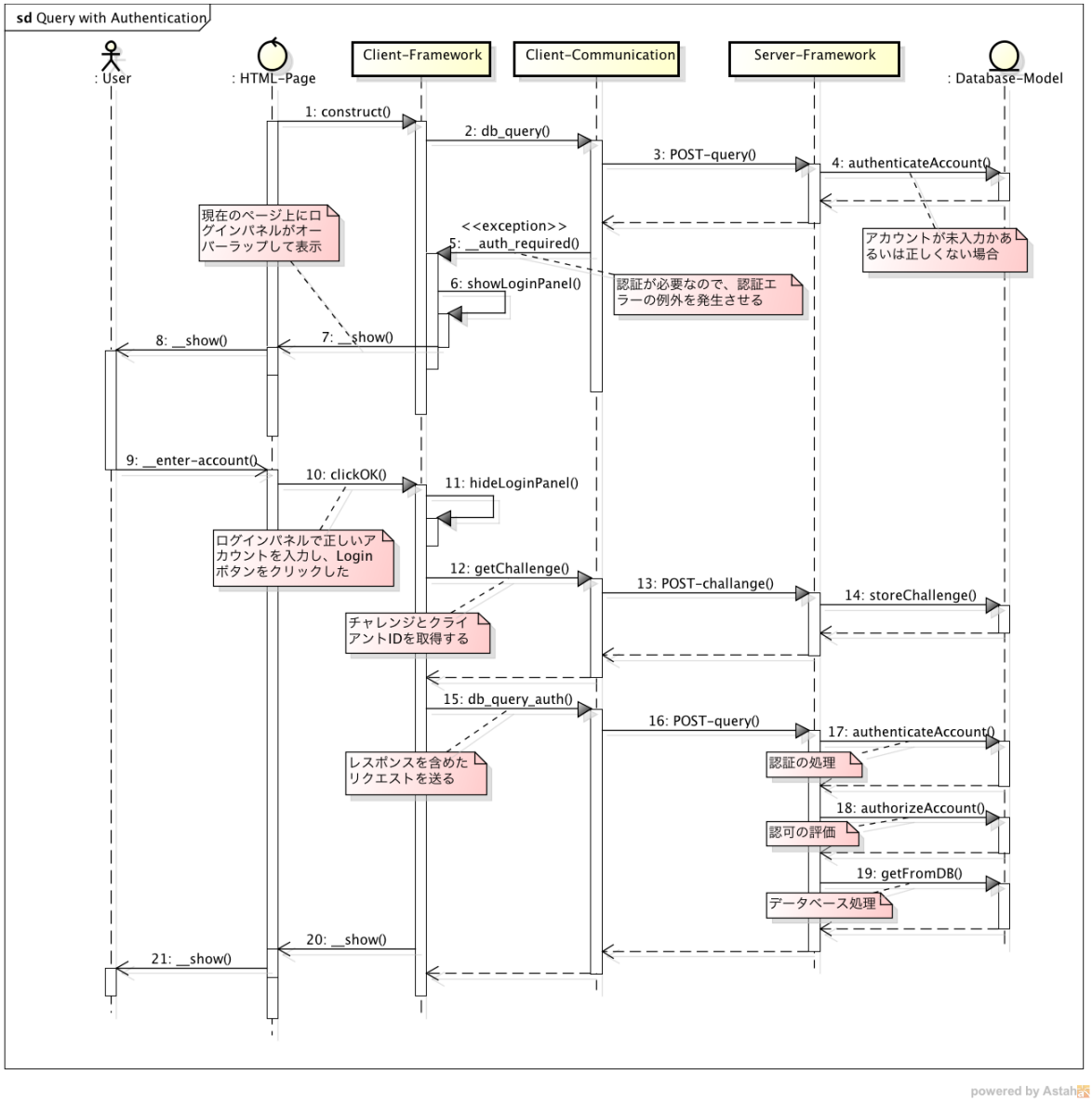

図 2 認証や認可を伴うデータベースアクセス
Fig. 2 The Database Accessing with Authentication and Authorization.

メッセージの番号)．①ページの合成を開始し，②内容に応じてデータベースのデータを要求する．③クライアントがサーバに通信を行い，④サーバ側で実際にデータを呼び出して結果を返す．その結果を受けて合成を続けて終了すると⑤ユーザはデータベースのデータを含むページを参照できる．

4. フレームワークへの認証と認可の組み込み

前出のフレームワーク INTER-Mediator に認証や認可の機能を組み込む上での設計方針と実装内容を説明する．クライアントとサーバへの機能の配分を決定し，実際に組み込み動作することを確認した．

認証や認可を実現する以外の要件は，フレームワーク利用者がプログラムを追加しなくても，設定のみで認証と認可ができるようにすることである．ユーザインタフェースとしてログインパネルやエラー表示を表示する．ユーザやグループの情報はデータベースに記録する．認可については，テーブルやビューに対応するエンティティごとに，CRUD（Create, Read, Update, Delete）のそれぞれについて可否を設定可能な認可が定義できる機能を基本とした．さらに粒度の小さな単位でのアクセス権については第 5 章で論じる．

4.1 機能の分配と実装方法

認証と認可に必要な機能のうち，ユーザやグループのテーブルはサーバサイドのデータベースに構築する．認証や認可の適用についても，クライアントサイドで実装すればユーザによって変更される可能性があることから，サーバサイドで実現しなければならない．一方，ログインパネルなどのユーザインタフェースはクライアントサイドでも実装が可能である．ログイン状態については原則は JavaScript 側の変数に記録するが，クッキーによるハッシュ値の保持でページ遷移にも対応する．

図 2 に認証と認可を含めたシーケンス図を示した．この処理のために，サーバとクライアント間では，チャレンジの取得のための通信処理を追加している．①の construct 関数はテンプレート処理を行うもので，ページロード時やボタンをクリックしたときに呼び出される．認証の確認は④や⑰のように，サーバ側のモデル化したデータベース処理の中で行っている．この一連のデータベース処理で認証が必要という設定があれば，④では認証エラーとなり，サーバサイドではこれ以上の処理は行わず，クライアントに処理を返す．

認証や認可の適用と，ユーザインタフェースの表示を両立させるための組み込みが必要である．図 3 は，既存の construct 関数を，認証を適用しユーザインタフェースを適切に表示する construct 関数に変更した結果である．JavaScript の例外処理と関数オブジェクトの機能を使って既存の関数を認証対応にラッピングするような実装を行った．元からある construct 関数を認証と認可に対応する．元の construct 関数を construct_main 関数に名前を変えておく．construct_main 関数の呼び出しで認証が正しくされていない場合は認証エラーが例外で返ってくるのをキャッチして，その場合にログインパネルを表示する showLoginPanel 関数を呼び出す．showLoginPanel 関数の引数は，パネルで「ログイン」ボタンをクリックしたときに呼び出される関数オブジェクトを指定するが，そのときにもう一度同じ条件で construct 関数を呼び出す．認証エラーがあれば，何度もログインパネルが表示され，認証結果が正しければ続けて処理を行うという仕組みを実現できる．

```
1  function construct(x) {
2    try {
3      construct_main(x);  //元からある処理の呼び出し
4    } catch (exception) {
5      if (exception == AUTH_EXCEPTION)  {
6        showLoginPanel(
7          function()  { construct(x); }
8        );
9      } else {
10       //その他のエラー処理
11     }
12   }
13 }
```

図 3 通信処理関数をベースに認証付きの通信処理関数を作る
Fig. 3 The authenticate available function from the normal communication function.

実際には，construct 関数から，クライアントサイドの通信処理を行っている間にいくつかの関数を経由する．それらの関数では，認証エラーの例外をキャッチし呼び出し元にスローすれ．その結果，ログインパネルを表示する仕組みと認証機能を組み込む前からある動作がいずれも正しく稼働する．フレームワークの外部から内部に対する呼び出し部分にこのような組み込みを適用することで，フレームワークのすべての機能が認証と認可に対応できた．

4.2 認証処理

図 2 に示したログインパネルでの「ログイン」ボタンをクリックした後についての動作を説明する．showLoginPanel 関数では，クリックした後，引数に指定した関数オブジェクトを実行する前に⑫から一連の流れで，サーバからチャレンジとクライアント ID，そしてログインしようとしているユーザのソルトを取得する．それをもとにして，⑮より実際のデータベースアクセス処理を行う．サーバサイドで⑰認証が成功し，⑱認可により許可された状態なら，⑲データベース処理を行うという流れになる．

表 2　認証のために転送されるデータと動作
Table 2　Communication for Authentication

クライアント側での処理	転送される認証情報	サーバ側での処理
チャレンジ要求	→ user →	データベースより hpw を取得し salt を求める ch を乱数より生成，cid が未発行なら乱数で生成 user, cid, ch, 日付時刻をデータベースに記録
res を求める	← salt, cid, ch ←	
データベース要求	→ cid, user, res →	データベースより，cid からチャレンジを取得 $res = HMAC(ch, hpw + cid)$ が成り立っていれば認証は OK

なお，パスワードはデータベースのフィールドに 4 バイトのソルトを付加した SHA-1[11] でハッシュした値を利用した．認証結果の比較は，チャレンジを鍵として，ハッシュ化したパスワードをメッセージとした HMAC-SHA512[12][13] によるハッシュを利用し，その値がレスポンスとして返される．定式化すると，以下の通りである．転送されるデータと動作は表 2 に示した．ネットワーク上にパスワードは流れてはいない．ソルトの値はパスワードのハッシュにつなげて 1 つの文字列としてデータベースに記録した．ソルトの長さは一定なので，データベースに記録した値からソルトおよびハッシュ値の部分は容易に分離できる．

+: 文字列の接続演算子
user: ユーザ名
pw: 本来のパスワード
salt: ユーザごとに異なるソルト
hpw: データベースに保存されているハッシュ値

$$hpw = SHA1(pw + salt) + salt$$

pw': 入力したパスワード
cid: クライアント id
ch: チャレンジ
res: レスポンス

$$res = HMAC(ch, SHA1(pw' + salt) + cid)$$

クライアント ID は 1 つのアプリケーション内でクライアントを識別するために利用する．ユーザ名でユーザを特定すると，複数のブラウザから同一のユーザでログインしたときの識別ができない．また，IP アドレスでの判断だと NAT の内部からのアクセスで区別できない可能性がある．クライアント ID が発行されていない場合にチャレンジ発行時に乱数で生成される．サーバ側ではデータベースのテーブルで，発行したチャレンジとクライアント ID，日時を記録しているので，クライアントからの応答に含まれる認証関連の情報はクライアント ID とレスポンスだけになっている．また，この段階ではセッションやクッキーは使用していない．

また，データベースの応答にも，チャレンジ応答に相当するデータが含まれていて，次のデータベースアクセス時にチャレンジ要求と応答は省略できるようにした．一度レスポンスを返したチャレンジはそこで無効化する．チャレンジの値が毎回違うので，レスポンスの値も毎回異なり，これらの値をネットワーク経路上などで横取りをしても，パスワードを得ない限りは認証処理を通すことはできないと言える．

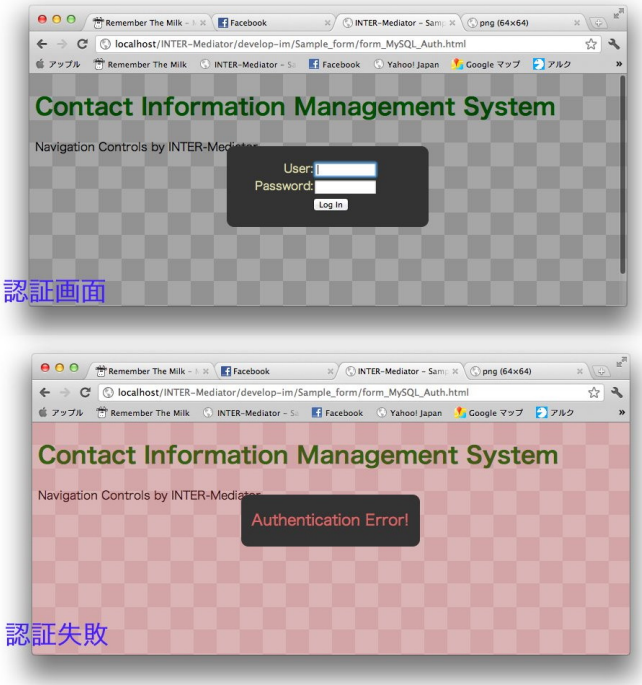

図 4　フレームワークが表示する認証用パネル
Fig. 4　The Login Panel the Framework Generated.

4.3　ログインパネルとエラー処理

クライアント側では図 4 のようなログインパネルと認証エラーを表示する機能を実装した．実際に生成するページはそのままに，ウインドウいっぱいに半透明で市松模様のオブジェクトを配置し，その上に黒いパネルを配置している．このパネルの 1 つ 1 つのオブジェクトは，DOM の仕組みを利用し JavaScript のプログラムで生成している．ログインパネルに入るタイミングで何回連続して表示したか

をカウントし，一定以上の場合は認証エラーが出るようになっている．

4.4 認証の継続

別のページに移動したときにも認証結果を継続できるように，認証に必要なハッシュをかけたパスワードなどの情報をクッキーに記録することができる．クッキーが存在すれば，そこからハッシュなどを取り出して，認証に必要な応答を行うようにしている．

しかしながら，JavaScript の利用が前提であるのなら，1 つの HTML が 1 つのページ（ビュー）という構成でない作り方も可能である．CSS では display 属性が定義されていて，要素とそのすべての子要素を非表示あるいは表示にする処理が JavaScript のプログラムで簡単にできる．2 つのページがあるのなら，それぞれのページの中身を div 要素で囲み，適当な id 属性を設定すれば良い．認証に必要な情報は JavaScript のオブジェクトのプロパティに記録してあり，クッキーに記録する事なく，1 つの HTML ファイルに定義されている複数のページで認証が継続する．

5. クライアントでの動作に関連する問題

サーバサイドで稼働する Web アプリケーションの場合，書き込みが可能なメソッドを記述しない限りは，データベースへの書き込みはアプリケーション経由ではできない．しかしながら，JavaScript で稼働するフレームワークがクライアントに存在することで，ユーザがフレームワークの機能を使用したり回避したりするプログラムを独自に追加してしまうことも可能である．この点が問題が生じる主要な理由である．認証と認可の処理の組み込み際して問題になった点と対処方法について検討する．

5.1 既定値で書き込みが可能な問題

読み書きの処理が可能なフレームワークを JavaScript で構成すると，アプリケーション自体が読み出しのみの処理しかしていない場合でも，ユーザによって書き込みプログラムを追加されてしまう可能性がある．認可の設定を正しく行い，クライアントから書き込みを防止したい場合には読み出しのみに権限を与えるといった設定をする．別の対策としては，フレームワークからデータベースに接続するときに利用するデータベースエンジン側のアカウントに対して書き込みを禁止するという方法がある．INTER-Mediator ではいずれの方法にも対応する．

5.2 スキーマ情報が参照できる点について

JavaScript で Web アプリケーションを構築する場合，データベースのスキーマ情報がクライアントで参照できる場合がある．スキーマ情報を完全に隠匿する方法としては，テーブル名やフィールド名の別名を定義する方法がある．この点は SQL インジェクションをしやすくする可能性はあるものの，データベースアクセス部分で SQL インジェクションへの対策が行われていれば，スキーマ情報が見える事は問題ないと考えられる．さらに，INTER-Mediator ではテーブルからのデータを取得するときに，クライアントから改変できない設定として検索条件が与えられているため，たとえば，主キーフィールドの値を 1 から順番に全部取得しようとしても，あらかじめ指定されている条件に会わないレコードは取得できないようになっている．

5.3 意図しないフィールドの更新

認可の設定によりテーブルを更新できるようにしてある場合，そのテーブルのすべてのフィールドが更新可能になる．その結果，主キーフィールドや外部キーフィールドの値をクライアント側から改変ができてしまい，データ間のリンクが破壊されて問題になる可能性は高い．この問題を回避するためには，書き込み可能なテーブルに対しても特定のフィールドの書き込みを禁止する認可の設定が必要となる．

レコード単位の認可を実現するために，フレームワーク内に必要な機能の組み込みを行っている．特定のレコードに対するアクセス権があるユーザあるいはグループを特定するため，ユーザあるいはグループ名を記録するフィールドがテーブルには必要になる．認証結果に応じてサーバ側で自動的にユーザ名やグループ名をそのフィールドに入力する．そのフィールドの値についてもクライアント側から値を直接書き込む事もできなくする必要がある．

6. 関連研究

OpenXUP[14] では，クライアントサイドとサーバサイドの通信プロトコルを定義した上で Ajax 通信を行い，ユーザインタフェースに対する応答をサーバ側と連動する事を実現している．クライアントのステータスをサーバ側で保持し，クライアントの変更に応じた処理をサーバに伝達した上で必要なデータを取得するなどの処理を行い，応答をクライアントに送る．この手法だと，クライアント側に JavaScript のライブラリを使ったプログラムを追加することは困難になる．そのことは安全にもつながるものの，拡張性は限定される．INTER-Mediator では，ステータス管理をクライアントに集約し，通信はデータのやりとりだけといった単純なものとする方が開発コストが低いと判断し，これまでに解説した手法を採用している．

Matt Raible 氏による ajax-login[15] は，Spring フレームワーク [16] と jQuery を使って Ajax 機能の上でログイン処理を行うというものである．ページ遷移をせずにログインパネルを表示し，認証は Spring フレームワークの仕組みと連動する．サーバサイドのフレームワークでのクライアントサイドスクリプトの取り込みは Spring だけでな

く多くのフレームワークで行っており，既存の Web アプリケーションで JavaScript を組み込む場合の有力な手法である．従来は Ajax ベースでの認証のためには独自に開発をする必要があったが，フレームワーク上で実現することで，こうしたユーザインタフェースの利用が促進されることが期待できる．

7. まとめ

サーバとクライアントの双方に実行プログラムがあるフレームワークを使用した Web 開発アプリケーションにおいて，認証と認可の仕組みを実装した．認証と認可の確認をサーバで行い，ユーザインタフェースをクライアント側で行うことで，必要な動作を実現できる事を確認した．クライアントとサーバの間では，ユーザインタフェース情報はやりとりせずに，データのみをやりとりさせているが，その上で，認証に必要な通信内容として，チャレンジのやり取りを追加した．ユーザインタフェースは，DOM モデルの処理を利用して，既存のページ上に半透明のパネルを上書きする形式で表示し，ページ遷移なく表示できた．一方，クライアント側にプログラムが存在し，状況によってはユーザによって書き換えが可能な状況にもなる．そこでのセキュリティ上の問題点を洗い出し，フレームワークに搭載した．

最後に，本稿に対して有益なアドバイスをいただいた，国立情報学研究所アーキテクチャ科学研究系助教の鄭顕志先生にお礼を申し上げます．また，Ruby on Rails や CakePHP の機能についてアドバイスをいただいた株式会社エミック代表取締役の松尾篤様にもお礼を申し上げます．

参考文献

[1] The jQuery Foundation: jQuery, , available from ⟨http://jquery.com/⟩ (accessed 2012/9/9).
[2] Nii, M.: INTER-Mediator, , available from ⟨http://inter-mediator.info/⟩ (accessed 2012/9/9).
[3] 新居雅行：ページ要素をデータベースに直接結び付けるフレームワークと Web サイト設計手法，情報処理学会研究報告．データベース・システム研究会報告， Vol. 2011, No. 31, pp. 1–8（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110008682639/⟩ (2011-10-27).
[4] The Apache Software Foundation: Struts, , available from ⟨http://struts.apache.org/⟩ (accessed 2012/9/9).
[5] The Apache Foundation: Apache Tomcat, , available from ⟨http://tomcat.apache.org/⟩ (accessed 2012/9/9).
[6] Rails Core Team: Ruby on Rails, , available from ⟨http://rubyonrails.org/⟩ (accessed 2012/9/9).
[7] Valim, J., da Silva, C. A. and Flores, R.: Devise, , available from ⟨https://github.com/plataformatec/devise⟩ (accessed 2012/9/9).
[8] Bartsch, S.: Declarative Authorization, , available from ⟨https://github.com/stffn/declarative_authorization⟩ (accessed 2012/9/9).
[9] Cake Software Foundation, Inc.: CakePHP Cookbook 2.x, Core Libraries, Authentication, , available from ⟨http://book.cakephp.org/2.0/en/core-libraries/components/authentication.html⟩ (accessed 2012/9/9).
[10] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. and Stewart, L.: HTTP Authentication: Basic and Digest Access Authentication, , available from ⟨http://tools.ietf.org/html/rfc2617⟩ (accessed 2012/9/9).
[11] National Institute of Standards and Technology: FIPS 180-1, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-1, Technical report, US Department of Commerce (1995).
[12] Bellare, M., Canetti, R. and Krawczyk, H.: Keying Hash Functions for Message Authentication, *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, London, UK, UK, Springer-Verlag, pp. 1–15 (online), available from ⟨http://dl.acm.org/citation.cfm?id=646761.706031⟩ (1996).
[13] National Institute of Standards and Technology: FIPS 180-2, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-2, Technical report, US Department of Commerce (2002).
[14] Yu, J., Benatallah, B., Casati, F. and Saint-Paul, R.: OpenXUP: an alternative approach to developing highly interactive web applications, *Proceedings of the 6th international conference on Web engineering*, ICWE '06, New York, NY, USA, ACM, pp. 289–296 (online), DOI: 10.1145/1145581.1145638 (2006).
[15] Raible, M.: github: ajax-login, , available from ⟨https://github.com/mraible/ajax-login/⟩ (accessed 2012/9/9).
[16] SpringSource: Spring, , available from ⟨http://www.springsource.org/⟩ (accessed 2012/9/9).