

セキュアプログラミング(1)

—総論と各論, Webを中心に—

上原哲太郎
UEHARA Tetsutaro

Webアプリケーションの特性

アプリケーション : Application

- WebブラウザとWebサーバとの緩いクライアント・サーバ構成
 - HTTPという, もともとは文書配布用の単純なプロトコルの上に構築
- ブラウザはサーバにHTTP requestを送信
- サーバはブラウザにHTML(+スクリプト)やCookie、画像ファイル等を送信
- ブラウザはスクリプトやユーザの操作(クリック等)によって新たなHTTP requestをサーバに送信 以下繰り返えし
- ブラウザとサーバの間の通信は中継されることがある(プロキシサーバなど)
- ブラウザとサーバは互いに信用できない！
 - HTTPは常にサーバが期待したrequestであるに限らない
 - HTMLは書き換えられているかもしれない

クライアント・サーバ : Client/Server

ネットワークアクセス : Network Access

プロトコル : Protocol

プロキシサーバ : Proxy Server

ブラウザ : Browser

スクリプト : Script

Webに起こりがちなセキュリティ問題

- セッション問題

セッション : Session

- セッション維持の仕組みが堅牢なものではなく、乗っ取りのおそれ

- アクセス制御

- ページ遷移をきちんと設計しないと迂回のおそれ

- 入力問題

- サーバへの入力(リクエスト)に悪意ある内容が含まれる恐れ

- SQLインジェクション
- コマンドインジェクション

インジェクション : Injection

- エコーバック問題

エコーバック : Echo-back

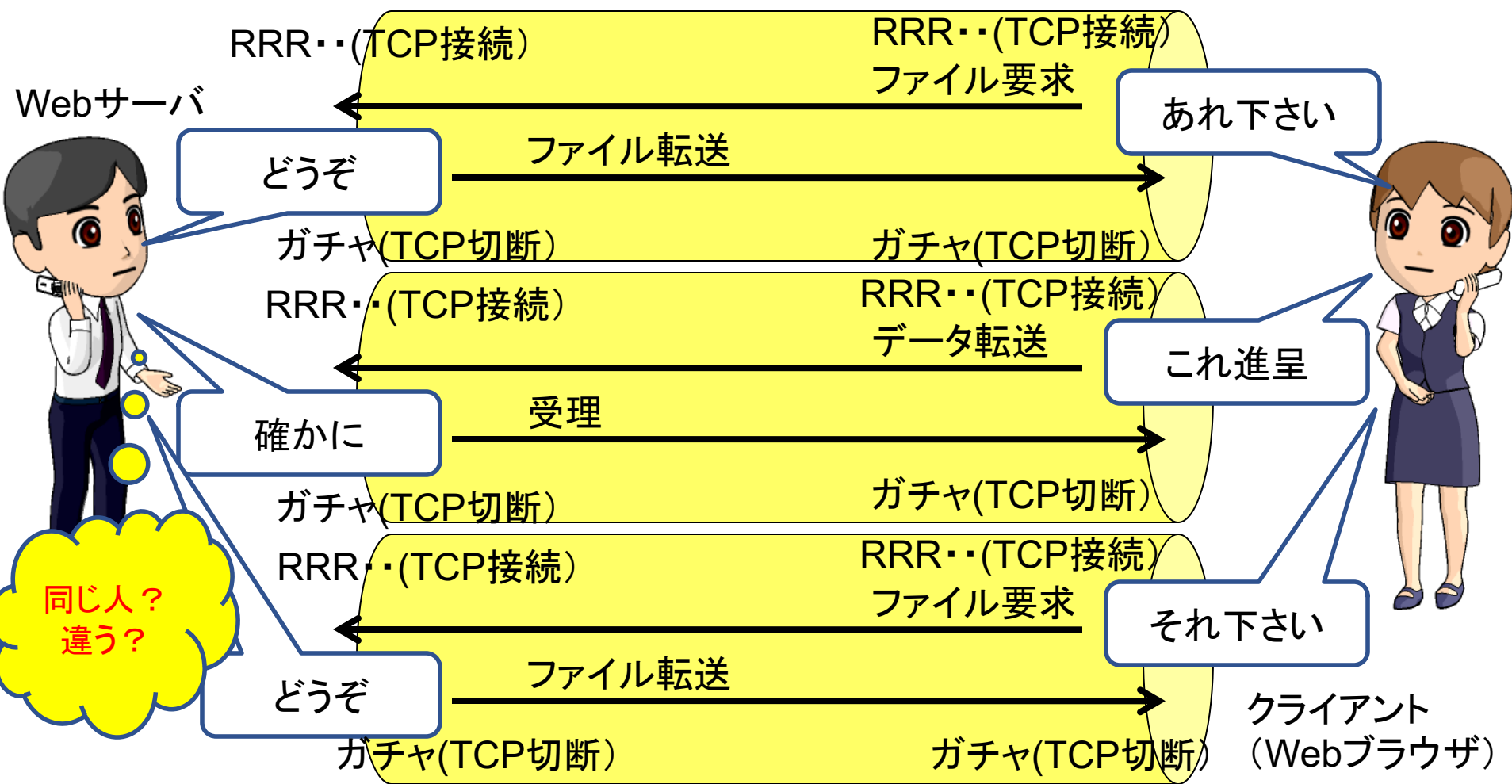
- サーバからの応答に悪意ある内容が埋め込まれる恐れ

- クロスサイトスクリプティング
- キャッシュ偽造

キャッシュ : Cache

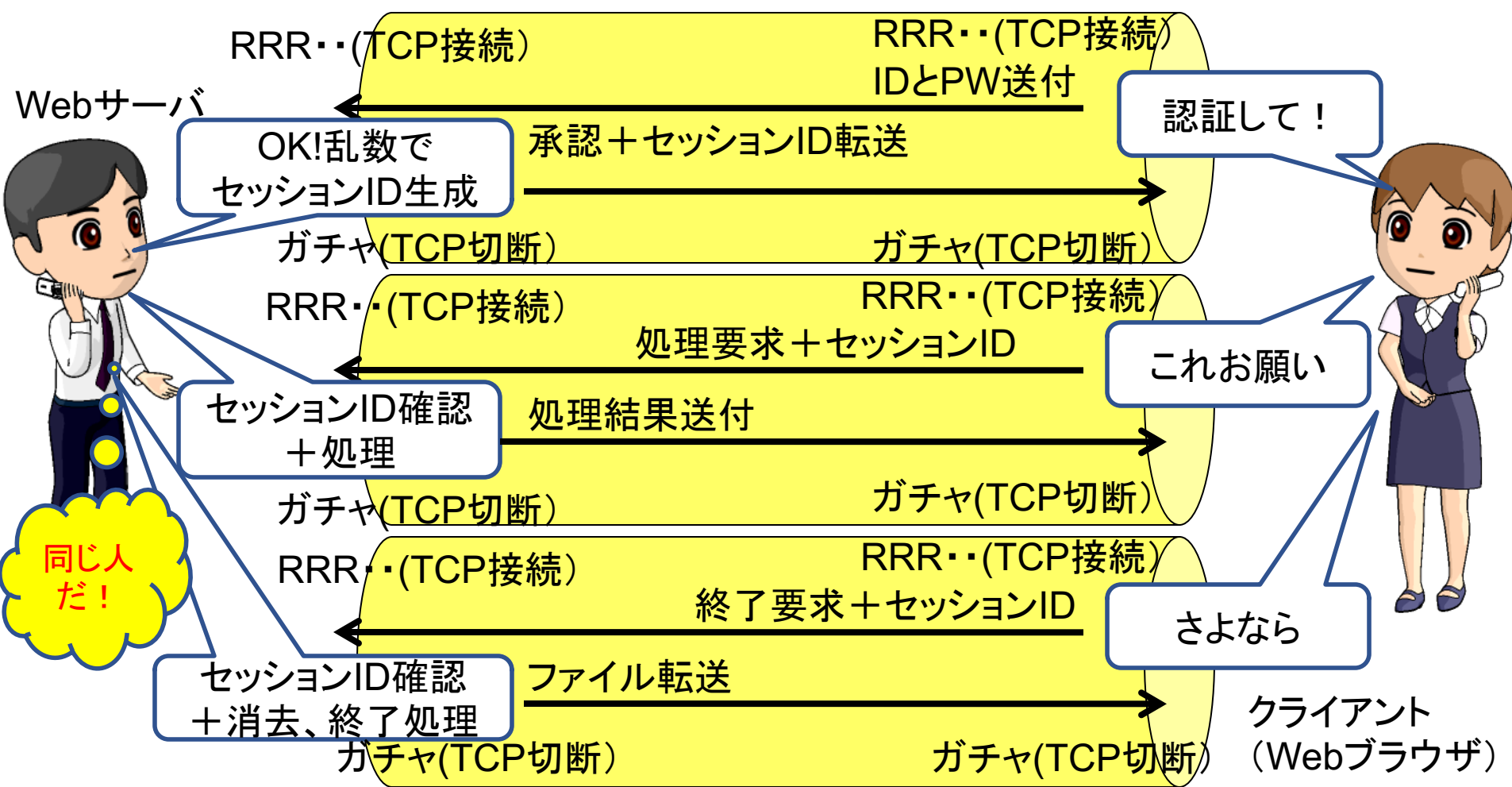
HTTPのおさらい

- 「TCP接続」「ひとつのコマンド実行」「終了」「TCP切断」という流れ



HTTPでのセッション管理

- 複数のTCP接続に対し同一相手であることを保証するためセッションID利用



アクセス制御

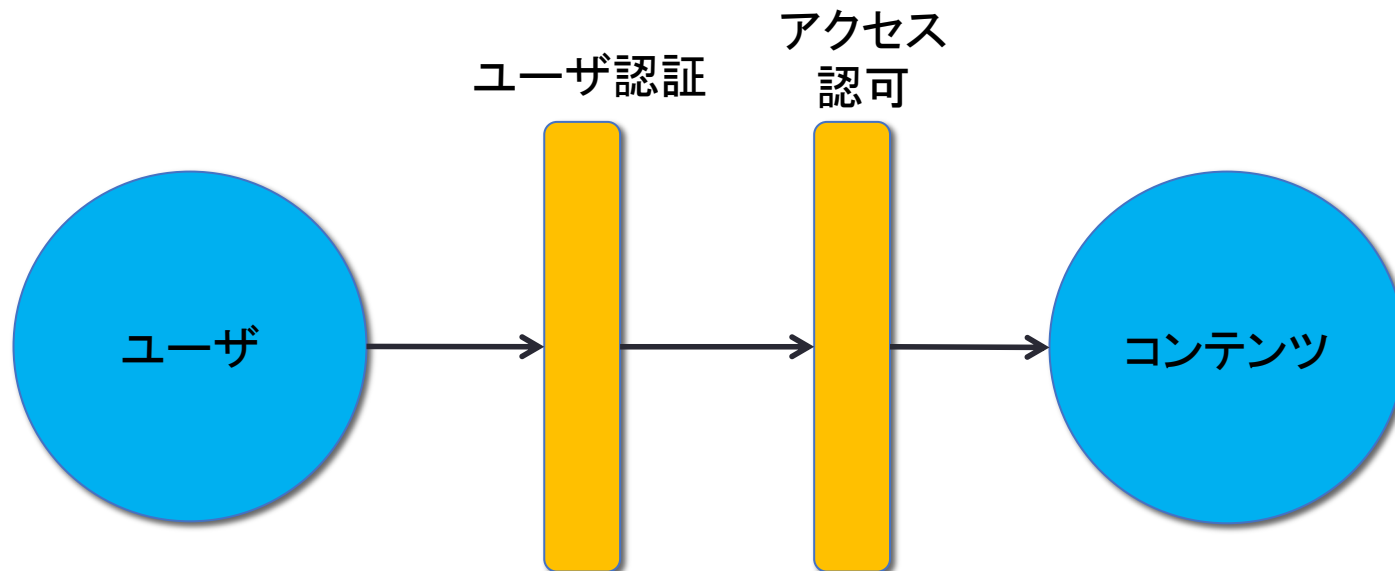
アクセス : Access

- コンテンツへのアクセスの構成要素

コンテンツ : Contents

- ユーザ認証: アクセス要求してきた実体が本人であることを確かめる
- アクセス認可: ユーザ認証された相手に対して, リソースのアクセスを許可あるいは禁止する

リソース : Resource



ユーザ認証で考慮すべきこと

- 常にユーザIDとパスワードを求める
 - ユーザ固有のIDとユーザのみが知っている(はず)のパスワードを照合することにより、まずは本人確認をする
 - 必ずサーバ側で行う
- アカountのロックアウト・パスワードの有効期限
 - ログイン失敗回数が基準値を超えた場合にアカウントをロック
 - 一定期間が過ぎたパスワードを強制的に変更させる
 - いずれも総当たり攻撃への対処
- パスワードフィルタ
 - パスワード発行時に単純なものを防ぐ
- エラーメッセージに気をつかう
 - 攻撃の手がかりになるような情報を返さないようにする
 - 「パスワードが正しくありません」ではなく「ユーザIDまたはパスワードが正しくありません」

パスワード : Password

アカウント : Account

ロックアウト : Lock-out

ログイン : login

ロック : Lock

フィルタ : Filter

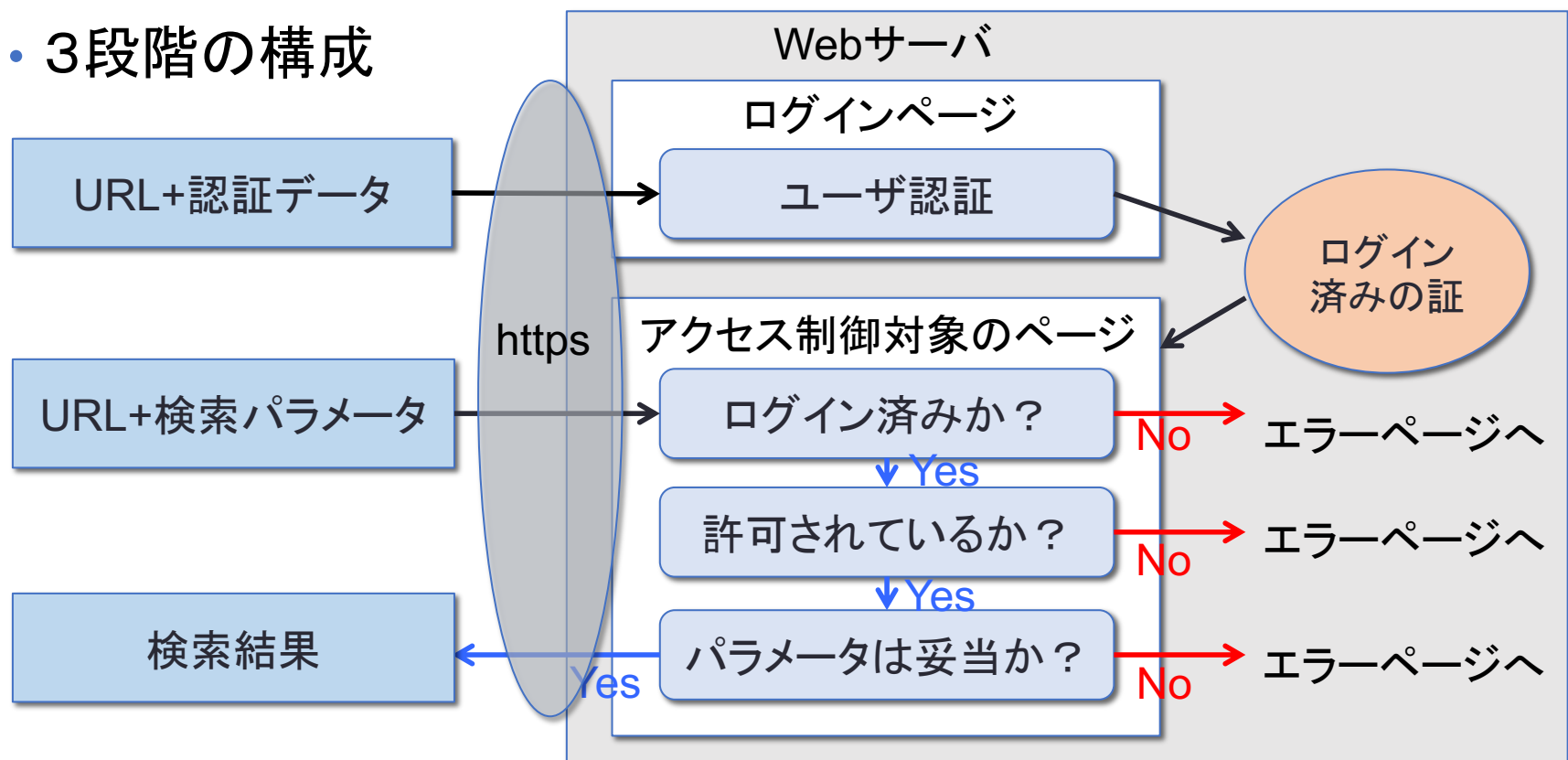
アクセス認可

- 留意点

- ユーザ認証の結果に基づく
- 全アクセスに認可のための仕組みを設ける
 - メニューから隠しただけではURL直接指定でアクセスできてしまう

パラメータ : Parameter

- 3段階の構成



CSRF: クロス サイト リクエスト フォージェリ

- リクエストを偽造し, 重要な処理を実現する
 - 物品の購入, 退会処理, 掲示板への書込み, パスワード変更...



リクエスト強要対策

- ユーザ以外の者が捏造したコンテンツに基づいたHTTPリクエストを、Webアプリケーション側が受け付けなくにする
 - 他者が推測困難な照合情報を用いる
 - セッションIDから導出されたハッシュ値
 - セッション開始時に生成された値
 - ページ毎に毎回異なる値(ページトークン)
- ページトークンを自動生成するFrameworkを用いる
 - Ruby on Rails, Django, CakePHP ...
 - 照合情報の埋め込みが有効になるように設定する
- TLSを用いる
 - 照合情報が傍受されないようにする

ハッシュ : Hash

ページトークン : Page Token

セッション乗っ取り対策

- セッションID強度を高める
 - ランダムで、桁数や文字数の多いものを使用する
 - 同じ値は使い回さない
 - APIで用意されたセッションID発行機能を使うべし
- httpsを適切に用いる
 - httpsを使っているのにセッションIDが漏洩する場合
 - httpのページとhttpsのページの相互の行き来
 - ログイン前のhttpのページでのセッションIDの発行
 - 対策
 - Cookieにsecure属性を付ける(⇒httpsでしか送信されない)
 - httpsのページでは新たにセッションIDを発行する
 - ページ構成を適切に設計する(httpとhttpsの行き来を無くす)
- セッションIDの付け替え, スクリプト注入対策を行う

ランダム : Random

XSS(クロス サイト スクリプティング)

- スクリプト注入攻撃の一種

Webページ
または
HTMLメール

攻撃者

罠
コンテンツ

Webサーバ

アプリケーション
プログラム

望まない
リクエスト

入力の
エコーバック

悪意の
スクリプト

- 偽ページ
 - Cookieの奪取
 - Cookieの植え付け
- 等

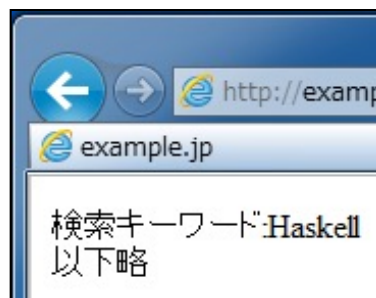


被害者のブラウザ

クロス サイト スクリプティングの例

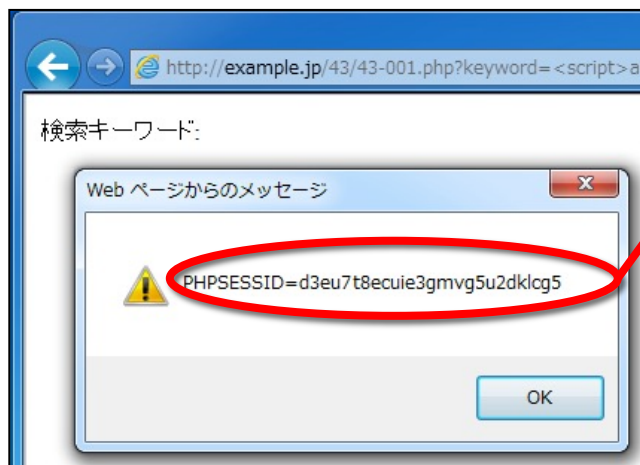
```
<?php session_start(); ログインチェック(略) ?>  
<body>検索キーワード:<?php echo $_GET['keyword']; ?><BR>以下略</body>
```

?keyword=Haskell の場合:



検索キーワードにスクリプトを入れると...

?keyword=<script>alert(document.cookie)</script>



セッションIDが表示される！

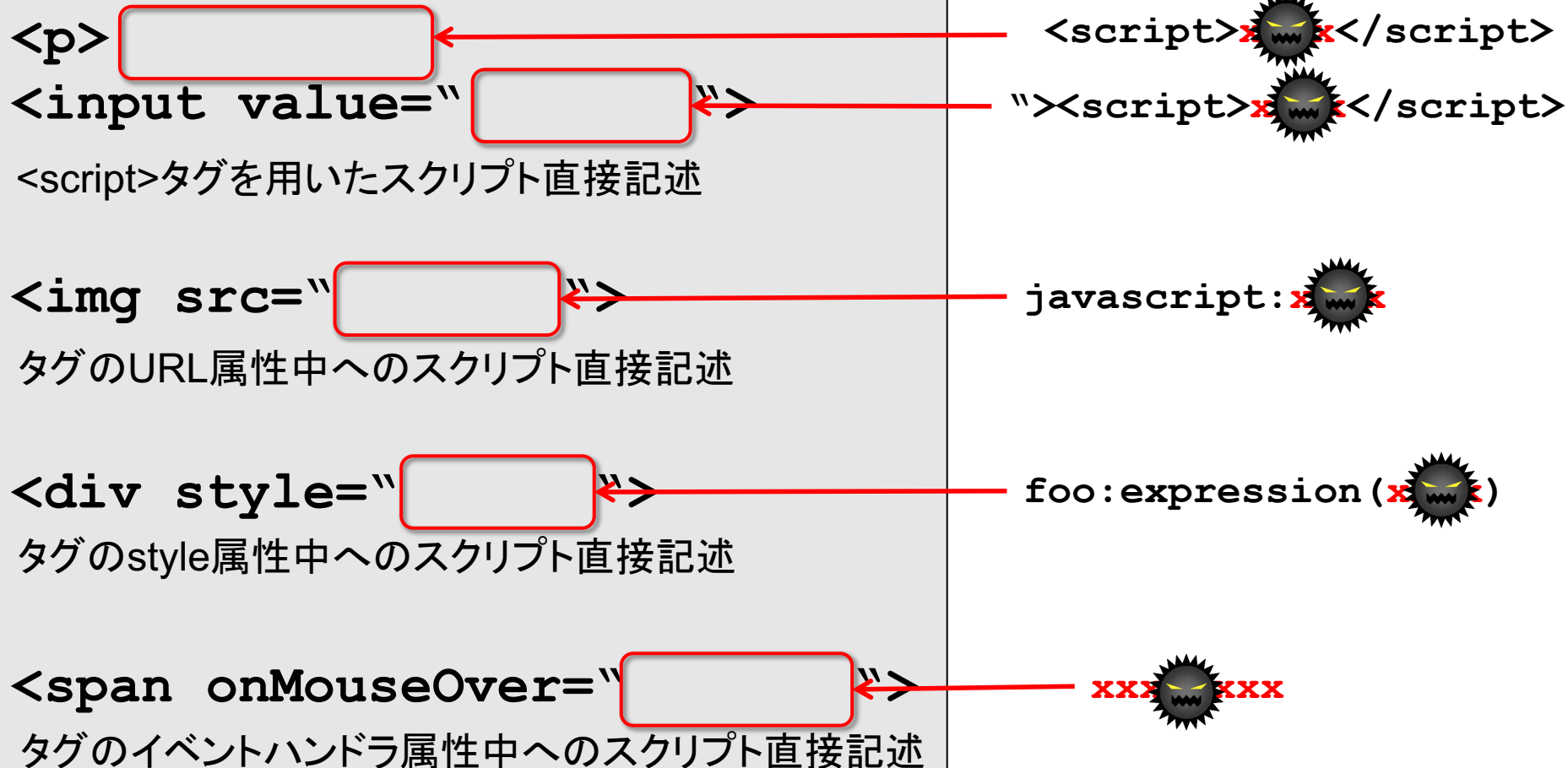
攻撃者はこの仕組みを応用して
セッションIDを盗むことができる:

- 例えばこれがセッションIDを攻撃者の元にメールするスクリプトだったら...

スクリプト インジェクション対策の準備

Script Injection

- まずはHTML文書内でスクリプトが動作する箇所を知る



HTML文書

スクリプトインジェクション対策

ドキュメント : Document

- 基本方針: 攻撃者が送り込む文字列がHTMLドキュメントの一部になり, ブラウザがスクリプトとして解釈することを阻止する
 - HTMLに書き込まれるデータに含まれる特殊文字の置換
 - <, >, “, ‘, &などをそれぞれ対応する実体参照を表現する記法に置き換える:
< → <; > → >; “ → "; ‘ → '; & → &
 - URLを値として持ちうるタグの属性(href, srcなど)やstyle属性, イベントハンドラ, <script>..</script>の内側には動的に値を書き込まない
 - どうしても必要な場合は, 書き込まれた結果がスクリプトとして意味を持つことの無いよう, 慎重に書き込まれる値を制限する必要がある
 - 使用する文字セットを, HTTPヘッダやHTMLドキュメントに必ず明示するようにする
- 対策漏れに注意
 - 出力用のAPIの準備やフレームワークの利用が対策漏れを無くすことにつながる

イベントハンドラ : Event Handler

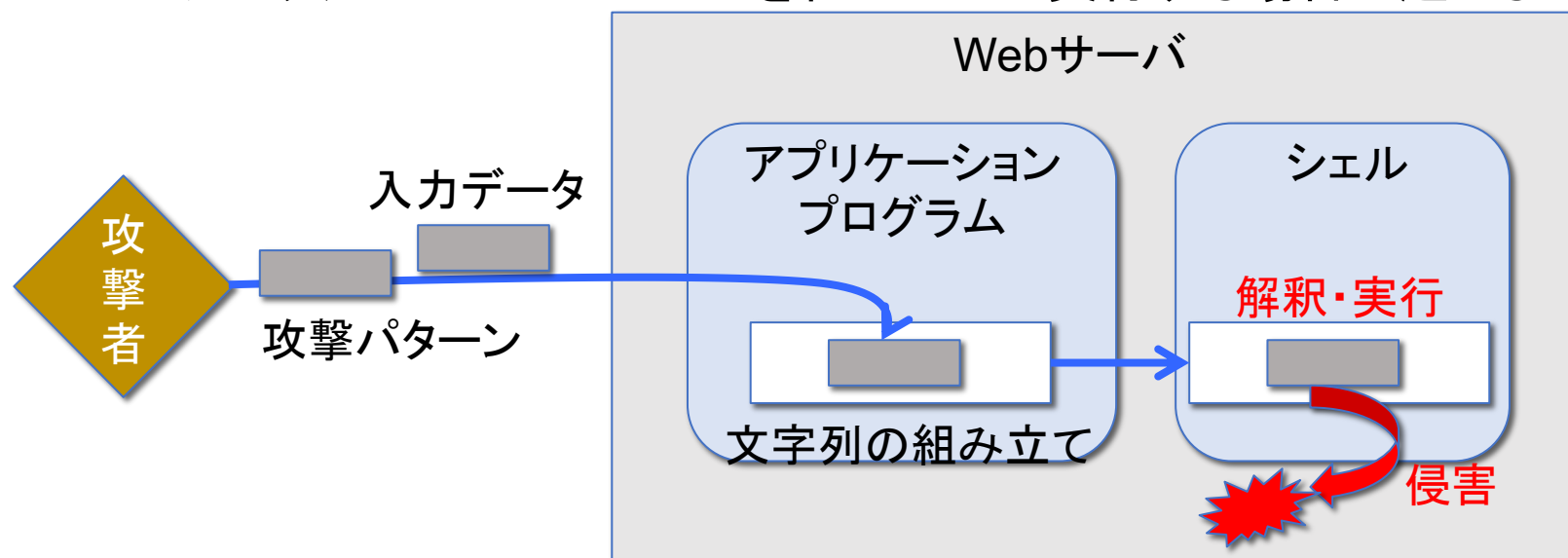
フレームワーク : Framework

コマンド インジェクション

Command Injection

シェルコマンド : Shell Command

- 外部からデータに紛れさせたシェルコマンド文字列を送り込まれてコンピュータを不正に操られてしまう問題
 - Webアプリケーションがコマンドを組み立てて実行する場合に起こる



- 例 (sendmailコマンドでメールを発信するプログラム):

```
$to_address=cgi->param{ 'to_address' };  
$message_file="...";  
system("sendmail $to_address<$message_file");
```

\$to_addressに "; shutdown #" を投入するとシステムが停止

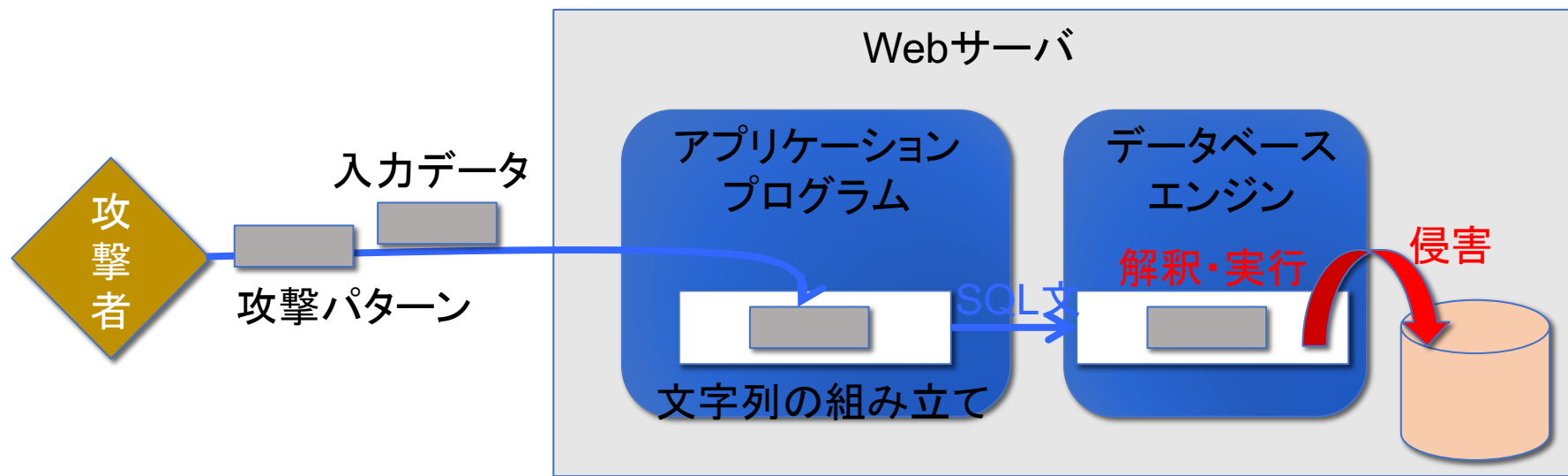
コマンドインジェクション対策

- うっかりシェルを動かさない
 - 気付かぬうちにAPIがシェルを起動することがある
 - Perl, PHP, Rubyなどのsystem(), exec()関数など
- 別プログラムの起動を避ける
 - 別プログラムの起動を避けた設計を行う。言語の実行時ライブラリで解決できる場面で別プログラムを用いない
- 別プログラムの起動を避けられないときは
 - シェルが用いられないAPIを選ぶ
 - 特殊記号の排除
 - 環境変数のリセット
 - chrootによるアクセス可能なファイル領域の制限

SQLインジェクション

シェルコマンド : Shell Command

- 外部から特殊な文字列が送られることによってデータベースに不正にアクセスするSQL文が組み立てられてしまう



- ログイン判定プログラムの例:

```
SELECT uid FROM account WHERE uid='ユーザID' AND pw='パスワード'  
` OR 1=1 -- が埋め込まれると...
```

```
SELECT uid FROM account WHERE uid='' OR 1=1 --...
```

誰でもログイン可能に！

常に成り立つ コメント

SQLインジェクション対策

ロジック : Logic

- SQLを文字列として扱わず、ロジックとしてプログラムの中に書けるプログラミング言語を使う
 - .NET LINQ(.NETの各言語で利用可能な統合クエリ言語)
 - PL/SQL (Oracleデータベース用プログラミング言語)
 - PL/pgSQL (PostgreSQLデータベース用プログラミング言語)
- .NET LINQの例

```
using System.Linq;  
...  
AccountDataContext adc = new AccountDataContext();  
IEnumerable<Account> query =  
    from uid in adc.Account where uid=...(入力値);
```

- クエリの構造は実行時に組み立てられるのではなく、コンパイル時に決まる

コンパイル : Compile

SQLインジェクション対策 —コーディングでの対策

コーディング : Coding

チェック : Check

- 入力値チェックの徹底

- 入力値の要件について, 文字種, 桁数などの仕様を徹底的に絞る
 - ユーザIDは8文字以内の英数字⇒ ‘ や - などの文字ははじかれる

- Prepared Statementの使用

プレースホルダ : Placeholder

- Prepared Statement : プレースホルダ(予約場所)を含む解析済みのSQL文. プレースホルダに入力値を割り当てるAPI(バインドメカニズム)を持つ

バインドメカニズム : Bind mechanism

- Javaの例:

```
String uid=..., pw=...; //ユーザが入力した値
Connection c = ...(データベースオブジェクト)
PreparedStatement st =
    c.prepareStatement(
        "SELECT uid FROM account WHERE uid=? AND pw=?");
st.setString(1,uid);
st.setString(2,pw);
```

データベースオブジェクト : Database Object

入力検査漏れ対策

- 全入力項目を網羅的に検査する
 - ユーザが入力しなくてもプログラムがブラウザから受け取る値に注意
 - Checkbox, Radio ButtonなどのForm項目
 - Cookie:, Referer: などのHTTPヘッダ項目
- 入力検査をクライアント(ブラウザ上のJavaScript)だけに任せない
 - 悪意あるユーザはブラウザ上での検査ロジックを迂回できる
- プログラミング言語による文字列から数値への変換に注意
 - 言語によっては整数値の範囲の検査において侵入を許す場合がある

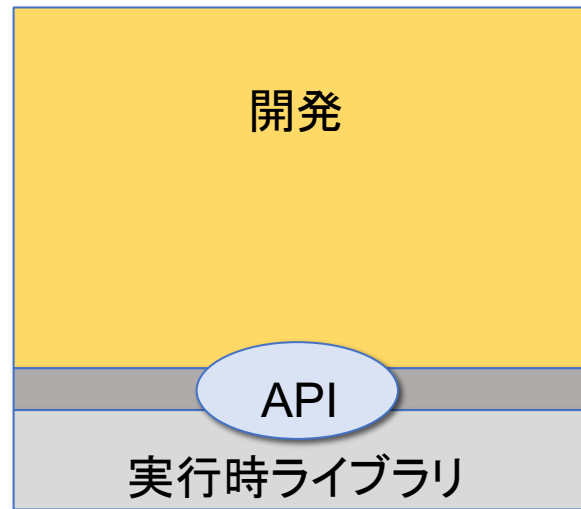
```
if ($a<100 || $a>200) {  
    エラー処理  
}  
本来の処理
```

エラー : Error

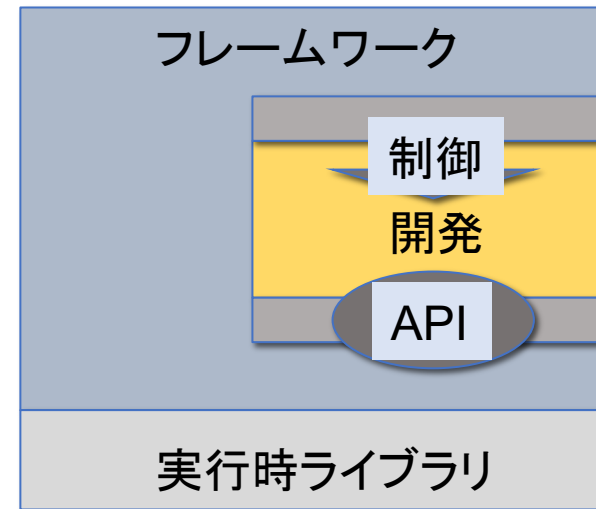
`$a=123<script>悪意</script>`
でもエラー処理に行かない言語がある！

Web アプリケーション フレームワーク (Web Application Framework)

- Webアプリケーションの骨組みを提供し、開発を助けてくれる仕組み



従来の開発



フレームワークを利用した開発

ライブラリとの違い

ライブラリ: Library

- ライブラリ: ユーザコードから呼び出されるもの
- フレームワーク: アプリケーションの実行(初期化・実処理・終了)の流れを全て管理, 要所要所でユーザコードを呼び出す

フレームワーク導入の利点

- 開発生産性の向上

- 根幹の設計や基盤部分の実装をフレームワークに任せられるので、大規模な再利用が可能

- 保守性の向上

- コーディングのルールを強制されるので、不具合や仕様変更が生じた場合に該当箇所を特定しやすい

- 一定以上の品質の維持

- フレームワーク側のコードは主にオープンソースで、多くの人間の目に晒され、テストされており、一般的に高い品質を期待できる。そのようなコードを大規模に再利用しているため、品質が維持できる

様々なフレームワーク

プログラミング言語	フレームワーク名
Java	Apache Struts, Spring, Grails, Apache Tapestry, Apache Shiro, etc.
Ruby	Ruby on Rails, Sinatra etc.
PHP	CakePHP etc.
Python	Django, TurboGears, Flask etc.
Perl	Catalyst, Interchange, Mason etc.

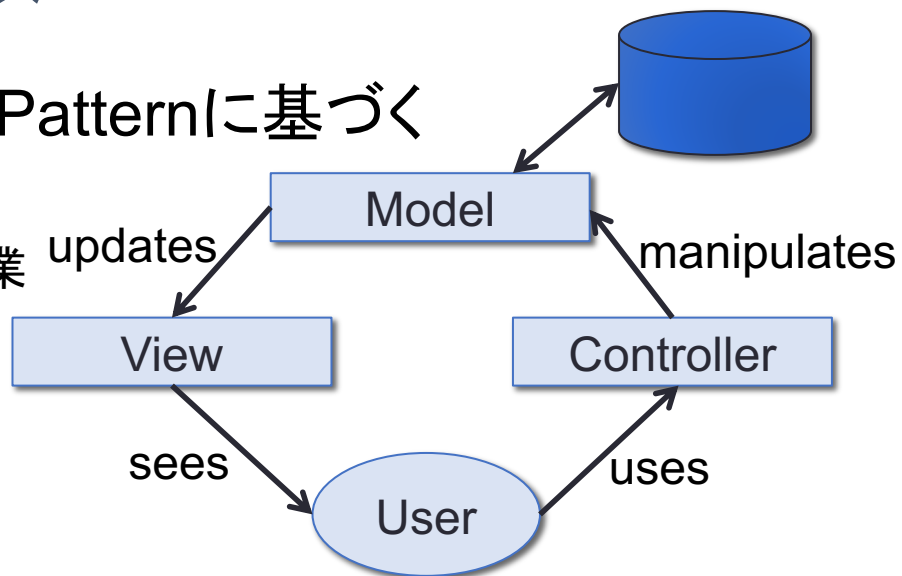
Ruby on Railsの特徴

- MVC (Model-View-Controller) Patternに基づく

- 各構成要素の明確な分離

- ProgrammerとDesignerとの並行作業

- 保守の容易さ



- DRY (Don't Repeat Yourself) = 同じ記述を繰り返さない

- 例: データベースのSchemaを設定ファイルとして持たない.

Tableを作ればRailsが自動的に認識

- CoC (Convention over Configuration) = 設定よりも規約

- Railsが予め用意している名前付けのルール

ルール: Rule

- 例: users(複数形)テーブルを読み込むためにはUser(単数形)クラスを使用

テーブル: Table

クラス: Class

Railsの構造

スケルトン : skeleton

- rails newコマンドでアプリケーションのスケルトンを作成.
様々な作業を自動化
 - アプリケーションコードの生成: プログラマはこのコードを拡張してアプリケーションを開発する
 - ディレクトリ構造の生成

ディレクトリ: directory
 - スタイルファイルの生成

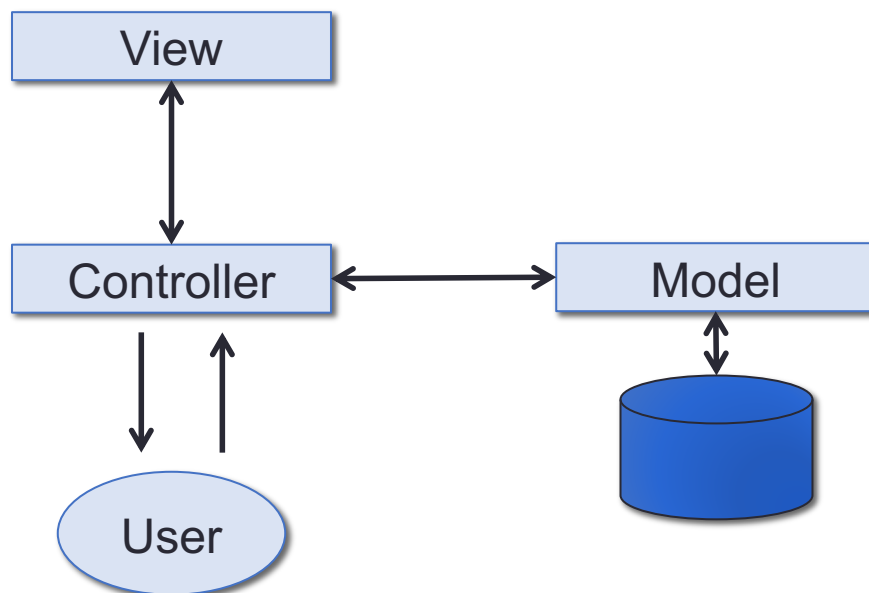
スタイル: Style
 - 必要なパッケージのインストール
 - パッケージ: Package
 - インストール: install
- アプリケーションのディレクトリ構造は共通 (app/ 以下)
 - models/ : MVCのModelに相当するクラスが置かれる
 - views/ : MVCのViewに相当するクラスが置かれる
 - controllers/ : MVCのControllerに相当するクラスが置かれる

本当はデモを見せたかったが...

```
$ rails new first_app
  create
  create README.rdoc
  create Rakefile
  create config.ru
  ..
  create app
  create app/assets/images/rails.png
  create app/assets/javascripts/application.js
  create app/assets/stylesheets/application.css
  create app/controllers/application_controller.rb
  create app/models
  create app/views/layouts/application.html.erb
  create config/routes.rb
  ..
```

Controller

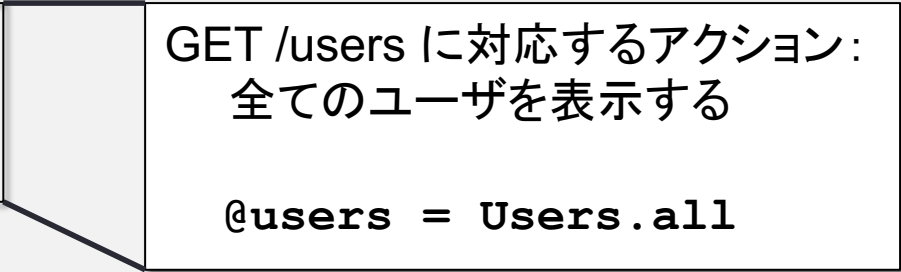
- ブラウザから送られたHTTPリクエストを受け取るmodule
- リクエストの内容に応じて、対応するviewによって生成されたHTMLファイルをブラウザに送り返す
 - 必要に応じて、データベースとの通信を担当するmodelと対話
- いわゆる「Business Logic」に相当



Controllerの構造

- Pageや内部処理に対応したActionの実装

```
class UsersController < ApplicationController
  ..
  def index
    ..
  end
  def show
    ..
  end
  def new
    ..
  end
  def edit
    ..
  end
  ..
end
```



GET /users に対応するアクション:
全てのユーザを表示する

@users = Users.all

Model

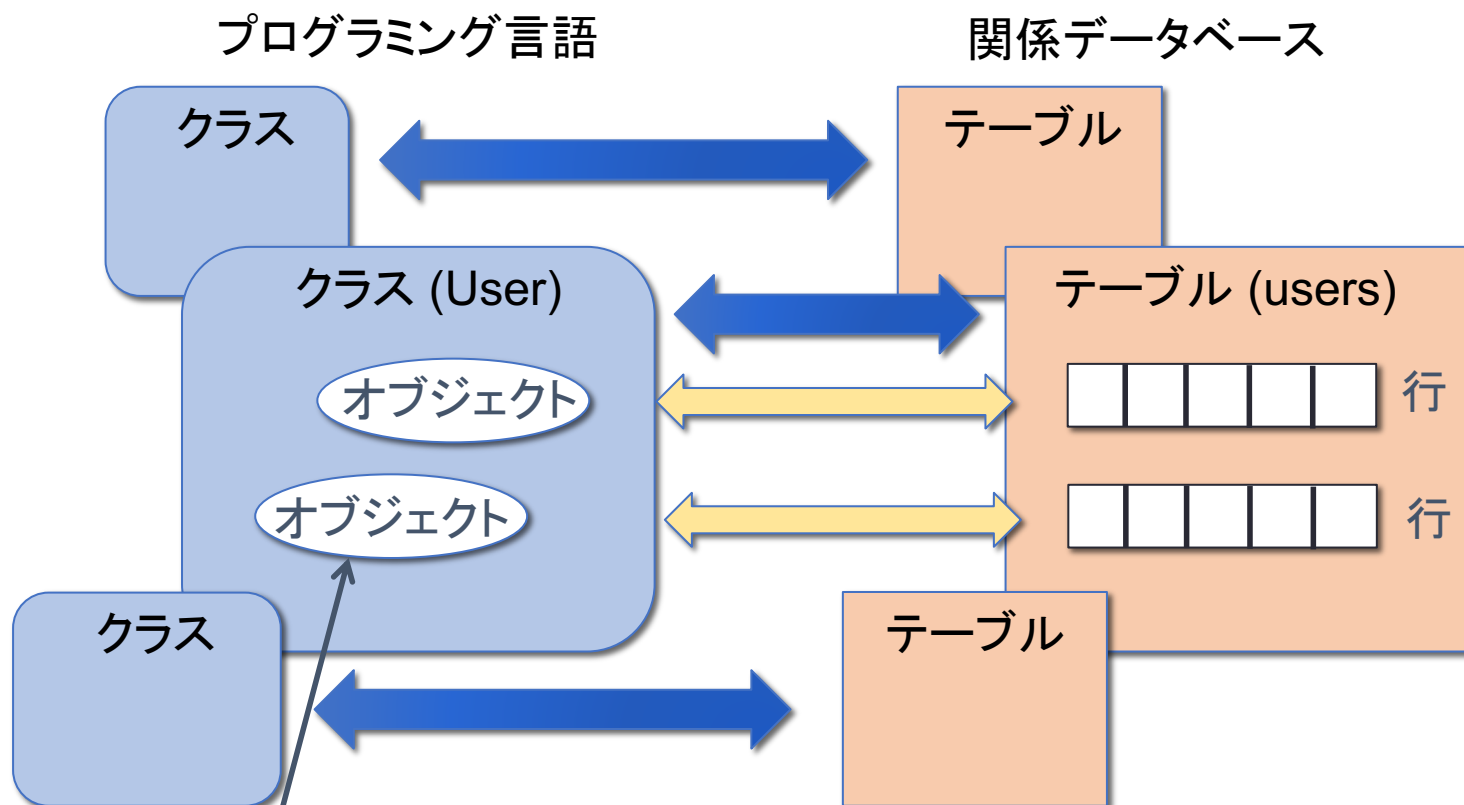
- modelを設計し, classを生成
 - `rails generate model User name:string email:string`
 - nameとemailという属性(いずれも文字列)を持つUserクラスが生成される
- データベースへの反映(Migration)
 - `bundle exec rake db:migrate`
 - Userクラスと同等の関係データベースの users テーブルが生成される
 - Convention over Configuration
 - 名前付け規約として, User(単数形)クラスからusers(複数形)テーブルが作られる
 - 設定ファイルは必要ない
- クラスとテーブルの対応付け: O/R Mapperのメカニズム

Mechanism

O/Rマップ (Object-Relational Mapper)

インスタンス : Instance

- 関係データベースへのアクセスを, オブジェクト指向言語のクラスとインスタンスの操作を通じて行うライブラリ



クラスやオブジェクトへの操作を通じてデータベースを操作する

View

- HTMLの中にRubyの式が埋め込まれたTemplate

index.html.erb

```
<h1>Listing users</h1>
<table>
  <tr>
    <th>Name</th>
    <th>Email</th>
  </tr>
</table>
<% @users.each do |user| %>
  <tr>
    <td><%= user.name %></td>
    <td><%= user.email %></td>
  </tr>
<% end %>
</table>
```

usersテーブルを表す変数.
ここでも名前付け規約が使われる

各行の属性. やはり名前付け規約

ViewはHTMLに変換して, Controllerがブラウザに送り返す

Ruby on Rails 上のセキュアプログラミング

- 著名な脆弱性対策
 - SQL注入対策: Model上でのO/RマップとQuery APIによる対策
 - スクリプト注入対策: View上での出力における特殊文字のEscape
 - リクエスト強要対策: ControllerにおけるロジックにToken自動埋め込み, 自動照合の処理
- 入力値対策
 - 入力値検査
 - 入力値追跡
- 計画的なセキュリティ機能性
 - ユーザ認証
 - アクセス認可

RailsにおけるQuery(クエリ)の発行

- ARelと呼ばれるデータベース・クエリ用のドメイン固有言語 (DSL) (Rails 3から)

- 簡単な例

```
User.where(:name => "Tetsutaro")
```

SELECT * FROM users WHERE name = "Tetsutaro" と同等

- LIMITを付けたい場合:

```
User.where(:name => "Tetsutaro").limit(20)
```

- OR検索

```
name = User.arel_table[:name]
```

```
User.where(name.eq("Taku").or(name.eq("Tetsutaro")))
```

検索条件文を先に生成

- SQL構文を反映したAPIにより, 意図しない構造をしたSQL文が生成されることを防ぐ

テンプレートエンジンTemplate EngineによるWebページの生成

- 出力ページのテンプレートにRubyの変数や式を埋め込み, それらの結果をWebページに合成して出力する仕組み
 - `<%= user.name %>`
 - スクリプト注入攻撃の対象となる特殊文字は自動でエスケープ処理されてから出力される
- Rails 2 では以下のように hメソッド(html_escapeの別名)を明示的に使用する必要がある
 - `<%= h user.name %>`

スクリプトはあらゆる箇所に埋め込まれ得る

- 「プログラムが書き込む」箇所(<%= %>内)は全てEscape

`<p><%= %></p>`
`<input value="<%= %>">`
<script>タグを用いたスクリプト直接記述

`<script><%= %></script>`
`"><script><%= %></script>`

``
タグのURL属性中へのスクリプト直接記述

javascript<%= %>

`<div style="<%= %>">`
タグのstyle属性中へのスクリプト直接記述

foo:expression(<%= %>)

`<span onMouseOver="<%= %>">`
タグのイベントハンドラ属性中へのスクリプト直接記述

<%= %>

トークン(Token)の自動埋め込み

- HTMLフォームを生成するmethod(form_for)を使用すると、リクエスト強要対策のトークンが自動で埋め込まれる

```
<%= form_for(@users) do |f| %>
```

```
  <%= f.label :name %>
```

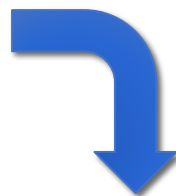
```
  <%= f.textfield :name %>
```

```
  <%= f.label :email %>
```

```
  <%= f.textfield :email %>
```

```
  <%= f.submit "Register" %>
```

```
<% end %>
```



Viewが勝手にauthenticity_token
を埋め込む:

```
<input name="authenticity_token"  
      type="hidden"  
      value="....." />
```

フォームを埋めて、submitしてブラウザから送られるリクエストに、authenticity_tokenが含まれる。

このトークンはRailsによって照合が行われ、合わないエラーになる

トークンの照合

- Railsが生成したApplicationControllerクラスに記述された `protect_from_forgery` メソッドによってリクエスト強要対策が行われる
- `before_filter` というRailsに備わる機能によって、コントローラクラスの全メソッド呼出の前に以下のチェックが行われる
 - 設定ファイルでリクエスト強要対策を無効にしているか
 - リクエストのHTTPメソッドがGETまたはHEADか
 - サーバが持っている値とリクエスト中のトークンの値を照合した結果、正しいと判断されるか
- 全てに当てはまらなかった場合はエラーにする

```
class ApplicationController < ActionController::Base
  protect_from_forgery
end
```

```
def protect_from_forgery(options = {})
  ..
  prepend_before_action :verify_authenticity_token
  ..
end
```

```
def verify_authenticity_token
  if verify_request?
    // エラー処理
  end
end
```

```
def verify_request?
  !protect_against_forgery? || request.get? || request.head?
  valid_authenticity_token?(session, form_authenticity_token) ||
  valid_authenticity_token?(session, request.headers['X-Auth-Token'])
end
```

フレームワーク(Framework)まとめ

- フレームワークを用いることでWeb Applicationは...
 - 開発の効率が上がる
 - 開発後の保守(Maintenance: メンテナンス)が楽になる(保守性向上)
- ただしフレームワークを用いることでWeb Applicationが完全にSecureになる(安全になる)わけではない
 - そのためにもセキュリティ上の脆弱性が「何故」発生するかその原理を理解しておくことが大切
- フレームワーク内で何が起きているかを知る
 - 「よく分からないけどフレームワーク使ってれば安全」は間違い
 - フレームワークが利用できない場合もある
 - フレームワークに脆弱性があるかもしれない
 - 中身を知らないと正しく使いこなせない
 - form_for を使わずにフォームを作ってしまうとリクエスト強要対策に引っかかり、正しいリクエストもエラーになる
- Ruby on Rails チュートリアル (by Michael Hartl)
 - <http://www.railstutorial.org> (日本語版 <http://railstutorial.jp>)