

WEBアプリケーションの仕組み (3-2, 3-3)

1721155

平嶋 夏樹



パーシャル、アプリケーション クッキー

includeとパーシャル(P140～)

WEBページの表示にパーシャルというテクニックがある

パーシャルとは、

テンプレートの一部を部品化する機能

パーシャルについて

- テーブルの表示 → 汎用性が高い！

for文などで表示する方式がよく使われる

`<table>`内の`<tr>~</tr>`をテンプレート化する

パーシャルを使うとテーブルのカスタマイズしやすくなる

パーシャルの作成

エディタ等で「data_item.ejs」を作成する
このファイルにテーブル内の表示を記述する
リスト3-7を入力して保存してください。

リスト3-7(p141)

```
<tr>
  <th><%= key%></th>
  <th><%= val[0]%></th>
</tr>
```

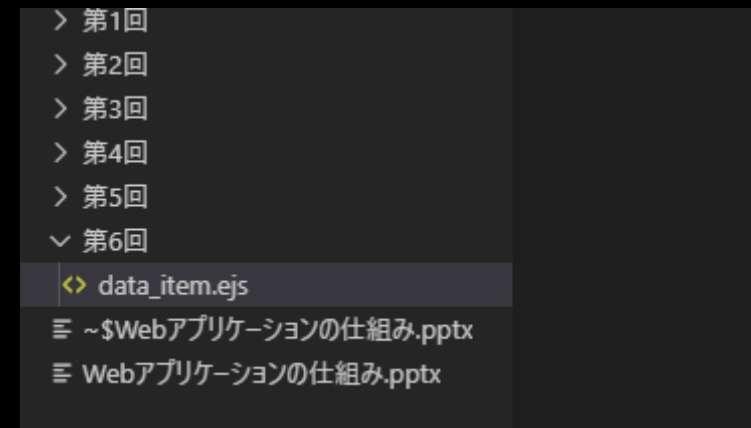


図1 ファイル作成

パーシャルの作成

- リスト3-7は、テーブル内の構造を部品化したものである。
- この部品にデータを与えることで、このパーシャルを呼び出したテンプレートに追記する

includeでパーシャルを読み込み

- index.ejsのテーブルのfor文内を下の文に置き換える
- `<%- include('data_item', {key: key, val: [data[key]]}) %>`
- これはdata_item.ejsを読み込み、keyとdata[key]を渡している。

includeの構文

```
<%- include( ファイル名, { ...受け渡すデータ... } ) %>
```

includeでパーシャルを読み込み

- index.ejsの<body>内をリスト3-8に書き換えてください。(p142-143)
- 重要な箇所・・・<table>内のinclude文

リスト3-8

```
<div role="main">
  <p><%=content %></p>
  <p>
    <table>
      <% for(var key in data) { %>
        <%- include('data_item', {key: key, val: [data[key]]}) %>
      <% } %>
    </table>
  </p>
</div>
```


補足#1 <%= %>と<%- %>の違い(1/3)

- <%= %>と<%- %>は、使い方はほぼ同じだが場合によっては使い分けが必要
- 前者は与えられた値をエスケープして表示する為、タグ等はテキストとして表示される
- 後者は値をエスケープしないため、タグが機能する。ユーザが値を入力する場合は攻撃される場合があるため使用しない。
- 実際にどうなるのかを試した。ソースコードを次のページ、実行結果をその次のページに示した

エスケープとは: HTMLで'<'等の意味を持った記号を表示したい場合にこれを対象付けた文字列を置換することでブラウザが意図しない動作をしないようにしつつ希望の文字列を出力するために行う処理
ユーザが悪意のあるタグを埋め込む等の攻撃を対策するためによく使われる。

例: <p>hoge</p> は<it;p>hoge</p>のように変換する。

補足#1 <%= %>と<%- %>の違い(2/3)

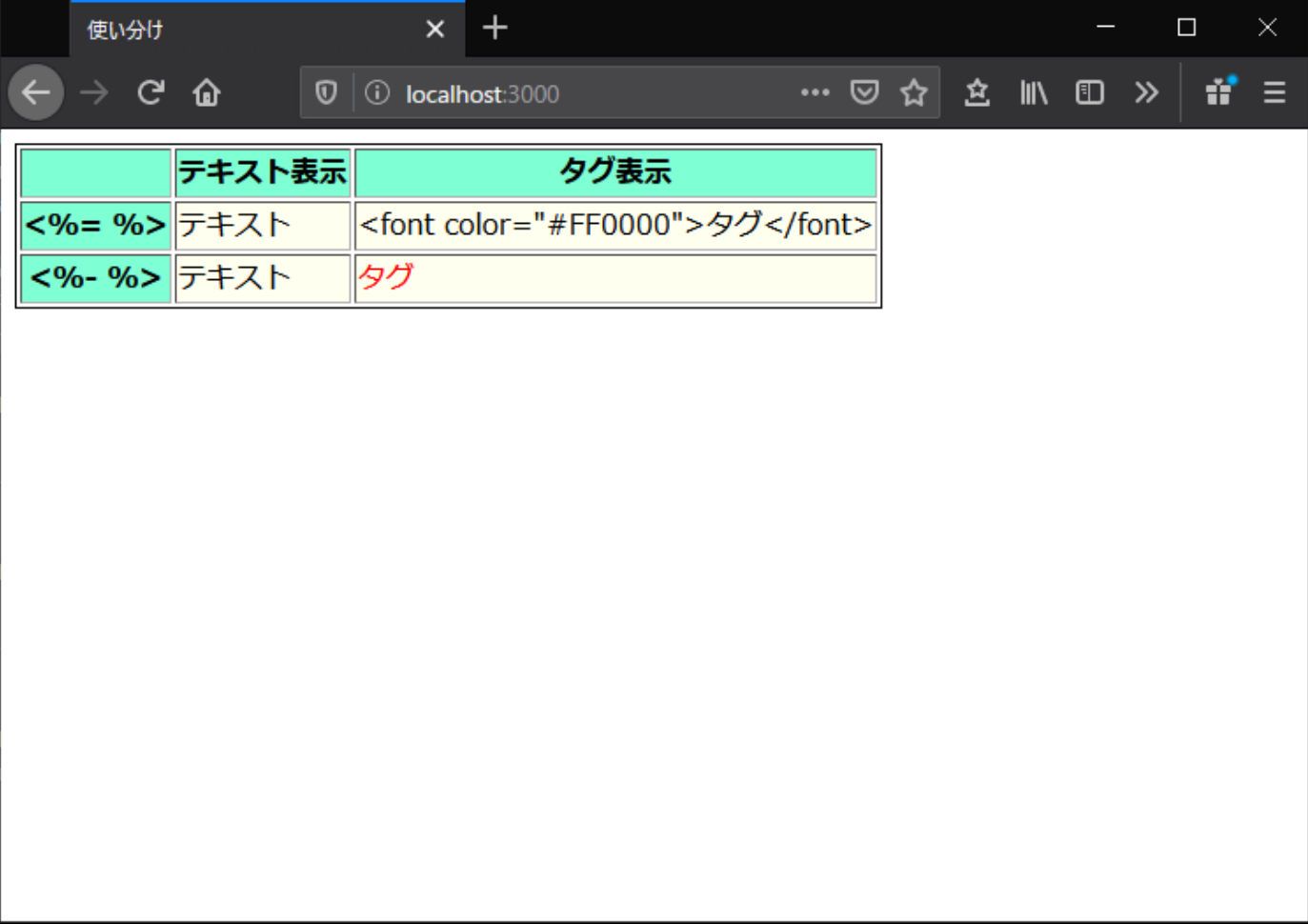
- index.ejs(出力部のみ)

```
<table border="1">
  <tr>
    <th> </th>
    <th>テキスト表示</th>
    <th>タグ表示</th>
  </tr>
  <tr>
    <th>&lt;%= %&gt;</th>
    <td><%=text %></td>
    <td><%=tag %></td>
  </tr>
  <tr>
    <th>&lt;%- %&gt;</th>
    <td><%-text %></td>
    <td><%-tag %></td>
  </tr>
</table>
```

- app.js(値受け渡し部のみ)

```
var content = ejs.render(index_page, {
  text: "テキスト",
  tag: '<font color="#FF0000">タグ</font>'
});
```

補足#1 <%= %>と<%- %>の違い(3/3)



	テキスト表示	タグ表示
<%= %>	テキスト	タグ
<%- %>	テキスト	タグ

図2 app.jsの実行結果

filenameでパーシャルファイルを指定


- 最後にapp.jsのresponse_index関数を修正する。
- P142-143のリスト3-9のように書き換えてください。

• これは、filenameという値でファイル名を渡している。

リスト3-9

これを忘れるとパーシャルがうまく利用できない

```
// indexのアクセス処理
function response_index(request, response) {
  var msg = "これはIndexページです。"
  var content = ejs.render(index_page, {
    title: "Index",
    content: msg,
    data: data,
    filename: 'data_item'
  });
  response.writeHead(200, { 'Content-Type': 'text/html' });
  response.write(content);
  response.end();
}
```



実行してみる#1

- 実際にapp.jsを実行してみてください。

図3のようになりましたか？

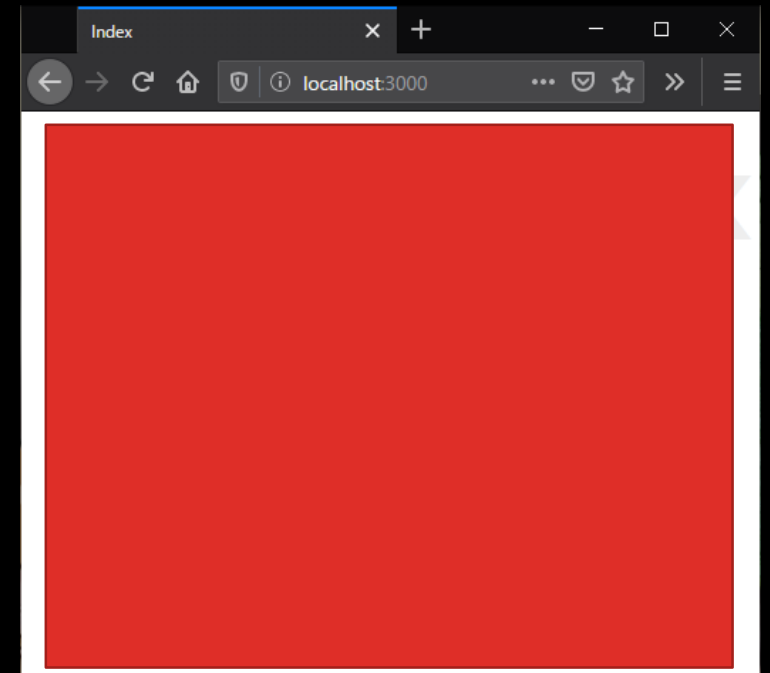


図3 実行結果

パーシャルを書き換える

- テーブルの表示ができることを確認したら、「data_item.ejs」をリスト3-10(p143-144)のように変更してください

リスト3-10

```
1  <tr>
2    <table>
3      <tr><th><%= key%></th></tr>
4      <% for(var i in val){ %>
5        <tr><td><%= val[i]%></td></tr>
6      <% }%>
7    </table>
8  </tr>
```

実行してみる#2

- 実際にapp.jsを実行してみてください。

図4のようになりましたか？

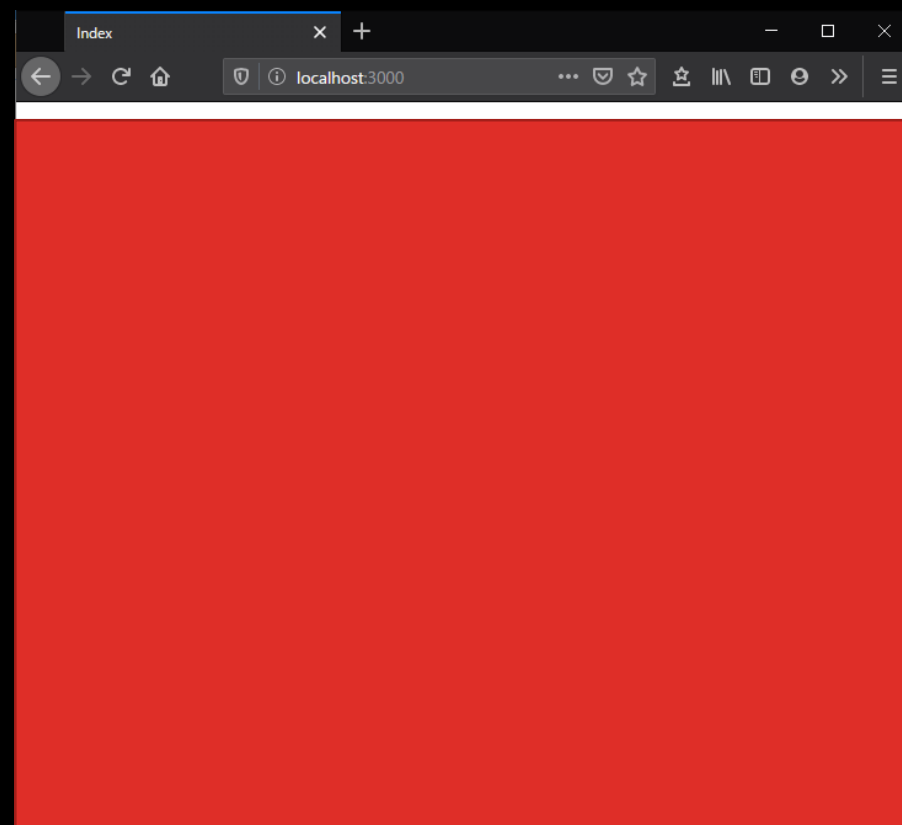


図4 実行結果

別のデータを表示させる

- パーシャルは工夫すれば様々なデータの表示にも利用できる。
- other.ejsを利用するため、
response_otherをリスト3-11 (p146)のように変更してください。

リスト3-11

```
var data2 = {  
  'Taro': ['taro@yamada', '09-999-999', 'Tokyo'],  
  'Hanako': ['hanako@flower', '080-888-888', 'Yokohama'],  
  'Sachiko': ['sachi@happy', '070-777-777', 'Nagoya'],  
  'Ichiro': ['ichi@baseboll', '060-666-666', 'USA']  
}
```

```
// ★otherのアクセス処理  
function response_other(request, response) {  
  var msg = "これはOtherページです。";  
  var content = ejs.render(other_page, {  
    title: "Other",  
    content: msg,  
    data: data2,  
    filename: 'data_item'  
  });  
  response.writeHead(200, {'Content-Type': 'text/html'});  
  response.write(content);  
  response.end();  
}
```

テンプレートの修正

- テンプレートother.ejsの<body>内をリスト3-12のように修正してください。(誤植注意)

リスト3-12

```
<body>
  <header>
    <h1><%=title %></h1>
  </header>
  <div role="main">
    <p><table>
      <% for(var key in data) {%>
        <%- include('data_item', {key: key, val:data[key]})%>
      <% }%>
    </table></p>
  </div>
</body>
```

実行してみる#3

- 実際にapp.jsを実行して、<http://localhost:3000/other>にアクセスしてみてください。

図5のようになりましたか？

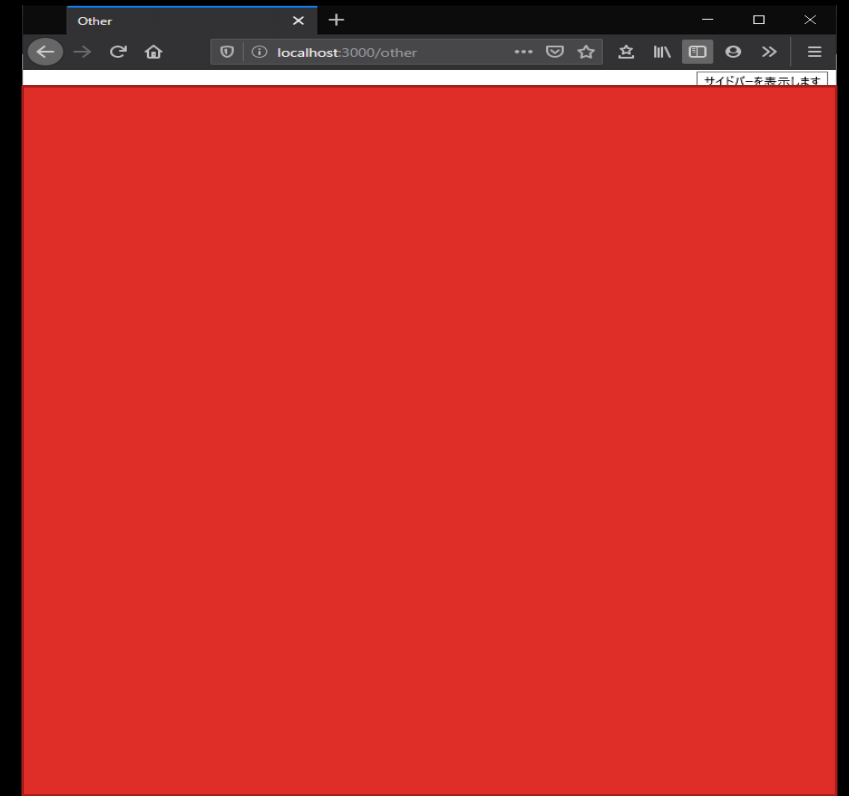


図5 実行結果

アプリケーション変数

- Webで使われるデータは大まかに2つある
- 固有データ
- 共有データ

共通データ

- 共通するデータはapp.jsのグローバル変数dataのように誰がアクセスしても同じ値であるもの
- Node.jsは常にプログラムが実行されているため、グローバル変数はその値が常に保持されてる → 誰がアクセスしても同じ

メッセージの伝言ページを作る

- 共通データの例としてメッセージを置いておく簡易伝言ページを作ってみる
- index.ejsをリスト3-13のように修正してください。
- このdata.msgの部分にデータを渡すことで表示します。

リスト3-13

```
<header>
  <h1><%=title %></h1>
</header>
<div role="main">
  <p><%=content %></p>
  <p><table style="width: 400px;">
    <tr><th>伝言です！</th></tr>
    <tr><td><%= data.msg %></td></tr>
  </table></p>
  <p>
    <form method="post" action="/">
      MESSAGE <input type="text" name="msg">
      <input type="submit" value="送信">
    </form>
  </p>
</div>
```

response_indexを修正する

- app.jsの修正をする。
- response_indexとグローバル変数dataをリスト3-14のように修正してください。

リスト3-14

```
// 追加するデータ用変数
var data = {
  msg: 'no message...'
};
```

```
// indexのアクセス処理
function response_index(request, response) {
  //POSTアクセス時の処理
  if(request.method == 'POST'){
    var body = '';

    //データ受信のイベント処理
    request.on('data', (data)=>{
      body += data;
    });

    //データ受信終了のイベント処理
    request.on('end', ()=>{
      data = qs.parse(body);
      write_index(request, response);
    });
  }else{
    write_index(request, response);
  }
}
```

```
//★indexの表示の作成
function write_index(request, response){
  var msg = "※伝言を表示します。";
  var content = ejs.render(index_page, {
    title: "Index",
    content: msg,
    data: data,
  });
  response.writeHead(200, {'Content-Type': 'text/html'});
  response.write(content);
  response.end();
}
```


response_indexを修正する

- リスト3-17は、POSTデータがあるときはその値をグローバル変数data.msgに格納する。
- なければグローバル変数data.msgを表示する

実行してみる#4

- 実際にapp.jsを実行してみる。

図5,6のようになりましたか？

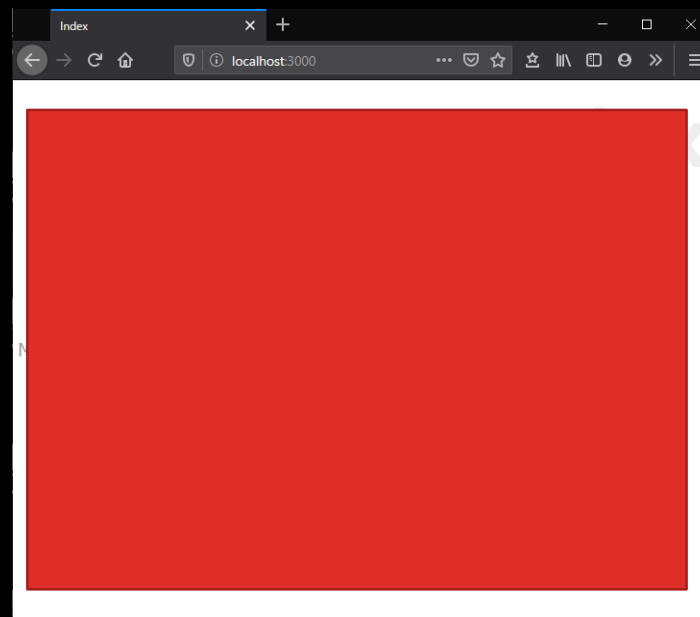


図5 送信前

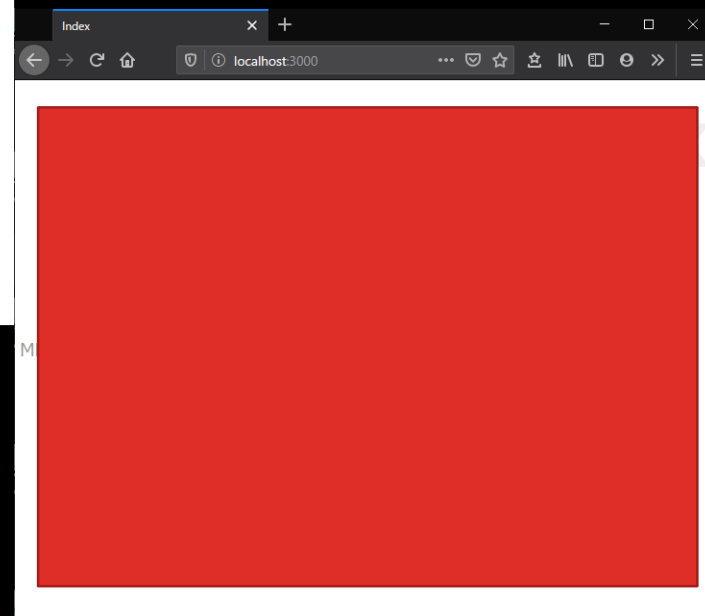


図6 送信後

クッキーの利用

- 固有データの代表的なものとしてクッキーが挙げられる。
- クッキーとはWEBブラウザに用意されてサーバから送られた値を保持する仕組み
- クッキーはヘッダー情報としてサーバとやり取りされる。
- サーバに送られたクッキーは処理され、またブラウザにヘッダー情報として送り返される

クッキーの難点

- クッキーには日本語などを**直接保管**できない。
- そのために**特殊な形式に変換して保管**し、使用するときにはもとに戻す必要がある
- クッキーに日本語を保管するためにはURLエンコード等を行う。
関数`encodeURIComponent(エンコードする値)`もしくは関数`encodeURIComponent(値)`でエンコード
関数`decodeURIComponent(エンコードする値)`もしくは関数`decodeURIComponent(値)`でデコード
- 教科書では`escape`関数でエスケープ`unescape`関数でアンエスケープしている。
- 例: ようこそ → `%E3%82%88%E3%81%86%E3%81%93%E3%81%9D`

テンプレートを修正する

- index.ejsの<body>部分をリスト3-15のように変更してください

リスト3-15の変更部

```
<p>your last message: <%= cookie_data %></p>
<form method="post" action="/">
<p>MESSAGE <input type="text" name="msg">
<input type="submit" value="送信"></p>
</form>
```

クッキー利用の処理を作成

- app.jsをリスト3-16のように変更してください

リスト3-16

```
// indexのアクセス処理
function response_index(request, response) {
    //POSTアクセス時の処理
    if(request.method == 'POST'){
        var body = '';

        //データ受信のイベント処理
        request.on('data', (data)=>{
            body += data;
        });

        //データ受信終了のイベント処理
        request.on('end', ()=>{
            data = qs.parse(body);
            setCookie('msg', data.msg, response);
            write_index(request, response);
        });
    }else{
        write_index(request, response);
    }
}
```

```
//クッキーの値を設定
function setCookie(key, value, response){
    var cookie = escape(value);
    response.setHeader('Set-Cookie', [key + '=' + cookie]);
}

//クッキーの値を取得
function getCookie(key, request){
    var cookie_data = request.headers.cookie != undefined ? request.headers.cookie : '';
    var data = cookie_data.split(';');
    for(var i in data){
        if(data[i].trim().startsWith(key + '=')){
            var result = data[i].trim().substring(key.length + 1);
            return unescape(result);
        }
    }
    return '';
}
```

実行してみる#5

- 実際にapp.jsを実行してみる。

図7, 8のようになりましたか？

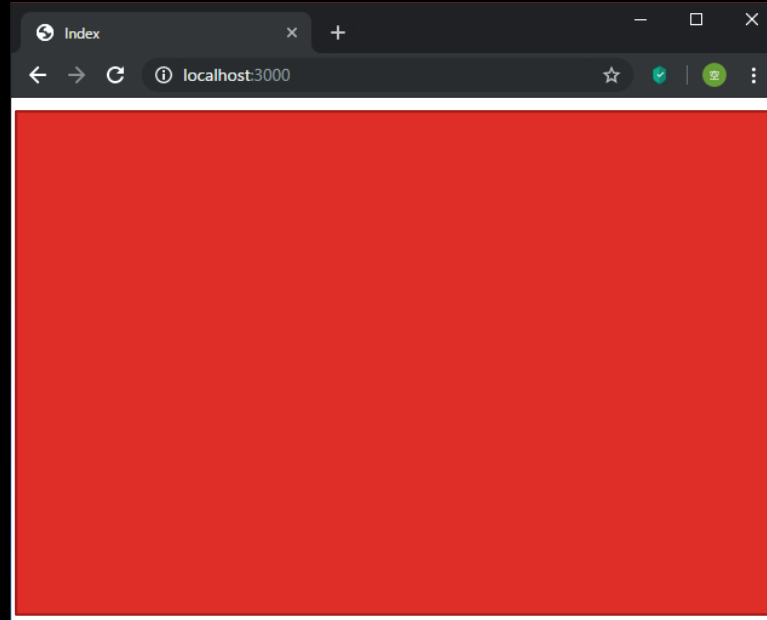


図7 送信前

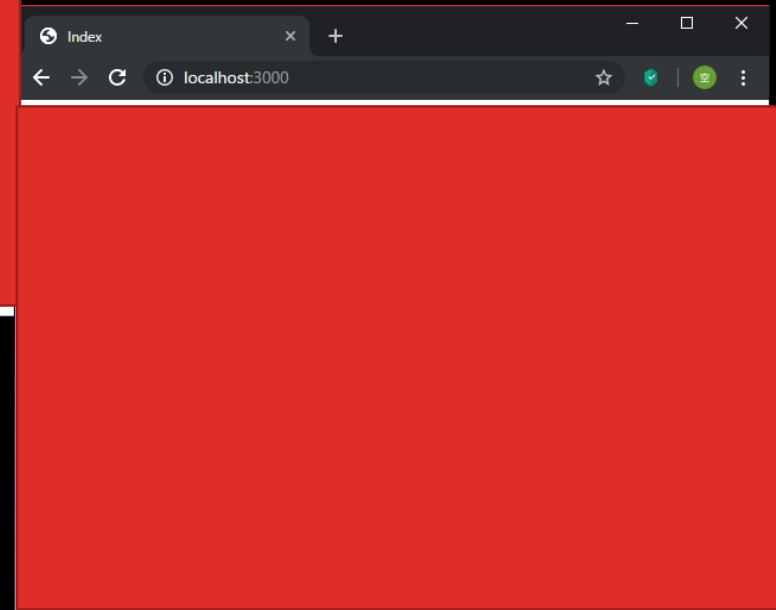


図8 送信後(リロード必須)

クッキーへの値の保存

- 先程のコードはクッキーの保存と読み込みを関数として用意してある。
- 関数の呼び出し時にresponseを渡している。... クッキーはヘッダー情報として送信する為
- 変数cookieに値を渡すときにescape(val)としている ... valの値をエスケープしている
- Headerに「'キー値=値'」のような形式で'Set cookie'する。

クッキーの値を取り出す

- Set-Cookieの値は['キー値=値', 'キー値=値', ...]の形式になっている。
- 実際の値は'キー値=値; キー値=値; ...'のようにセミコロンでつながっている。
- 値を取り出すときはそれぞれを;で切り離す必要がある。

三項演算子でCookieを得る

- 三項演算子とは、変数へ値を代入するときに条件式を当てはめるもの

- 書式: 変数 = 条件式 ? trueのときの値 : falseのときの値;

例えば,

```
var a = request.headers.cookie != undefined ? request.headers.cookie : '';
```

の場合は、request.headers.cookie が存在していれば内容を変数aに代入、存在していなければ変数aに空文字を代入する。

クッキーを分解する

- クッキーはセミコロンで連なっている為、これをsplit関数で切り離す。
- `var data = cookie_data.split(';');`のように実行すると `cookie_data`を`;`で区切ったものが配列で渡される。
- `a = 'Yamada;Kato;Tanaka;Hirashima';`を `rslt = a.split(';');`することによってrsltの値は、`['Yamada', 'Kato', 'Tanaka', 'Hirashima']`のようになる。

クッキーを分解する

- 前項で得られた`data`という配列をfor文で1つずつ処理していく。
- まず`data[i]`という文字列を`trim`メソッドでトリムして前後の空白を取り除く
- この値が`key=`という文字列で始まっているかを`startsWith`メソッドで判定する。
- 判定し、`true`だった場合、`data[i].trim()`したものを更に`substring(key.length + 1)`をすることで=より後を切り出す。

クッキーを分解する

- `key = 'abc'; data = 'abc=def';`だった場合に、
- `rslt = data.trim().substring(key.length + 1);`をした場合、
`key.length = 3`になるため、+1した4がインデックスの文字以降を**変数rslt**に格納する。

[0]	[1]	[2]	[3]	[4]	[5]	[6]
a	b	c	=	d	e	f

- よって、'def'の部分が取り出されて、**変数rslt**に格納される。

これ以上はセッション

- かんたんな値程度ならクッキーでいい。
- 複雑で大きなデータを保存するときにはセッションを用いる。
- しかし、Node.jsに標準ではその機能はない。



超簡単掲示板を作ろう

掲示板に必要なものは？

- 投稿データをファイルに
変数に保存すると、サーバを閉じたときに値が保持されない。どこかにファイルとして残す必要がある。
- 自分のIDをローカルストレージに
それぞれのクライアントごとにデータを保存する方法として、クッキーの他にローカルストレージという物がある。(クライアント側でしか動かない = Node.jsには非対応)

必要なファイルを整理する

app.js	メインプログラム
index.ejs	表示ページのテンプレート
login.ejs	ログインページのテンプレート
style.css	スタイルシート
data_item.ejs	表示用のパーシャル
mydata.txt	データを保管するテキストファイル

フォルダを用意する

- 掲示板として使用するフォルダを用意します。

面倒くさいのでワークスペースを分けるのではなく現在のワークスペース内にフォルダを作ることをおすすめします。

Index.ejsテンプレートを作成する

- リスト3-17をindex.ejsとして保存してください。
- サンプルデータの「mini_board」フォルダ内に完成品があります。
- `<body>`内の`<head>~</head>`を`<header>~</header>`に書き換えてください。

index.ejs

- `<head>`内の`<script>`タグは`javascript`を記述する際に使用するタグです。
- タグ内にある`function init()`は`<body onload="init();">`のように記述されている。

これは、ページが読み込み完了したときに`init()`を呼び出しています。

ローカルストレージの値の取得

- Node.jsはローカルストレージの値を取得できない。よって前項のinit()内に書かれたjavascriptで値を取得してサーバ側が読み取れるようにする。
- ここではローカルストレージを扱うためのlocalStrageというオブジェクトを使用する。
- var id = localStrage.getItem('id');で、idというキーのデータを取り出す。
- 書式: 変数 = localStrage.getItem(キー);

IDがなければログインページに移動

- 前項で取得したデータ'id'が存在しなかった場合、ログインページに移動する。

```
if(id == null){  
    location.href = './login';  
}
```

- idが存在しない場合、値はnullとなるため、location.hrefでログインページのパスを指定して、移動させている。

IDを非表示フィールドに設定する

- スライド44で取得したidをテンプレート内の2箇所に設定する。
- `document.querySelector()`というメソッドでindex.ejs内にあるエレメントを取得する。
- `document.querySelector('#id')`は'id'というidが指定されたエレメントを取得する。
- (``の部分)
- `document.querySelector('#id_input')`はid_inputというidが指定されたエレメント
- (`<input type="hidden" id="id_input" name="id" value="">`)

IDを非表示フィールドに設定する

- `document.querySelector('#id').textContent = 'ID:' + id;`

これは、``に、`ID: id`の内容を設定する。また、表示される。

- `document.querySelector('#id_input').value = id;`

これは表示されない`<input>`の`value`に値を入れている。これはformと一緒に送信される。

テーブルのパーシャル・テンプレート

- リスト3-18をdata_item.ejsとして保存してください。これもサンプルにあります。

リスト3-18

```
<% if (val != ''){ %>
  <% var obj = JSON.parse(val); %>
  <tr>
    <th style="width:100px;"><%= obj.id %></th><td><%= obj.msg %></td>
  </tr>
<% } %>
```

パーシャル側の処理

- `<% if(val != '') { %>`で念の為、`val`という値が存在するかを確認し、存在すればパーシャル内を処理する。
- `val`は **JSON**データとして与えられる。
- **JSON**はオブジェクトをテキストの形式で記述したもの。よってそのままでは使用できない。
- そのため、`var obj = JSON.parse(val)`のようにパースしてオブジェクトに戻す必要がある。
- あとはテーブルの中身として表示するだけ

Login.ejsを作る

- リスト3-21をlogin.ejsとして保存してください。サンプルにあります。

```
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta http-equiv="content-type"
    content="text/html; charset=UTF-8">
  <title>LOGIN</title>
  <link type="text/css" href="./style.css" rel="stylesheet">
  <script>
    function setId(){
      var id = document.querySelector('#id_input').value;
      localStorage.setItem('id', id);
      location.href = '/';
    }
  </script>
</head>
<body>
  <div>
    <h1>LOGIN</h1>
    <p>あなたのログインネームを入力下さい。</p>
    <p><input type="text" id="id_input">
      <button onclick="setId();">送信</button>
    </p>
  </div>
</body>
</html>
```

ローカルストレージに値を保存する

- フォームを用意して、そこにIDを入力できるようにする。
- `<input type="text" id="id_input">`で入力欄を表示し、
- `<button onclick="setId();">送信</button>`でクリック時に関数`setId`を呼び出す。
- 関数`setId`は
- 入力欄を`querySelector`でエレメントとして取得してその値を`id`という変数に保持
- `localStorage.setItem('id': id);`で`'id'`というキーとしてローカルストレージに格納する。
- 書式: `localStorage.setItem(キー: データ);`

データファイル「mydata.txt」とスタイルシート

- 掲示板のデータを保持するためのデータファイルmydata.txtを作成する。
- また、スタイルシートstyle.cssを作成し、記述をする。
サンプルにあります。

メインプログラムapp.js

- メインプログラムを作成する。
- リスト3-20をapp.jsとして保存してください。
サンプルにあります。

必要な人は(蛇足)

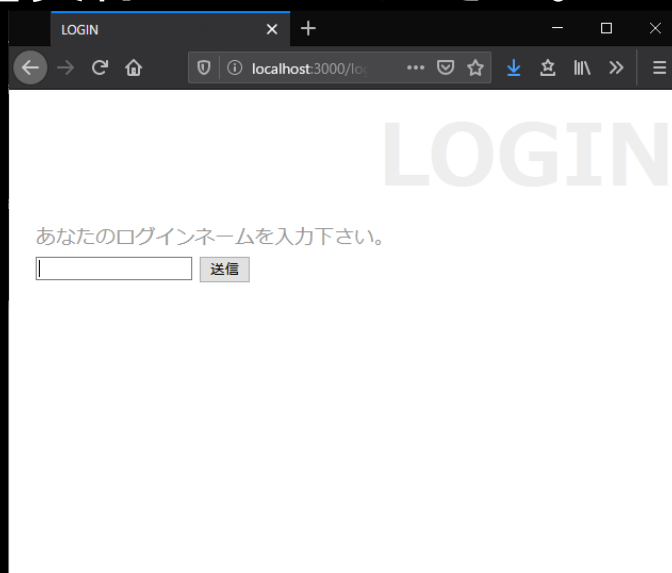
- ejsをインストールしてない人は
- `npm install -g ejs`(パスの設定ができてる人)
- `npm instal ejs`(パスを設定してない人)

のコマンドをコマンドプロンプトで実行してください。(後者はcd ディレクトリをしたあと)

- パスの設定は環境変数のNODE_PATH
に `%AppData%\npm\node_modules`に指定してください。

実行してみる#6

- app.jsを実行してみてください。



データファイルの処理について

- ファイルの最大数は
- `const max_num = 10;` の部分で変更できます。
- ファイルのロード
- 関数`readFromFile(fname)`内で、`fs`モジュールを使用しファイルを開いている。
- 開かれたファイルから`split('¥n')`で改行コードで1行ごとに取り出されている。

データの更新

- index.ejsでは、requestのendイベントでデータをすべて受け取ったらaddToData(id, msg, fname, request)関数を呼び出している。
- 関数内部では var obj = {'id': id, 'msg': msg};のように、一旦オブジェクトにまとめる。
- これをvar obj_str = JSON.stringify(obj);でJSONデータにする
- オブジェクトを渡すときはJSONにすると受け渡しが便利！

データの更新

- 前項で作成したJSONデータを
- `message_data.unshift(obj_str);`で`message_data`の一番上に挿入する。
- 以下の文で`message_data`の数が`max_num`より多かった場合、一番古いデータを削除する。
- `if (message_data.length > max_num){`
- `message_data.pop();`
- `}`

配列を保存する

- `var data_str = message_data.join('¥n');`
- 上の文で`message_data`という配列を1つのテキストデータとしてまとめる。
`.join('¥n')`はそれぞれの区切りを改行コードにするという関数
- ここでできたファイルをfsモジュールのfsモジュールの`writeFile`メソッドでファイルに書き込む。
- `fs.writeFile(fname, data_str, (err)=>{...});`
- **ファイル名**を指定してそこに`data_str`を書き込んでいる。

まとめ(1/2)

- フォームの送信とイベント処理

フォームの送信はこの章で最も重要。

イベントの処理はonを使用した。よく使うため覚えておく。

- ローカルストレージとクライアント機能の利用

ローカルストレージはクライアント専用の機能。かんたんに使える。

Node.jsはサーバサイドであるため使用するには工夫が必要。

サーバとクライアントの連携が重要！

(JSはクライアントサイド)

まとめ(2/2)

- ファイルの書き込みはデータ保存の基本
サンプルではJSのオブジェクトをJSONテキストにして保管した。
また、それを利用してオブジェクトを生成した。この交互変換はできるようになっておく。
テキストファイルの書き込みもできるようになる
- 素のNode.jsは難しい！
次回のExpressができるようになるともっと便利に同様の機能が作れる！