

|  |
| --- |
| jQueryとAjax |

目次

[1. jQueryの準備 1](#_Toc476154133)

[1 jQueryを使う準備 1](#_Toc476154134)

[2 jQueryの基本 3](#_Toc476154135)

[2. jQueryによるDOM操作 7](#_Toc476154136)

[1 要素の操作 7](#_Toc476154137)

[2 メソッドチェーン 9](#_Toc476154138)

[3. jQueryのメソッド 10](#_Toc476154139)

[1 イベント 10](#_Toc476154140)

[2 フォーム機能 14](#_Toc476154141)

[4. Ajaxの概要 22](#_Toc476154142)

[1 Ajaxとは 22](#_Toc476154143)

[2 AjaxとJavaScriptライブラリ 22](#_Toc476154144)

[3 HTTP 24](#_Toc476154145)

[4 jQueryによるAjax 25](#_Toc476154146)

[5. Ajaxの実装 28](#_Toc476154147)

[1 Ajaxとデータベース 28](#_Toc476154148)

[2 EclipseとjQuery 36](#_Toc476154149)

# jQueryの準備

jQuery は最も広く利用されている JavaScript ライブラリの1つです。jQueryの最大の特徴は、JavaScriptとは異なる手軽な書き方でJavaScriptの命令を記述できることにあります。JavaScriptの記述には多くの手続きを必要としますが、jQueryを利用すると非常に短いコードで済みます。また、CSSに近い記述方法を採用しているため、CSSが分かるWebデザイナーやエンジニアが手軽に習得できるよう設計されています。

jQueryはMITライセンスで公開されているため、ライブラリ中の著作権表示を消さなければ商用、非商用を問わず誰でも自由に利用することができます。また、jQueryは非常に軽量なライブラリとなっており、ファイルサイズは約100KB程しかありません。そのため、Webページの読み込み速度にほとんど影響を与えることなく利用できるのも大きな特徴と言えるでしょう。

jQueryは次のURLから入手することができます。

|  |
| --- |
| http://jquery.com/ |

## jQueryを使う準備

　jQueryのWebサイトのトップページで「Download jQuery」ボタンをクリックすると、ダウンロードページが表示されます。ダウンロードページでは2種類のファイルをダウンロードできます。

jQueryの配布物には以下の2種類のファイルが含まれています。

|  |  |  |
| --- | --- | --- |
| **ファイル** | **ファイル名** | **説明** |
| compressed | jquery-(バージョン番号).min.js | ファイルサイズを圧縮、最小化したもので、サイト運用時など転送量を減らす場合に使用する |
| uncompressed | jquery-(バージョン番号).js | ファイルサイズは大きいが、開発時にjQueryのソースコードを参照、デバッグなどを行う場合に使用する |

* 「Download the compressed, production jQuery 1.10.2」

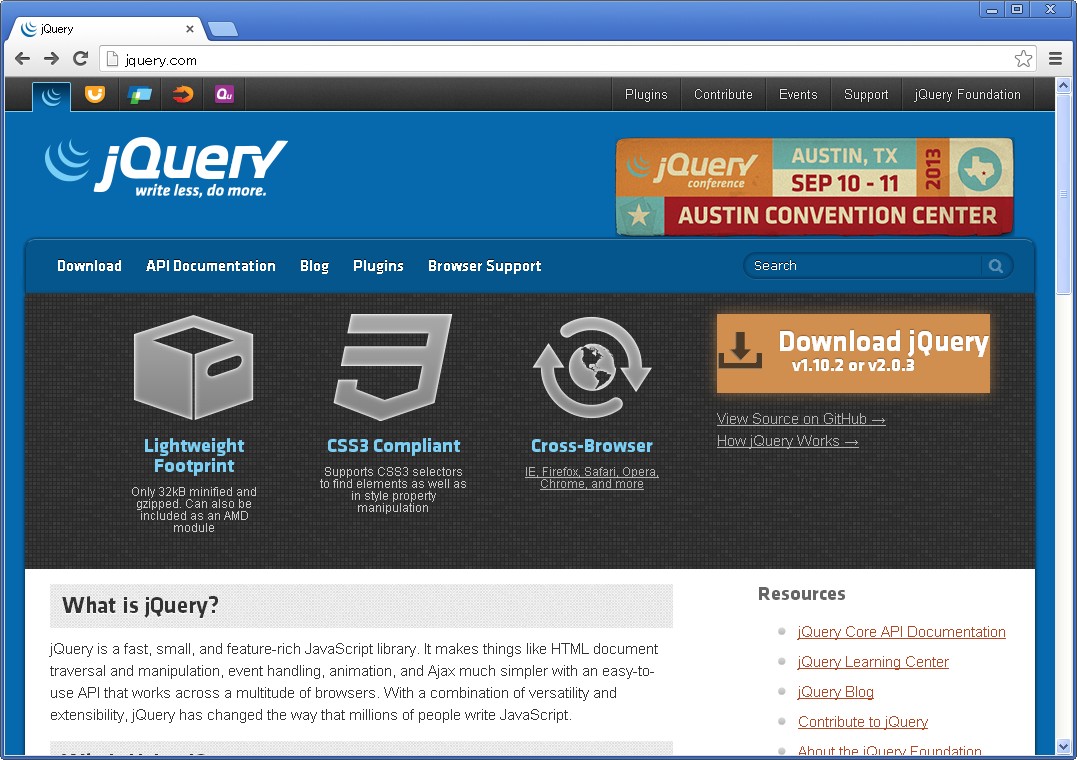
ファイルサイズを圧縮するために最小化されたもの。

サイトの運用時に転送量を減らしたい場合に使用する。

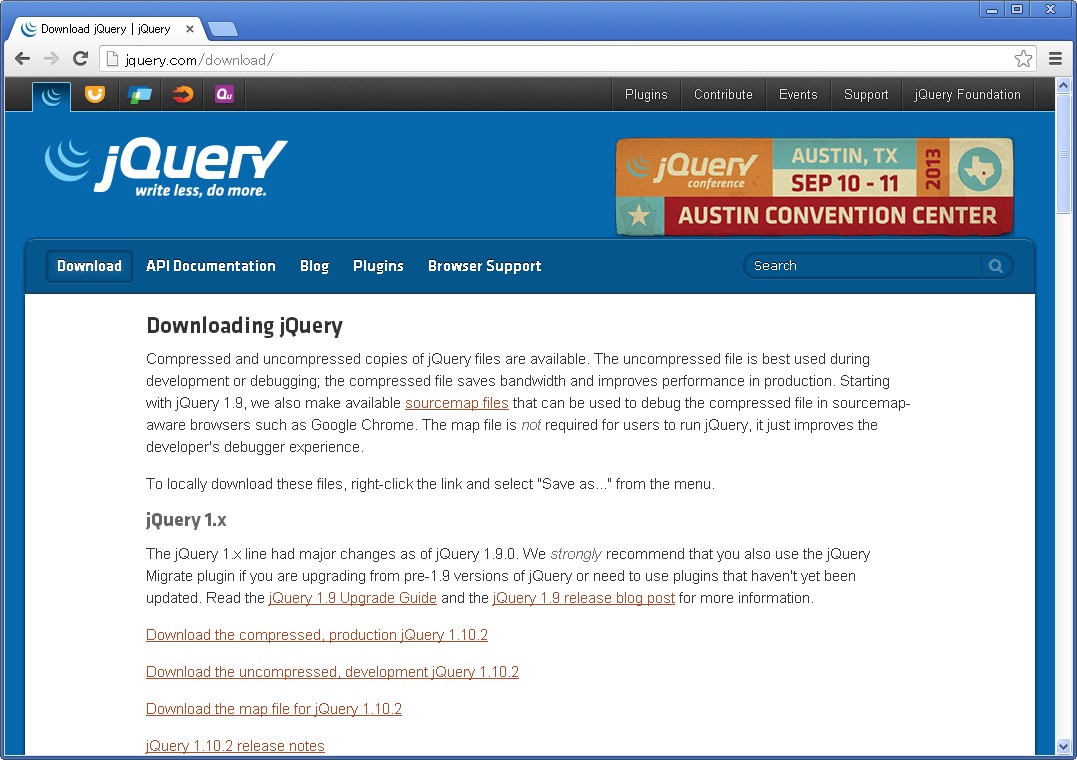
* 「Download the uncompressed, development jQuery 1.10.2」

最小化されていないためファイルサイズは大きいが、ソースは読みやすい。

開発時にjQueryのソースコードを参照し、デバッグを行う場合に使用する。



jQueryのWebサイトのトップページ



jQueryのダウンロードページ

ここではuncompressedバージョンを使用します。「Download the uncompressed, development jQuery 1.10.2」のリンクを右クリックし、コンテキストメニューから「対象をファイルに保存」を選択して適当なフォルダに保存してください。

HTMLではダウンロードしたJavaScriptファイルを次のように参照することでjQueryを使用することができます。

|  |
| --- |
| <script src="jquery-1.10.2.js"></script> |

script要素はページ内のどこに記述しても構いませんが、一般的にはhead要素の内部に記述します。

|  |
| --- |
| <head>  <meta charset=”utf-8”>  <title>jQueryを読み込む</title>  <script src=”jquery-1.10.2.js”></script>  </head> |

ダウンロードしたファイルを参照する代わりに、CDN(コンテンツデリバリーネットワーク)を使用することも可能です。

|  |
| --- |
| <script src="http://code.jquery.com/jquery-1.10.2.min.js"></script> |

## jQueryの基本

jQueryを含むJavaScriptのコードはscript要素の中に記述する必要がありますが、

WebブラウザはHTMLをファイルの先頭から1行ずつ読み込んでおり、script要素が読み込まれた段階で、script要素内記述された命令を実行します。以下jQueryを使用していない簡単なサンプルです。

|  |
| --- |
| <!DOCTYPE html>  <html lang=”ja”>  <head>  <meta charset=”UTF-8”>  <title>sample</title>  <script>  alert(“Hello world”)  </script>  </head>  <body>  <h1>サンプルテキスト</h1>  </body>  </html> |

サンプルを実行すると「Hello world」というダイアログがブラウザ上に表示されます。「OK」ボタンを押すとbody要素に記述されているテキストが表示されますが、ボタンを押すまでは何も表示されません。「alert」という命令が実行された時点では、script要素以降のHTMLをブラウザは認識していないため、alertの命令を終了させた後にテキストが表示されるという仕組みになっています。

head要素内のscript要素にbody要素内のHTMLを操作するための命令を書いても、ブラウザがまだscript要素までしか読み込んでいないので、本来意図した通りに動きません。

そこで登場するのが、スクリプトを実行するタイミングを制御する「ready」というjQueryのメソッドです。readyメソッドは以下のような形式で使用します。

|  |
| --- |
| $(document).ready(function(){  //jQueryの処理を記述  }); |

readyメソッドの内側(function(){…}内)に記述されている命令は、HTMLの読み込みが終わった後に実行されます。readyという名前のとおり、「準備ができた段階でスクリプトを実行」という仕組みになっています。

また、readyメソッドは以下のような省略形を使用することができます。

|  |
| --- |
| $(function(){  //jQueryの処理を記述  }); |

jQueryでスクリプトを記述するほとんどの場合、まずreadyメソッドを記述し、その内側に実際の命令を記述していきます。jQueryの基本的な利用方法を学ぶために、jQueryを使用して簡単なDOM操作を行うサンプルを見てみましょう。

jQueryサンプル.html

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <meta charset="UTF-8">  <title>sample</title>  <script src="jquery-1.10.2.min.js"></script>　❶  <script>  $(function(){　❷  $('#button').click(function(){　❸  var name = $('#name').val();　❹  alert('こんにちは、'+ name + 'さん！');  });  });  </script>  </head>  <body>  <h1>jQueryの基本</h1>  お名前をどうぞ：  <input type="text" id="name" />  <input type="button" id="button" value="クリック" />  </body>  </html> |

❶ではjQueryのJavaScriptファイルをインポートしています。

jQueryではイベントハンドラの登録などHTMLのロード後に実行される処理を、❷のように$(function(){…});の中に記述します。

❸のように$()関数にCSSセレクタを指定することで要素を検索できます。また、click()などイベントに対応したメソッドでイベントハンドラを登録できます。ここではid属性がbuttonの要素のclickイベントにイベントハンドラを登録しています。

$()関数で取得したjQueryオブジェクトには要素やDOMツリーの操作を行なう際に便利な各種のメソッドが用意されています。ここではval()メソッドでテキストフィールドの値を取得しています(❹)。

jQueryが提供する主な機能について以下にまとめます。

●CSSセレクタ

　通常のJavaScriptで要素を取得するには、DOMツリーを1つずつたどるか、id属性の値を指定してdocument.getElementById()メソッドを使用などの方法しかありませんが、jQueryでは$()関数の引数にCSSセレクタを指定することで非常に柔軟な検索を行なうことができます。

|  |
| --- |
| // id属性がmylement配下のdiv要素を取得  var element=$('#myelement div'); |

●DOM操作

　jQueryオブジェクトのメソッドを使用することで、煩雑なDOM操作を手軽に行なうことができます。たとえば次のコードはclass属性がimageのli要素の子要素の先頭にimg要素を挿入しています。このようにjQueryを使用することで、CSSセレクタに一致した複数の要素に対する操作を一度に行なうことができます。

以下にjQueryを使用したDOM操作の例を示します。

|  |
| --- |
| // <li class="icon">...<li>の先頭に<img src="icon.png"/>を挿入  $('li.icon').prepend($('<img>').attr('src', 'icon.png')); |

|  |
| --- |
| <ul>  <li class="icon">HTML</li>  <li class="icon">CSS</li>  <li class="icon">JavaScript</li>  </ul> |

|  |
| --- |
| <ul>  <li class="icon"><img src="icon.png"/>HTML</li>  <li class="icon"><img src="icon.png"/>CSS</li>  <li class="icon"><img src="icon.png"/>JavaScript</li>  </ul> |

先頭にimg要素を挿入

●イベントハンドラの設定

　JavaScriptでイベント処理を行なう場合、ブラウザごとのイベントモデルの違いやイベントハンドラに渡されるイベントオブジェクトの差異に留意する必要があります。jQueryを使用することで、ブラウザ間のイベントモデルの差異を吸収し、ブラウザの違いを意識することなくイベントを処理することができます。

●その他の機能

　jQueryにはその他にも、Ajaxによる通信を簡単に行なうための機能や、配列の操作などを手軽に行なうための便利なユーティリティが含まれています。

# jQueryによるDOM操作

## 要素の操作

　DOM要素の操作を行うためのメソッドには次のようなものがあります

|  |  |
| --- | --- |
| **メソッド名/構文** | **説明** |
| after(content) | 引数で指定したコンテンツ(content)を要素の直後に追加する |
| defore(content) | 引数で指定したコンテンツ(content)を要素の直前に追加する |
| insertAfter(target) | 引数に指定した要素(target)の直後にこのコンテンツを追加する |
| insertBefore(target) | 引数に指定した要素(target)の直前にこのコンテンツを追加する |
| append(content) | 引数に指定したコンテンツ(content)を要素の末尾に追加する |
| appendTo(target) | 引数に指定した要素(target)の末尾にこのコンテンツを追加する |
| prepend(content) | 引数で指定したコンテンツ(content)を要素の先頭に追加する |
| prependTo(target) | 引数に指定した要素(target)の先頭にこのコンテンツを追加する |
| remove([selector]) | 要素を削除する引数にセレクタ(selector)を指定した場合はこのjQueryオブジェクトに含まれている要素の中からセレクタに一致した要素が削除される |
| detach([selector]) | remove()メソッドと同じだが、要素に設定されているイベントハンドラやデータは削除しない。いったん削除した要素を再度DOMに追加するときに便利 |
| empty() | 要素内のすべての子ノードを削除する |
| attr(name[.value]) | 属性(name)を取得または設定する |
| attr(hash) | 複数の属性(hash)をまとめて設定する |
| removeAttr(name) | 属性(name)を削除する |
| replaceWith(content) | 引数にしていたコンテンツでこの要素を置換する |
| replaceAll(target) | 引数で指定した要素(target)をこのコンテンツで置換する |
| wrap(parent) | 引数で指定した要素(parent)でラップする |
| unwrap() | 新要素を削除する |
| wrapAll(parent) | 選択している要素(parent)をまとめて引数で指定した要素をラップする |
| wraplnner(parent) | 子要素を引数で指定した要素(parent)でラップする |
| html([content]) | 要素内のHTMLコンテンツ(content)も取得または設定する |
| text([content]) | 要素内のテキストコンテンツ(content)も取得または設定する |
| clone() | 要素をコピーして新しいオブジェクトを作成する |
| css(name[.value]) | 指定したCSSプロパティの値を設定・取得する |

●属性の追加/削除

　attr()メソッドはJavaScriptでいうところのsetAttribute()メソッドとgetAttribute()メソッドの複合形であり、要素の属性値の取得および設定、removeAttr()メソッドで属性の削除を行うことができます。attr()メソッドの引数にオブジェクトを指定すると、複数の属性をまとめて設定することができます。

|  |
| --- |
| //a要素のhref属性を取得  var url = $(‘a’).attr(‘href’);  //a要素のhref属性を設定  $(‘a’).attr(‘herf’, ‘http://www.google.co.jp/’);  //複数の属性をまとめて設定  $(‘a’).attr({  title: ‘Google’,  href: ‘http://www.google.co.jp/’  }); |

●要素内のコンテンツの取得/設定

　text()メソッドは要素内のテキスト、html()メソッドは要素内のHTMLを取得および設定できます。引数なしで呼び出すとコンテンツを取得し、引数を指定して呼び出すと引数に指定したコンテンツを設定します。

要素内のコンテンツの取得と設定.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="ja">  <head>  <meta charset="utf-8">  <title>sample</title>  <script src="jquery-1.10.2.min.js"></script>  </head>  <body>  <h1>元々のテキスト</h1>  <script>  //h1要素内のテキストを設定  $('h1').text('Hello, jQuery!');  //h1要素内のテキストを取得  var text = $('h1').text(); //→'Hello, jQuery!'  alert(text);  //h1要素内のHTMLを設定  $('h1').html('<a href="http://jquery.com/">Hello, jQuery!</a>');  //h1要素内のHTMLを取得  var html = $('h1').html(); //→ '<a href="http://jquery.com/"> Hello, jQuery!</a>  alert(html);  </script>  </body>  </html> |

## メソッドチェーン

　ここまでHTMLとCSSを操作する命令を一気に紹介してきました。jQueryではDOM操作するときに便利な「メソッドチェーン」という記述を利用することができます。

以下の例はprepend()メソッドとappdend()メソッドを利用し、p要素の先頭と最後にsample要素を挿入するサンプルです。

|  |
| --- |
| $(“p”).prepend(“<sample>先頭に挿入</sample>”);  $(“p”).append(“<sample>後方に挿入</sample>”); |

$(“p”)という同じセレクタに対して、prepend()とappend()の2つの命令が設定されています。このスクリプトをブラウザで実行すると、jQueryはp要素を探し、prepend()を実行し、次にまたp要素を探し、append()を実行します。p要素を2回探しにいくことになるので非効率的です。

上記の内容を、メソッドチェーンを利用して書き直すと以下のようになります。

|  |
| --- |
| $(“p”).prepend(“<sample>先頭に挿入</sample>”)  .append(“<sample>後方に挿入</sample>”); |

コードは少し短くなりましたが、実行してみると結果は前と同じです。このように、jQueryでは命令と命令を.(ドット)でつなげて記述することで、1つのセレクタに対して2つ以上の命令を連続して実行できます。この記述方法をメソッドチェーンと呼びます。

単に記述が短くなるだけではなく、今回の場合はp要素を一度検索した後に、各メソッドが実行されるので内部的な処理も効率が良くなります。同一のセレクタに対して複数の命令を指定する場合は、メソッドチェーンで記述するようにしてください。

# jQueryのメソッド

## イベント

　jQueryではプログラムが実行されるきっかけを「イベント」と言います。イベントの発生を感知したタイミングで処理が実行される仕組みです。イベントを感知して処理を実行するにはいくつかの命令を使用します。主に次のようなものがあります。

|  |  |
| --- | --- |
| **メソッド名/構文** | **説明** |
| click() | クリック時に処理を実行する |
| dblclick() | ダブルクリック時に処理を実行する |
| mousedown() | マウスのボタンが押された時に処理を実行する |
| mouseup() | マウスのボタンが離された時に処理を実行する |
| mouseover() | マウスオーバー時に処理を実行する |
| mouseout() | マウスアウト時に処理を実行する |
| mousemove() | マウスが移動したときに処理を実行する |
| one() | イベント発生時に一度だけ処理を実行する |
| on() | 対象要素を絞ってイベントを登録する |
| off() | 設定されているイベント処理を取り消す |
| hover() | マウスオーバー/マウスアウト時に処理を実行する |

●特定の要素がクリックされたときに処理を実行

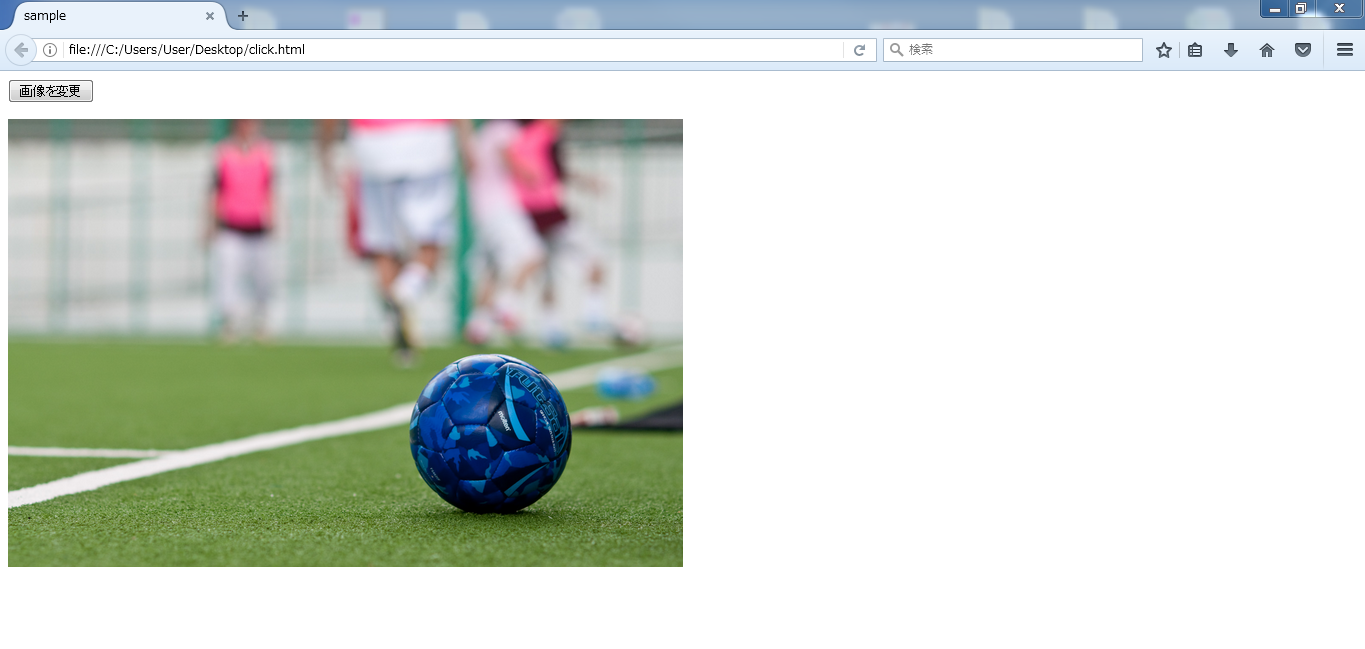
　click()メソッドを使用することで特定の要素がクリックされた時に処理を実行することができます。dblclickメソッドを使用した場合は特定の要素が「ダブルクリック」された時に処理が実行されます。以下サンプルになります。

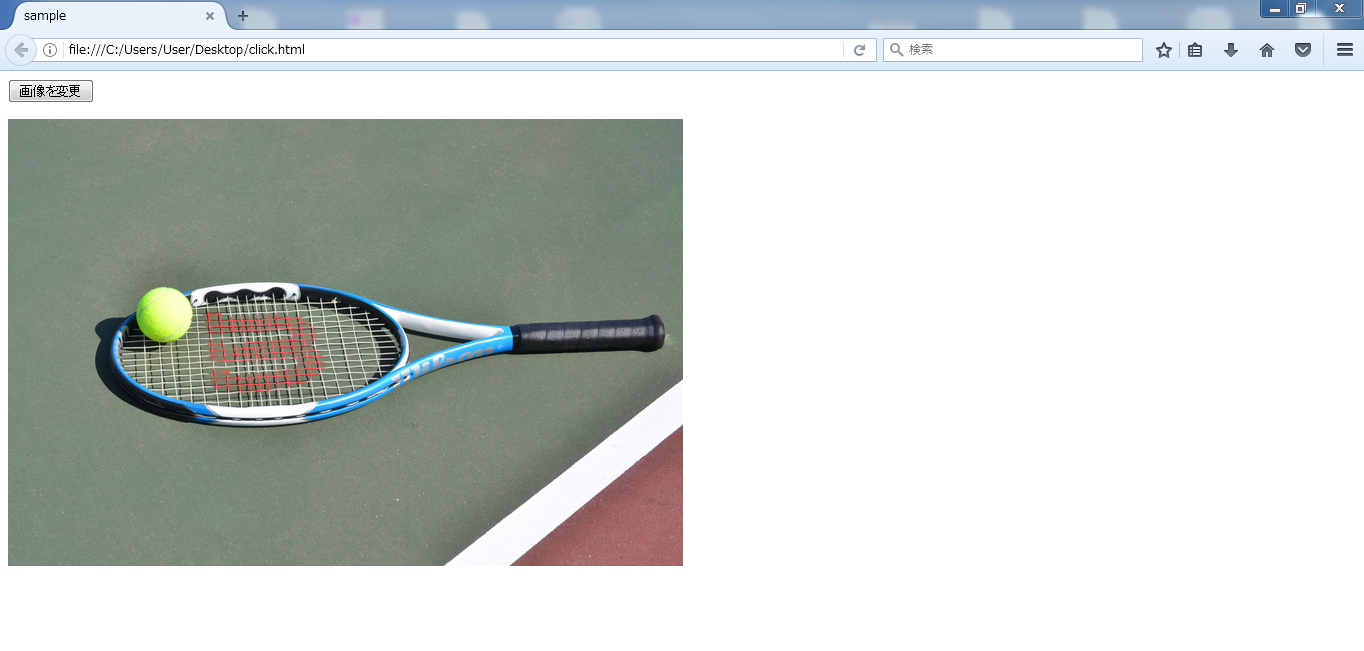
特定の要素がクリックされたときに処理を実行.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="ja">  <head>  <meta charset="UTF-8">  <title>sample</title>  <script src="jquery-1.10.2.min.js"></script>  <script>  $(function(){  $("button").click(function(){  $("img").attr("src","tennis.jpg").attr("alt","テニス");  });  });  </script>  </head>  <body>  <button>画像を変更</button>  <p><img src="soccer.jpg" alt="サッカー" style="width:50%; height:auto;"></p>  </body>  </html> |

上記サンプルはbutton要素がクリックされたらimg要素のsrc属性の値を「tennis.jpg」に、Alt属性の値を「テニス」に書き換えるスクリプトです。attr()メソッドはカンマ区切りで指定した属性値を変更する命令です。メソッドチェーンを利用し、src属性とalt属性を一度に書き換えています。また、あくまでsrc属性とalt属性を書き換えただけなので、style属性は書き変わっていません。画像サイズが変わらないことにも注目してみてください。

実行結果





●複数の要素の取得

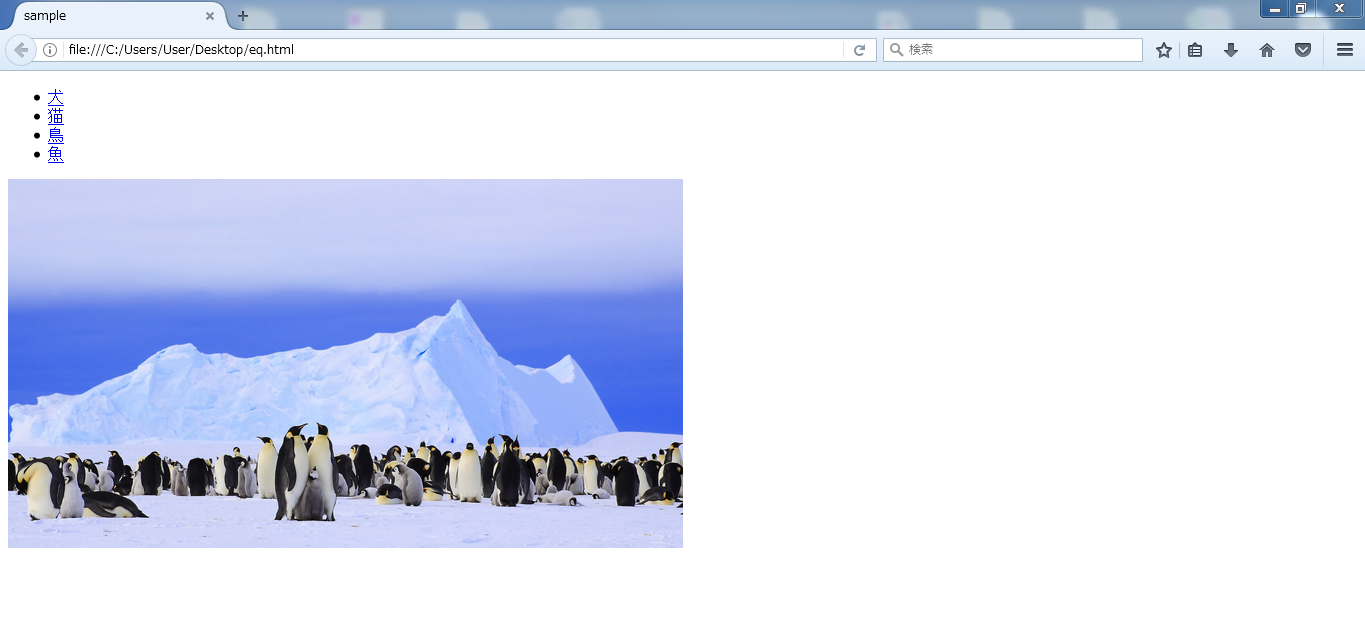
　複数の要素に対してイベント設定する場合もあるでしょう。下記のサンプルは複数のa要素に対してそれぞれ異なるクリックイベントを設定しています。複数の要素を指定した順番に取得するには「eq(equal)セレクタ」を使用します。JavaScriptは0から数値を数えるため、1番目の要素は「0」二番目の要素は「1」になります。

複数の要素の取得.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="ja">  <head>  <meta charset="UTF-8">  <title>sample</title>  <script src="jquery-1.10.2.min.js"></script>  <script>  $(function(){  $("a:eq(0)").click(function(){  $("img").attr("src","dog.jpg").attr("alt","犬");  return false;  });  $("a:eq(1)").click(function(){  $("img").attr("src","cat.jpg").attr("alt","猫");  return false;  });  $("a:eq(2)").click(function(){  $("img").attr("src","penguin.jpg").attr("alt","鳥");  return false;  });  $("a:eq(3)").click(function(){  $("img").attr("src","fish.jpg").attr("alt","魚");  return false;  });  });  </script>  </head>  <body>  <ul>  <li><a href="dog.jpg">犬</a></li>  <li><a href="cat.jpg">猫</a></li>  <li><a href="penguin.jpg">鳥</a></li>  <li><a href="fish.jpg">魚</a></li>  </ul>  <p><img src="dog.jpg" alt="犬" style="width:50%; height:auto;"></p>  </body>  </html> |

上記サンプルはclick()メソッドで対象としているセレクタがa要素ですが、もともとa要素は「クリックされるとhref属性に書かれたリンク先に移動する」働きがあります。そのままでは画面遷移をしてしまうので、clickイベントの最後に「return false;」を追加します。この場合a要素がクリックされた際にはjQueryの命令だけが実行され、href属性に設定されたリンク先には移動しません。

実行結果



●イベントが発生した要素を取得する

　画像を変更する機能は実装できましたが、このままだとコードがかなり長くなってしまい、画像が増えるたびにコードを追記しなくてはいけません。それを解消するために、イベントが発生した要素を取得するセレクタ「$(this)」が用意されています。イベントを設定しているclick(function(){・・・})内で$(this)と書くと、イベントが発生した要素を取得できます。

$(this)を使用すると、先程のサンプルのjQuery文を以下のように短く記述できます。

|  |
| --- |
| <script>  $(function(){  $("a").click(function(){  $("img").attr("src",$(this).attr("href")).attr("alt",$(this).text());  return false;  });  });  </script> |

クリックされた要素のhref属性を$(this).attr(“href”)で、クリックされた要素内のテキストを$(this).text()で取得し、img要素のsrc属性、alt属性にそれぞれ設定しています。仮に画像が10枚増えても、jQueryを書き換える必要がなくなり、HTMLの修正だけで済みます。

イベントが発生した要素を取得する$(this)はclickイベント以外でも使用する機会がとても多いので覚えておきましょう。

## フォーム機能

　jQueryのフォーム機能を使用すると、入力内容に応じてフォームのデザインを変更したり、エラーメッセージを表示したりでき、使い勝手のいいフォームを作成できます。

フォームの機能には主に次のようなものがあります。

|  |  |
| --- | --- |
| **メソッド名/構文** | **説明** |
| val() | フォームに入力または選択されている値を取得・変更する |
| focus() | マウスやタブキーによって選択された状態を感知する |
| blur() | 選択が外れたことを感知する |
| change() | フォームの内容変更を感知する |
| submit() | フォームの送信を感知する |
| checked() | チェックの入っているフォーム部品を選択する(チェックボックス) |
| selected() | 選択されている要素を選択する(セレクトボックス) |

●フォーム選択時に動的に操作を行う

　focus()メソッドを使用し、input要素などのフォーム部品がマウスやタブキーによって選択された状態(フォーカス状態)になったことを感知し、設定された処理を実行します。他のイベントと同じように、括弧内にfunction(){…}を記述し、その中に実行したい処理を記述します。また、フォーカスが外れたことを感知するblur()メソッド使用した例を紹介します。

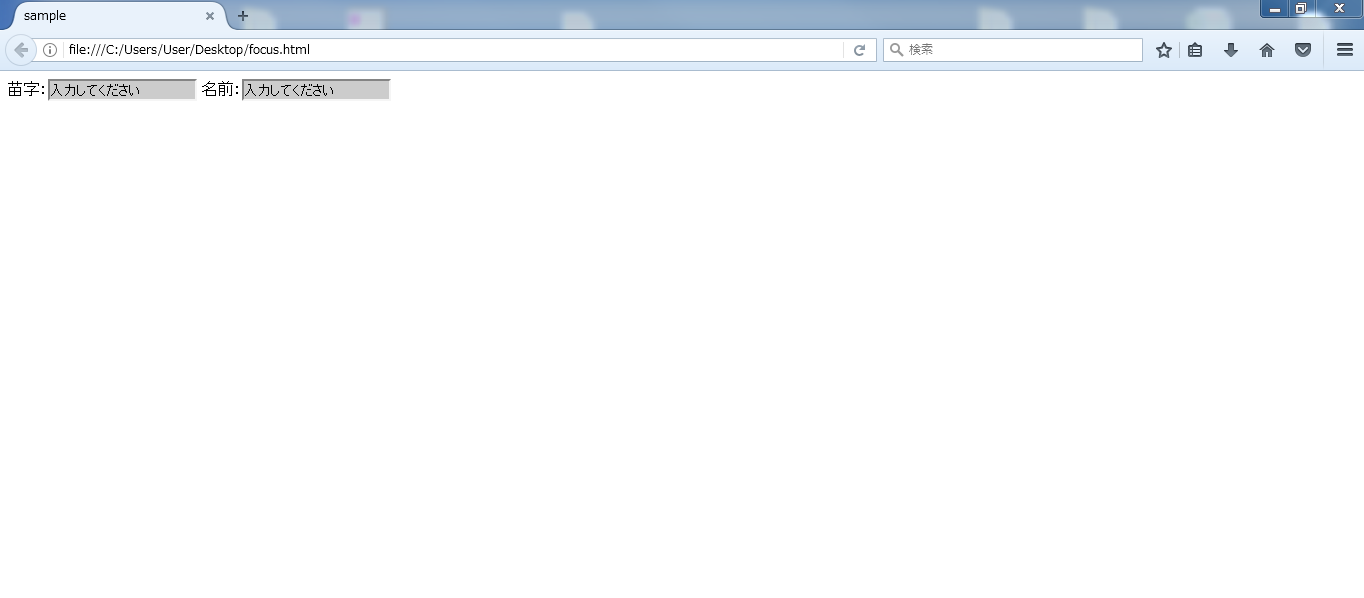
以下サンプルになります。

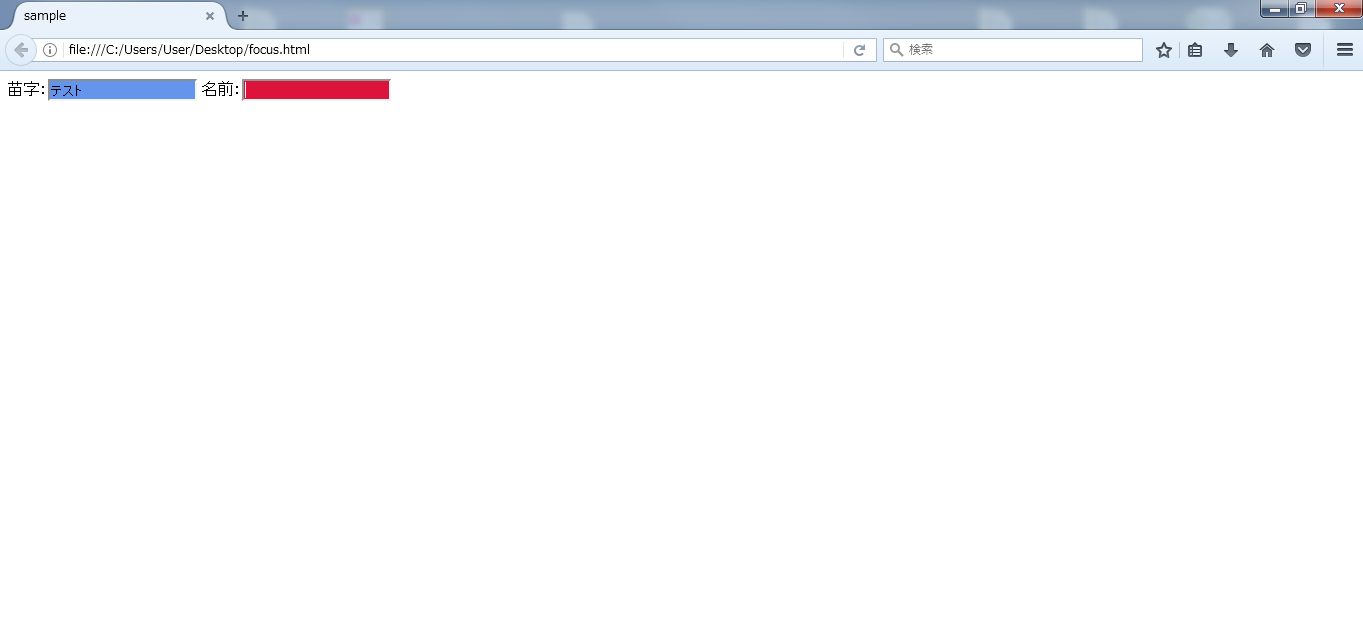
フォーム選択時に動的に操作を行う.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="ja">  <head>  <meta charset="utf-8">  <title>sample</title>  <script src="jquery-1.10.2.min.js"></script>  <script>  $(function(){  //初期設定 背景を灰色に、val()メソッドを使い初期値を設定、文字色を黒に  $('input[type="text"]').val("入力してください").css('color','#000000')  .css('background','#CCC').focus(function() {  //フォーカスされたらval()メソッドで初期化し、文字色を白に、背景色を赤にする  $(this).val("").css('color','#fffafa').css('background','#dc143c');  }).blur(function() {  //フォーカスが外れたら文字色を黒に背景色を青にする  $(this).css('color','#000000').css('background','#6495ed');  });  });  </script>  </head>  <body>  苗字：<input type="text">  名前：<input type="text">  </body>  </html> |

上記サンプルはメソッドチェーンを利用し、テキストボックスの入力欄に値と、文字色、背景色の三つを設定しています。テキストボックスを選択するとfocus()メソッドが実行され、入力欄が変更されます。フォーカスを外すとblur()メソッドが実行されます。各イベントを組み合わせることで、様々な処理を動的に実装することができます。

実行結果





●選択されている要素を使用不可に変更する

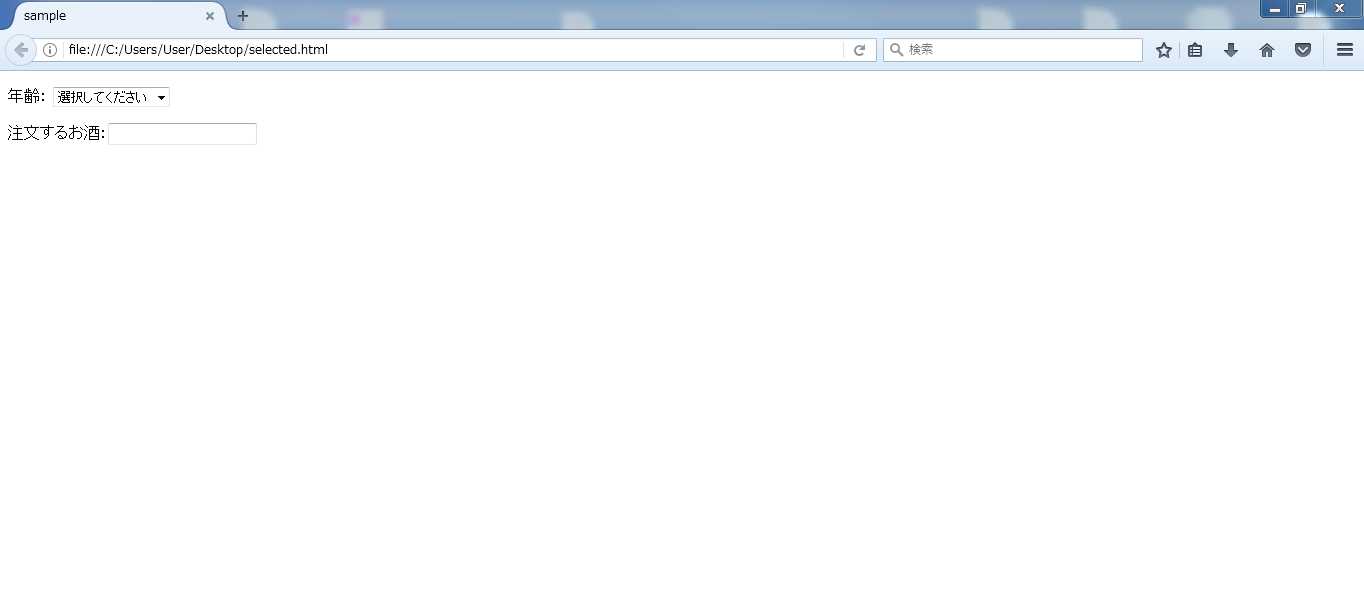
　selected()メソッドを使用し、選択されているセレクトボックスの要素を取得し、value属性の値によって処理を変更したい場合があります。以下サンプルになります。

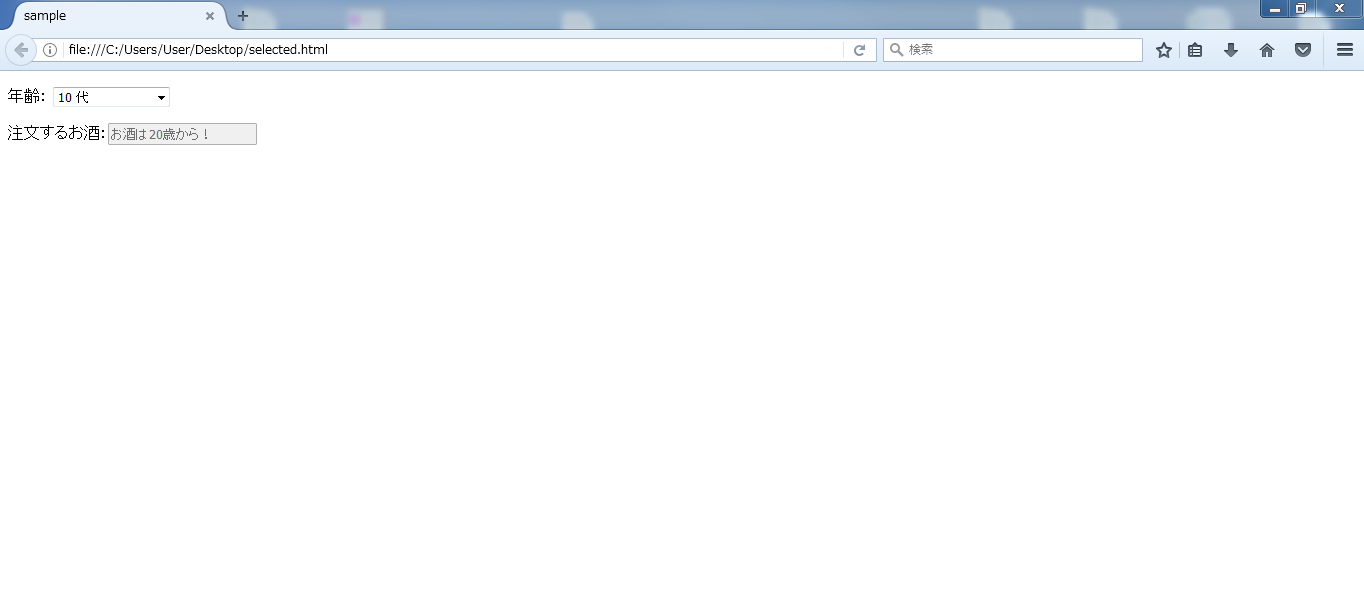
選択されている要素を使用不可に変更する.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="ja">  <head>  <meta charset="utf-8">  <title>sample</title>  <script src="jquery-1.10.2.min.js"></script>  <script>  $(function(){  $("select").change(function(){  if($(":selected").attr("value")=="10"){  $("input").val("お酒は20歳から！").attr("disabled", "disabled");  }else{  $("input").removeAttr("disabled");  }  });  });  </script>  </head>  <body>  <p>年齢：  <select name="age">  <option value="">選択してください</option>  <option value="10">10 代</option>  <option value="20">20 代</option>  <option value="30">30 代</option>  <option value="40">40 代以上</option>  </select></p>  <p>注文するお酒：<input type="text"></p>  </body>  </html> |

上記サンプルはchange()メソッドを使用し、セレクトボックスの値が変わった時に実行される文です。もし10代が選択された場合はvalue=”10”なので、if文の中身が実行されます。入力欄が選択不可になり、値が変更されます。10代以外を選択した場合はフォームの入力ができるようになります。

実行結果





●jQueryを用いたバリデーション機能

　フォームに入力された内容に漏れやミスがないかチェックを行うバリデーション機能を実装すると、ユーザーは格段に使いやすくなります。エラー箇所を分かりやすく示して、使い勝手のいいページにすることができます。注意点としましては、あくまでクライアント側で実行する簡易的な入力チェックです。セキュリティを確保するにはサーバ側での入力チェックが別途必要になりますので気を付けてください。

以下サンプルになります。

jQueryを用いたバリデーション機能.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="ja">  <head>  <meta charset="utf-8">  <title>sample</title>  <script src="jquery-1.10.2.min.js"></script>  <link rel="stylesheet" href="style.css">  <script>  $(function(){  $("form").submit(function(){  //エラーの初期化  $("p.error").remove();  $("dl dd").removeClass("error");  $("input[type='text'].validate").each(function(){  //必須項目のチェック  if($(this).hasClass("required")){  if($(this).val()==""){  $(this).parent().prepend("<p class='error'>必須項目です</p>");  }  }  //数値のチェック  if($(this).hasClass("number")){  if(isNaN($(this).val())){  $(this).parent().prepend("<p class='error'>数値のみ入力可能です</p>");  }  }  });  //エラーの際の処理  if($("p.error").length > 0){  $("p.error").parent().addClass("error");  return false;  }  });  });  </script>  </head>  <body>  <div id="container">  <h1>入力フォーム</h1>  <form action="complete.html" method="post">  <dl>  <dt>名前<span>※</span></dt>  <dd><input type="text" name="name" class="validate required"></dd>  <dt>郵便番号</dt>  <dd><input type="text" name="zip1" class="validate number"> - <input type="text" name="zip2" class="validate number"></dd>  </dl>  <p><input type="submit" value="送信"></p>  </form>  </div>  </body>  </html> |

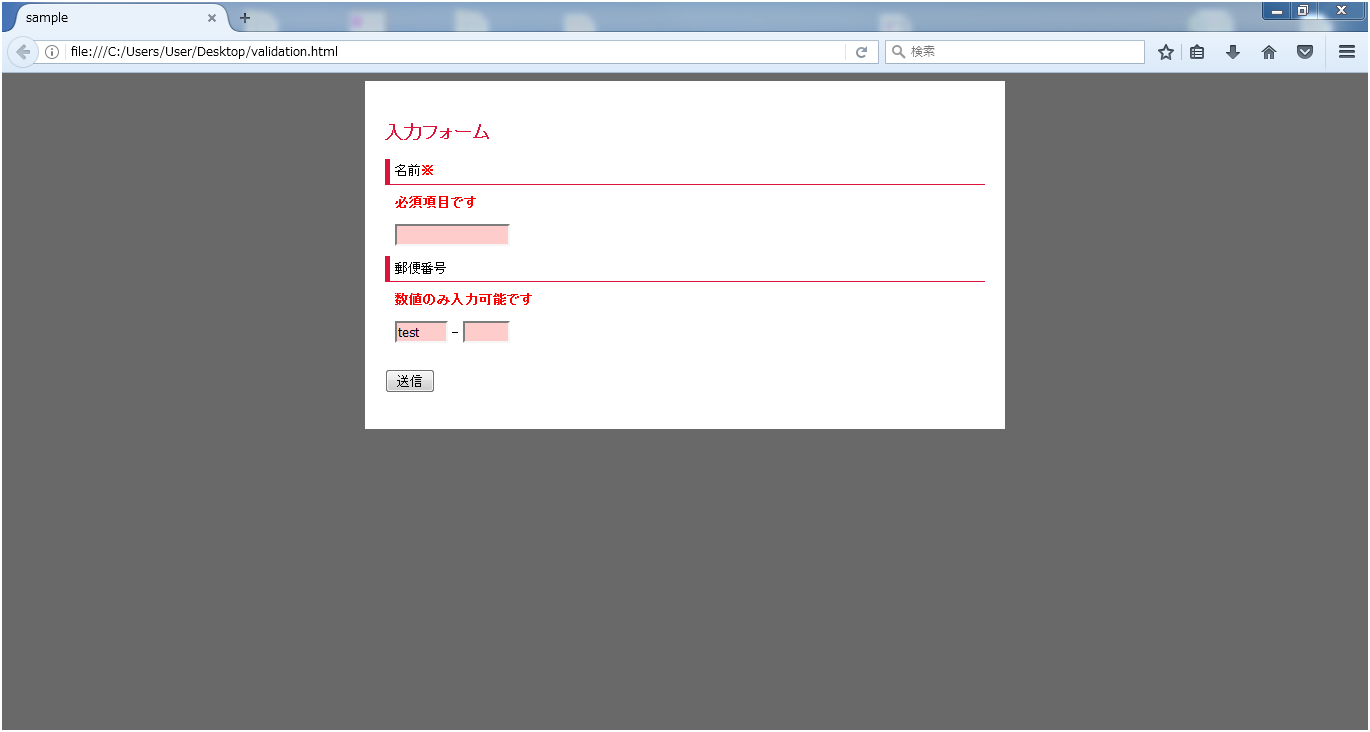
style.css

|  |
| --- |
| body{  background:#696969;  }  #container{  width:600px;  margin:0 auto;  padding:20px;  background:white;  }  h1{  margin-top:20px;  font-size:large;  color:#dc143c;  }  dl dt {  border-left:5px solid #dc143c;  border-bottom:1px solid #dc143c;  font-size:small;  margin:0;  padding:5px;  }  dl dt span{  color:red;  font-weight:bold;  }  dl dd{  font-size:small;  margin:0;  padding:10px;  }  dl dd input{  position:relative;  z-index:2;  }  dl dd label{  position:relative;  padding:5px 5px 5px 25px;  margin : 0 5px 0 -25px;  margin-left:-25px;  position:relative;  z-index:1;  }  dl dd.error input , dl dd.error label {  background:#FFCCCC;  }  \* html dl dd.error label {  background:none;  }  \*+html dl dd.error label{  background:none;  }  dl dd p.error{  margin:0;  color:red;  font-weight:bold;  margin-bottom:1em;  } |

バリデーション機能としては、必須項目の名前の入力チェックを行い、郵便番号では入力されている値が数値かどうかチェックを行っています。HTMLは、フォーム全体をdl/dt/dd要素で記述し、dt要素に「名前」などの項目名を、dd要素にinput要素などのフォーム部品を配置しています。各フォーム部品(input要素)にはバリデーションの条件を示す「validate、required」などのclass属性を付けています。CSSではフォーム部品や項目名の基本スタイルとともに、バリデーションエラー用のスタイルも定義しています。バリデーションエラー時にdd要素にclass属性「error」を追加します。エラーが発生した際は、メッセージを赤色で表示し、背景色をピンクに設定しています。

今回のサンプルは全てsubmitイベントがトリガーとなっています。送信ボタンがクリックされた際に、入力チェックが実行される仕組みです。また重要な点としましては、submitイベント内の処理では最初に、エラーを初期化します。送信ボタンが初めて押されたときはいいのですが、すでにボタンが押されてバリデーション処理が実行済みの場合、エラー時の処理が適用されている場合があります。初期化をしないとエラーメッセージが何度も追加されたり、修正済みの項目のエラーメッセージが消えなかったりしますので、removeClass()メソッドを使用し、「error」というclass属性を取り除き、remove()メソッドでエラーメッセージ(p要素)の削除をします。

実行結果



# Ajaxの概要

Ajax(エイジャックス)の登場は、一時期人気が衰えかかっていたJavaScriptが再び脚光を浴びるきっかけとなりました。Ajaxは新しい技術を使わずに既存の技術を組み合わせるだけで新しいサービスを生み出したという点でも非常に画期的でした。本資料ではAjaxで用いられている技術と、Ajaxによる非同期処理について説明します。

Ajaxを利用したプログラムについて説明する前に、Ajaxの生い立ちと背景となる技術を見ていきましょう。Ajaxを構成する技術は、いずれもAjaxのために開発されたものではなく、別の用途に使われていた技術を寄せ集めたものです。そのため、統一感がなく使いにくい場合もありますが、1つ1つの技術をおさえていくことで、理解がしやすくなるでしょう。

## Ajaxとは

　Ajaxとは「Asynchronous JavaScript and XML」の略称で、JavaScriptとXMLの技術を使って非同期で通信を行なう開発手法のことを指します。Ajaxを使うとページ遷移を行わずにWebページの一部だけを更新できます。たとえば、Google Mapsではマウス操作で地図をどこまでもスクロールできます。また、Google検索では入力候補のキーワードがリアルタイムに表示されます。

　Ajaxがこれほど注目されたのは、Webアプリケーションが苦手とされてきた操作性の悪さを、JavaScriptを積極的に用いることで克服した点にあります。Ajaxは「非同期通信を使用してWebページの一部を置き換える」という技術によって、Webページ全体を書き換えなくてはならないというWebアプリケーションの限界を打ち破り、従来のクライアント/サーバプリケーションのような画面遷移が少ない操作性の高いアプリケーションの作成を可能としました。

## AjaxとJavaScriptライブラリ

Ajaxを利用したWeb画面は、JavaScriptを多用するため、コード量が多くなる傾向にあります。そのため、JavaScriptのプログラムで煩雑になりがちなクラスの定義やDOMの操作などの処理を簡単に行うためのJavaScriptライブラリがいくつか提供されています。主なものとして、jQueryやDojo、Prototype.jsなどがあり、これらのライブラリが提供するオブジェクトやメソッドを利用することにより、プログラムのコード量を削減できます。

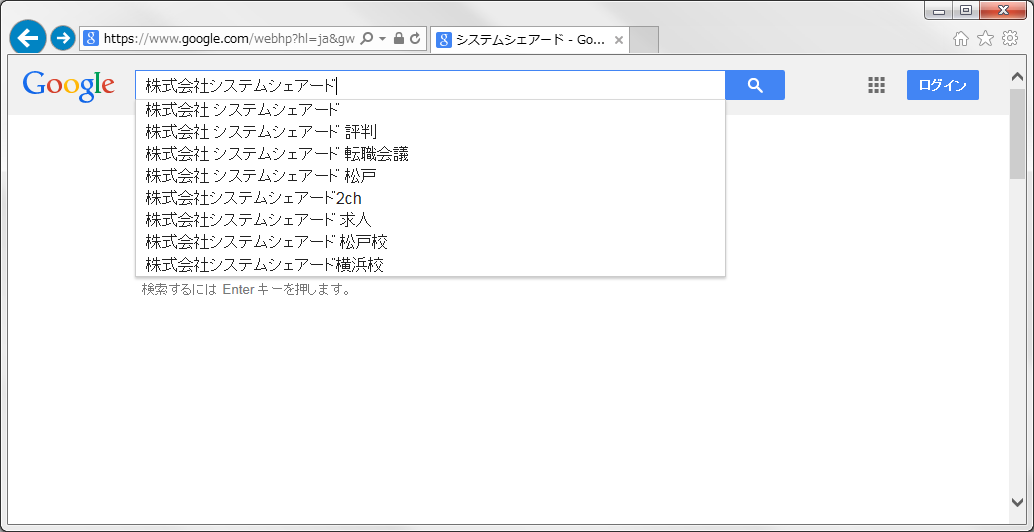
　さらに、これらのJavaScriptライブラリの中には、ツールチップやタブページ、カレンダーによる日付入力などのクライアントアプリケーションに劣らない豊かなUI(画面部品)をJavaScriptで実現しているものもあります。たとえば、jQuery UIのように表現性豊かな画面部品に強みを持ったライブラリもあり、JavaScriptライブラリを使うことで、豊かな表現力をもったWebアプリケーションが簡単に作成できるようになりました。現在では、これらのライブラリによって実現されている技術を総称して「Ajax」と呼ぶこともあります。

Ajaxの有名な利用例としては、Googleが提供する各種サービスが挙げられるでしょう。たとえば、地図サービスで有名なGoogle Mapsでは、ページ遷移せずに地図の移動や拡大縮小などを実現しています



Google Maps (<http://maps.google.co.jp/>)

　また、Google検索では、Webページを検索する際、検索語の候補をリアルタイムに表示し、入力補助を実現しています。



Google検索での入力補助(<http://www.google.co.jp/>)

いずれもWebブラウザさえあれば利用できるサービスであり大きな反響を呼びました。

## HTTP

Ajaxで利用されている技術に、HTTP(HyperText Transfer Protocol)があります。HTTPは、WebブラウザとWebサーバの間でHTMLなどのデータを送受信するための通信プロトコルで、Webアプリケーションにおける通信では一般的に用いられています。

HTTPはJavaなどWebアプリケーションの開発言語の多くでサポートされていますが、ほとんどの言語ではHTTPの仕様が隠蔽された形でサポートされています。そのため、それらの言語を使用する場合はHTTPの仕組みを理解していなくてもかまいません。それに対して、Ajaxで利用するXMLHttpRequestオブジェクトではHTTPコマンドやHTTPのフォーマット(リクエストやレスポンスのデータ形式)を隠蔽せずにそのままの形式に近い形で扱うのが特徴です。したがって、jQueryなどのライブラリを使用せずにAjaxを使用する場合はHTTPの知識が必要となります。

リクエストとレスポンス

　HTTPはクライアント(Webブラウザ)からサーバ(Webサーバ)に対して要求を送信し、その要求にサーバが応答するというシンプルな仕組みで構成されます。クライアントからの要求のことを「リクエスト」、サーバからの応答のことを「レスポンス」と呼びます(図1)。1つの処理は、必ず1回のリクエストと1回のレスポンスで構成され、レスポンスが返ってきた段階で処理が完結します。HTTPでは1つのクライアントとの間の通信を維持できないため、ログイン状態を保持する場合など、複数のリクエストの関連を持ち続ける場合はサーバ側から何らかの仕組みを作り込む必要があります。

リクエスト

(要求：処理の依頼)

レスポンス

(応答：処理の結果)

サーバ

(Webサーバ)

クライアント

(Webブラウザなど)

リクエストとレスポンス

## jQueryによるAjax

　jQueryはAjaxのためのユーティリティとして下記のメソッドを提供しています。これらのメソッドにより手軽にAjaxを使用することができます。

jQueryが提供するAjax用のユーティリティメソッド

|  |  |
| --- | --- |
| **メソッド名/構文** | **説明** |
| load(url[,data][,callback]) | jQueryオブジェクトが選択している要素にAjaxで取得したHTMLを挿入する |
| getScript(url[,callback]) | JavaScriptを取得して実行する |
| getJSON(url[,data][,callback]) | JSONを取得する |
| get(url[,data][,callback]) | GETリクエストを送信する |
| post(url[,data][,callback]) | POSTリクエストを送信する |
| ajax(settings) | jQueryが提供するAjax用のユーティリティのベースとなるメソッド。他のメソッドでは不可能な細かい処理が必要な場合に使用する |

　jQueryが提供するAjax用のユーティリティメソッドは、最終的に$.ajax()メソッドを呼び出します。つまりこのメソッドがjQueryによるAjaxサポートの基礎になっているということです。

　通常、$.ajax()メソッドを使用するよりも後述するユーティリティメソッドを使用するほうが簡単に処理を記述できますが、これらのユーティリティメソッドでは実現できない処理を行う必要がある場合は直接$.ajax()メソッドを使用します。

　次に$.ajax()メソッドの簡単な使用例を示します。なお、動作確認を行う場合は、「$.ajax()メソッドを使用したAjaxの例.html」、「books.txt」をEclipseの動的Webプロジェクトの同階層に配備してから実行してください。

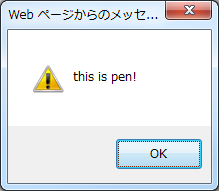
$.ajax()メソッドを使用したAjaxの例.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="ja">  <head>  <meta charset="utf-8">  <title>sample</title>  <script src="jquery-1.10.2.min.js"></script>  <script>  //books.htmlをGETリクエストで取得する  $.ajax({  url: 'books.txt' ,  type:'GET' ,  }).done(function(data, status, req){  // 成功したときに呼び出される  alert(data);  }).fail(function(req, status, error){  // 失敗したときに呼び出される  alert('失敗: ' + error);  });  </script>  </head>  <body>  </body>  </html> |

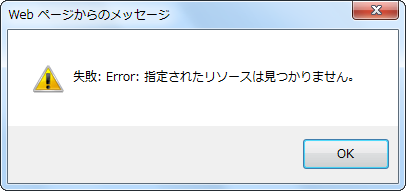
books.txt

|  |
| --- |
| This is pen! |

実行結果



処理が成功した場合



　　　　処理が失敗した場合

一見警告ダイアログが表示されただけのように思えますが、実際にはurlに指定したリソースに対し、非同期通信を行っています。今回はプレーンテキストであるbooks.txtに対しリクエストを送信しているため、固定文字列のみがレスポンスとして返されますが、servletなどを指定することも可能となっています。

なお、$.ajax()メソッドの引数に指定可能なパラメータには次のようなものがあります。

$.ajax()メソッドに指定可能な主なパラメータ

|  |  |
| --- | --- |
| **パラメータ名** | **説明** |
| url | 通信先のURLを指定する |
| async | 非同期で通信を行うかどうかをtrueまたがfalseで指定する。falseを指定した場合、通信が完了するまで処理をブロックする。省略する場合はtrueになる |
| cache | GETリクエストのレスポンスをキャッシュするかどうかをtrueまたはfalseで指定する。省略した場合はtrueになる。ただし、dataTypeにscriptまたはjsonpを指定した場合はfalseになる |
| contentType | リクエストのコンテンツタイプを指定する。省略した場合はapplication/x-www-form-urlencodedになる |
| context | コールバック関数内でthisが示すオブジェクトを指定する |
| data | 送信するパラメータをクエリ文字形式の文字列またはハッシュで指定する。 |
| dataType | サーバから返却されるデータの型を次のいずれかから指定する。省略した場合は自動的に判別される  xml  html  script  json  jsonp  text |
| global | Ajaxに関するイベントを発行するかどうかをtrueまたはfalseで指定する。デフォルトはtrueになる |
| ifModified | if-Modified-SinceヘッダとLast-Modifiedヘッダを使用して更新チェックを行なうかどうかをtrueまたはfalseで指定する。省略した場合はfalseになる |
| jsonp | JSONPのコールバック名がcallback以外の場合にコールバック名を指定する |
| jsonpCallback | JSONPのコールバック関数名を指定する |
| password | HTTP認証が必要な場合のパスワードを指定する |
| processData | dataにオブジェクトを指定した場合にクエリ文字列形式に変換して送信するかどうかをtrueまたはfalseで指定する。省略した場合はtrueになる |
| scriptCharset | Ajaxで取得したスクリプトを読み込む際の文字コードを指定する。dataTypeがjsonまたはjsonpで、かつHTMLと取得するスクリプトの文字コードが異なる場合のみ指定する必要がある |
| timeout | 通信のタイムライン時間をミリ秒で指定する |
| type | GETまたはPOSTを指定する。省略した場合はGETになる |
| username | HTTP認証が必要な場合のユーザー名を指定する |

# Ajaxの実装

これまでに学習したServlet/JSPを併用し、実際に非同期通信を行ってみましょう。Ajaxでは、サーバに対してパラメータ等の情報をAjaxリクエストとして送信し、ブラウザ等のクライアントはAjaxレスポンスとして、リクエストの結果を受け取り、受け取ったAjaxレスポンスのデータを元にページに変更を加えます。

## Ajaxとデータベース

　Ajaxを利用し、データベースと接続し、画面上に非同期通信で値を表示するサンプルを紹介します。今までのServlet/JSPと同様にEclipseの動的Webプロジェクトで作成していきます。(サンプルではプロジェクト名をAjaxSampleとしています)

以下、サンプルコードになります。また、サンプル用のデータベースも用意します。

* スキーマとテーブル

ユーザー名 ：ajax\_user

パスワード ：systemsss

権限 ：「SELECT ANY DECTIONARY」権限以外の全ての権限

ajax\_sampleテーブル

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | **論理名称** | **物理名称** | **物理データ型(桁数)** | **PK** | **制約** |
| 1 | ID | id | NUMBER(4) | ○ |  |
| 2 | 名前 | name | VARCHAR2(30 CHAR) |  | NOT NULL |
| 3 | お金 | money | NUMBER(8) |  | NOT NULL |

SQL文

|  |
| --- |
| --ユーザー名systemでログイン後、新規にスキーマを作成する。  CREATE USER ajax\_user IDENTIFIED BY systemsss;  GRANT ALL PRIVILEGES TO ajax\_user;  --一旦データベースとの接続を切断し、先程作成したユーザー名で接続する。  --接続後、ajax\_sampleテーブルを作成する。  CREATE TABLE ajax\_sample (  id NUMBER(4) PRIMARY KEY,  name VARCHAR2(30 CHAR) NOT NULL,  money NUMBER(8) NOT NULL  );  --テーブル作成後、各テーブルに動作テスト用レコードを挿入する。  INSERT INTO ajax\_sample VALUES (1, '山田太郎', 10000);  INSERT INTO ajax\_sample VALUES (2, '田中健一', 25000);  INSERT INTO ajax\_sample VALUES (3, '鈴木花子', 500000);  COMMIT; |

続いてクラスファイルを作成していきます。

動的Webプロジェクトで作成します。(サンプルではパッケージ名を jp.co.sss.ajaxSampleとしています)

AccountDAO.java

|  |
| --- |
| import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.SQLException;  public class AccountDAO {  public static Connection getConnection() {  Connection con = null;  try {  Class.forName("oracle.jdbc.driver.OracleDriver");  con = DriverManager.getConnection(  "jdbc:oracle:thin:@//localhost:1521/xe", "ajax\_user",  "systemsss");  } catch (Exception e) {  e.printStackTrace();  }  return con;  }  public static String findById(String inputParam) throws SQLException {  Connection con = null;  PreparedStatement ps = null;  try {  con = getConnection();  // テーブル名はajax\_sample、カラムはid、name、moneyの3つ  ps = con.prepareStatement("SELECT \* FROM ajax\_sample WHERE id = ?");  ps.setString(1, inputParam);  ResultSet rs = ps.executeQuery();  StringBuilder sb = new StringBuilder();  if (rs.next()) {  sb.append(" id: ");  sb.append(rs.getInt("id"));  sb.append(" name: ");  sb.append(rs.getString("name"));  sb.append(" money: ");  sb.append(rs.getInt("money"));  }  return sb.toString();  } finally {  if (ps != null) {  try {  ps.close();  } catch (SQLException e) {  e.printStackTrace();  }  }  if (con != null) {  try {  con.close();  } catch (SQLException e) {  e.printStackTrace();  }  }  }  }  } |

IndexAction.java

|  |
| --- |
| import java.io.IOException;  import java.sql.SQLException;  import javax.servlet.RequestDispatcher;  import javax.servlet.ServletException;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  public class IndexAction extends HttpServlet {  private static final long serialVersionUID = 1L;  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  String inputParam = request.getParameter("id");  try {  String outputParam = AccountDAO.findById(inputParam);  request.setAttribute("output", outputParam);  RequestDispatcher dispatcher = request.getRequestDispatcher("/list.jsp");  dispatcher.forward(request, response);  } catch (SQLException e) {  e.printStackTrace();  }  }  } |

続いてweb.xmlを修正します。

web.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"  id="WebApp\_ID" version="2.5">  <display-name>AjaxSample</display-name>  <welcome-file-list>  <welcome-file>index.jsp</welcome-file>  </welcome-file-list>  <servlet>  <description></description>  <display-name>IndexAction</display-name>  <servlet-name>IndexAction</servlet-name>  <servlet-class>jp.co.sss.ajaxSample.IndexAction</servlet-class>  </servlet>  <servlet-mapping>  <servlet-name>IndexAction</servlet-name>  <url-pattern>/IndexAction</url-pattern>  </servlet-mapping>  </web-app> |

続いてJSPファイルの作成を行います。

list.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  ${output} |

index.jsp

|  |
| --- |
| <%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  <script type="text/javascript"  src="http://code.jquery.com/jquery-1.10.2.js"></script>  <script type="text/javascript">  $(function() {  $("input[name=idSend]").click(function() { ❶  $.ajax({ ❷  type : "GET", ❸  url : "<%=request.getContextPath()%>/IndexAction", ❹  data : { ❺  id : $("input[name=accountId]").val()  }  }).done(function(data, status, req){ ❻  $("h2").html(data);  });  });  });  </script>  </head>  <body>  <h1>検索したいidを入力してください</h1>  <form>  <input type="text" name="accountId" />  <input type="button" name="idSend" value="AJAX通信" />  </form>  <h2></h2>  </body>  </html> |

注目してほしいのはAjaxを使用しているindex.jspです。

❶ではname=”idSend” のボタンがクリックされたら、実行されます。

ここまでは今まで使用したjQuery文と変わりません。

❷でAjaxが実行されます。$.ajax()メソッドの引数には色々なパラメータを指定することができますが、基本の種別があります。下記を参考にしてください。

|  |  |
| --- | --- |
| **名前** | **説明** |
| url | リソースのURLを指定します。  Servlet/JSP(フレームワーク)の場合はweb.xmlで設定したpathを指定します。 |
| type | HTTPメソッドの種別を”GET”か”POST”で指定します。  Servlet/JSPの場合はdoGet()メソッドを呼ぶか、doPost()メソッドを呼ぶか指定することができます。 |
| data | サーバに送信するフォームデータを指定します。  例: data: {id:”test”, pw:”pass”} と指定するとidとpwという名前を紐づけてリクエストを実行します。 |
| timeout | timeoutに指定した時間(ミリ秒)が経過しても通信が完了しない場合はエラー処理(fail(),error)が実行されます。  スマートフォン向けのサイト等では、電波の影響で通信に時間が掛かる場合などにメッセージを表示するので現在では使用機会がとても多いオプションです。  ※Ajaxのタイムアウト値は10秒～30秒が目安になります。 |

❸ではHTTPメソッドを”GET”で指定しているので、urlで指定したServletのdoGet()メソッドをリクエストします。

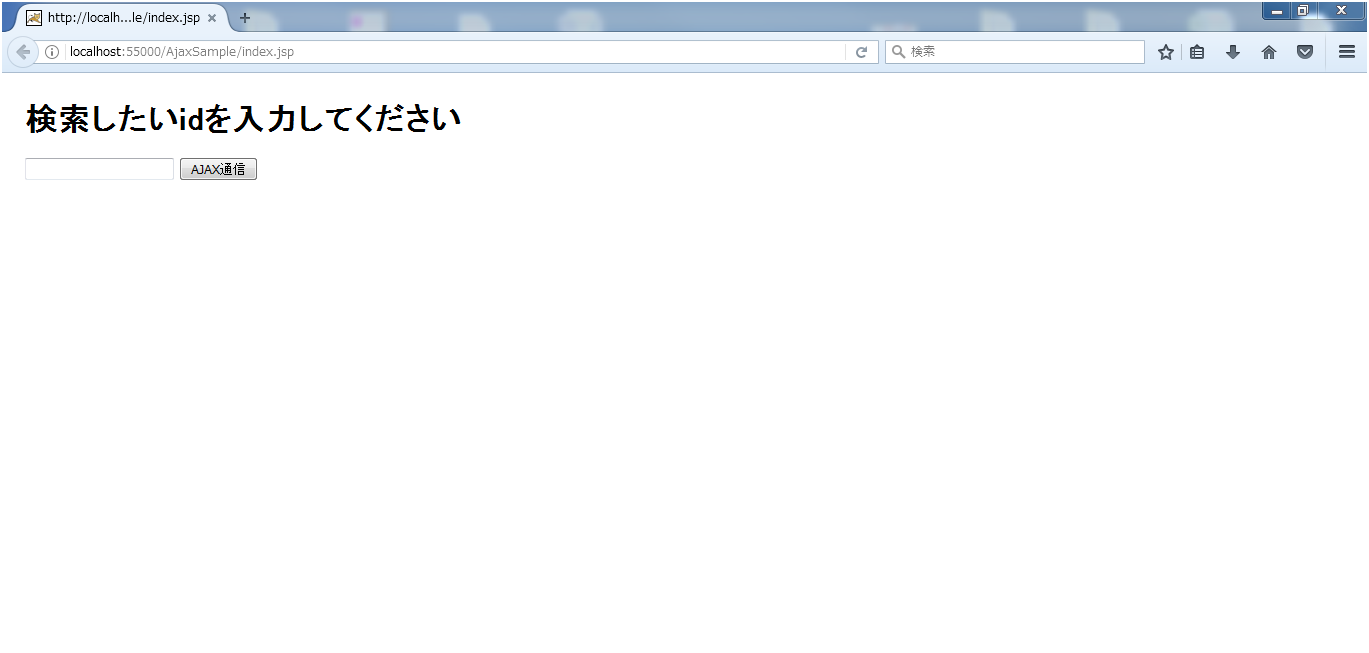
❹ではリソースのURLを指定しています。今回の場合はweb.xmlで記述したpath”/IndexAction”を指定しています。

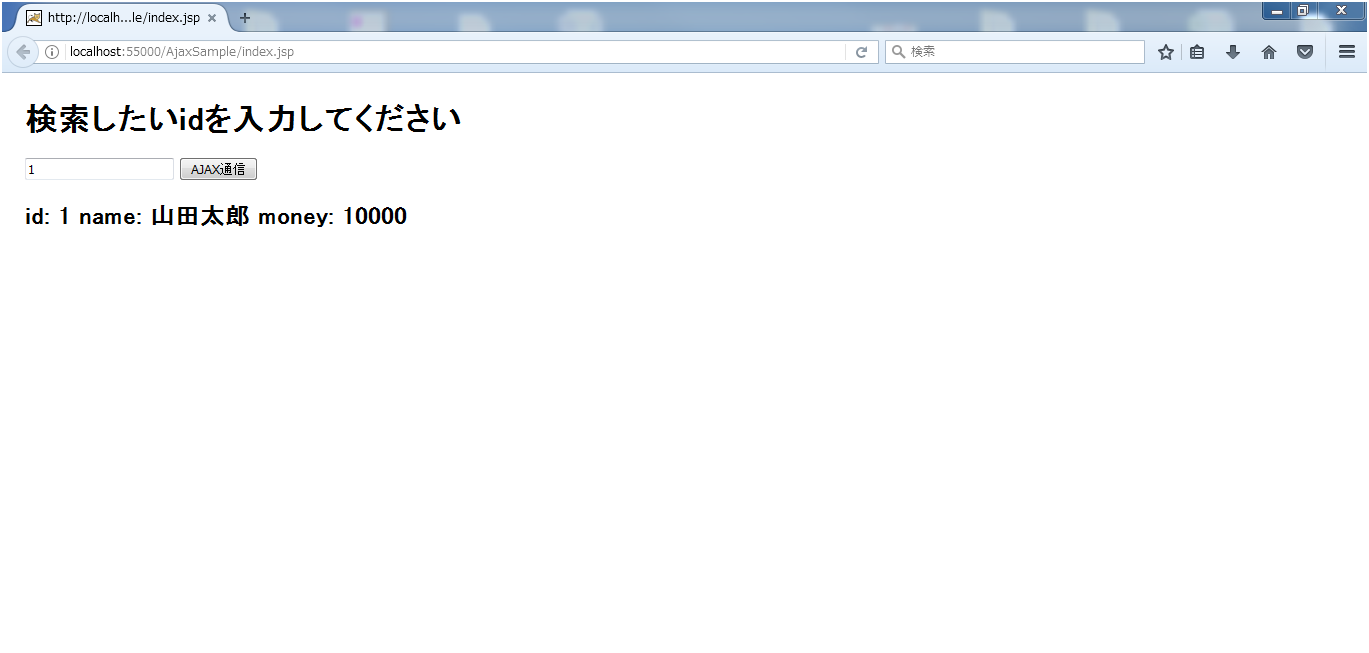
❺ではサーバ側に送信するフォームデータを指定します。今回の場合、name=”accountId”の値をサーバに送信します。<input type=”text” name=”test” />とした場合はdataオプションにname=”test”と指定します。

❻の.done ()ではajax()メソッドの戻り値を引数として受け取ります。今回の場合は受け取った引数の値を画面に表示しています。

以下実行画面の例になります。画面遷移をしていないので、値を表示してもURLが変わっていないことに注目してみてください。

実行結果

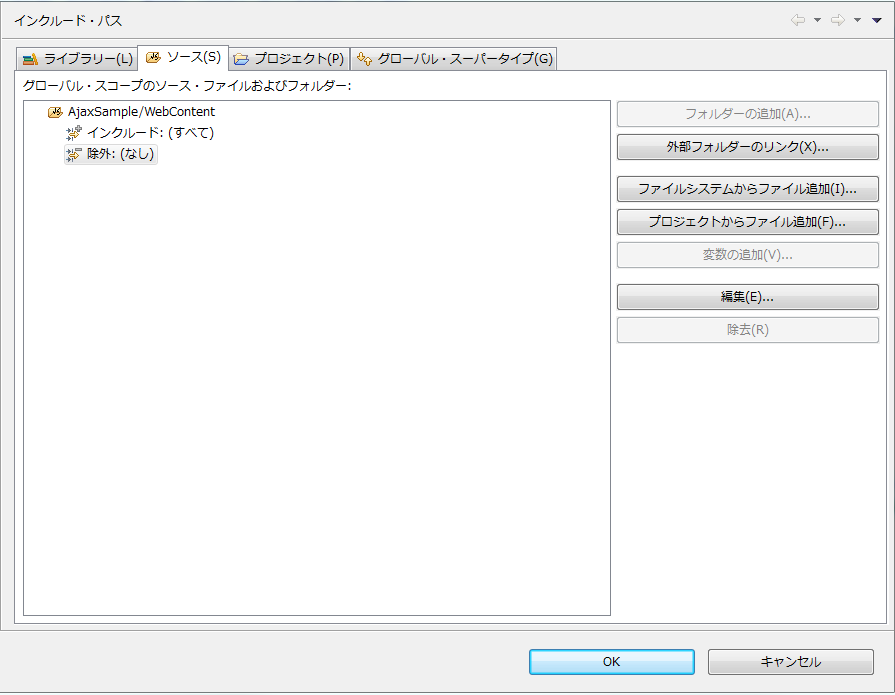




## EclipseとjQuery

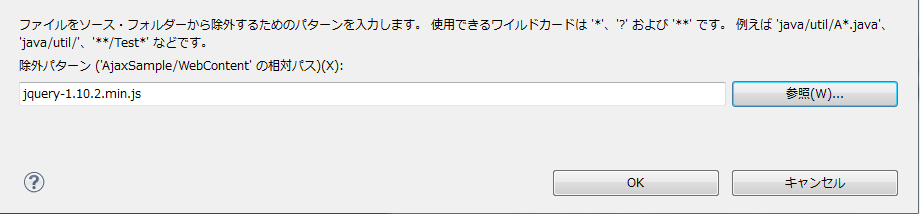
Eclipseで動的Webプロジェクトを作成し、jQueryを使用すると、本体のjsファイルが構文エラーになってしまいます。そのままでも問題なくプロジェクトの実行は可能ですが、プロジェクトに×印が付いてしまうので、Eclipseの除外設定を次の手順で行います。

* 1. 該当プロジェクトを右クリックし、プロパティーを選択。
  2. 「JavaScript」を開く。
  3. 「インクルード・パス」を開き、ソースタブを選択。
  4. フォルダを指定し、「除外」を選択後、右側の「編集」ボタンをクリック。

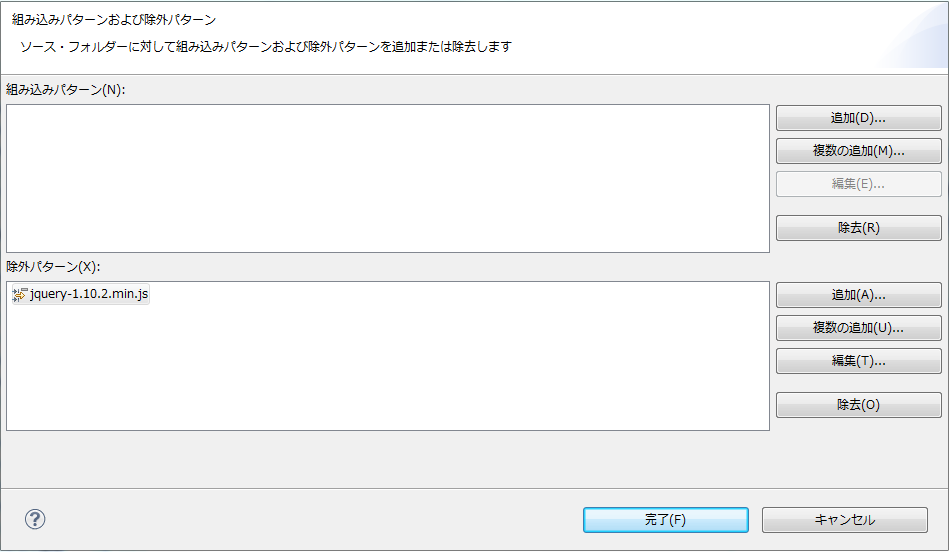


* 1. 除外パターンテキストボックス横の「追加」ボタンをクリック。
  2. 「参照」ボタンをクリックし、\*\*/jquery\*.jsを指定し、「OK」ボタンをクリック。

(例ではjquery-1.10.2.min.jsを指定)



* 1. 「完了」ボタンをクリックする。



jQueryのプラグインだけではなく、外部のjsファイルは、問題なく起動するのにEclipseでは赤い警告が発生してしまうことがあります。Javaファイルのコンパイルエラーと見分けづらくなるので、その際は上記と同様に除外設定を行うようにしてください。