はじめに

数年前であれば Flash を使用して表現していたインタラクティブな演出やアニメーションが、ここ最近は Flash に対応していないスマートフォンやタブレットなどの多様化に伴って、そのほとんどが JavaScript で表現されるようになりました。

このような大きな潮流の中で注目を集めているのが「**jQuery**」です。複雑なコードになりがちな JavaScript を手軽に扱うことのできる jQuery は、今では Web クリエイターにとって必要不可欠な存在になっています。それにより、関連書籍も多数存在していますが、反面そのほとんどは開発者が一人で執筆しているため、プログラムを深く理解していることが前提の解説となっているものが多く、日常的にプログラミングを行っていないデザイナーにとってはとっつきにくい内容になっているものが多いのが現状です。

そこで本書では「**プログラム経験の少ないデザイナーでも容易に理解できる構成**」を第一の目標として執筆を行いました。技術的な側面の解説やプログラミング手法、jQuery の詳細解説については、現場の第一線で日々業務を行っている経験豊富なプロフェッショナルの開発者が執筆を担当していますが、その一方で、執筆期間中は jQuery の知識が全くないデザイナーや、プログラミング未経験者を執筆チームに加えることで、これから jQuery を学ぶ人が「どこにつまづき」、「何に疑問を抱くのか」を常に検証し、そしてブラッシュアップしていきました。こういった日々の作業によって、確かな技術書として、より多くの人にとって最良の書籍となることを目指しました。この点は本書の強みの 1 つです。

また、Web デザイン関連の書籍においては、学習過程で使用するサンプルのクオリティも非常に大切だと考えています。そのため本書では、ただ単に解説を補足するような単純すぎるサンプルの掲載を避け、多くの人が「作ってみたい」、「仕事でも使えそうだ」と感じるようなサンプルであることを目指しました。本書内で使用するサンプルはすべて、第一線で活躍するアートディレクター監修の元、現場で多くのプロモーションサイトを制作しているインタラクションデザイナーが制作しています。ぜひ、実際に手を動かしてこれらのサンプルを制作し、jQuery をより深く習得していただけると嬉しいです。

今後、世の中のデジタル化が進むことは間違いありません。そして、その社会を作っていくうえで重要な役割を担うのは、みなさんをはじめとするクリエイター達です。本書が、みなさんがこれから制作していくであろうサイトや作品の一助となり、新しい表現が Web 上に登場することを心より祈っています。

最後に、通常の案件業務がありながらも、社内をマネジメントしてくれた仲村くん、素晴らしいサンプルを作ってくれた許くん、奥村さん、そして、執筆を担当した、安友さん、丈さん、鈴木慶太郎くん、本当にお疲れさまでした。そして執筆チームの面倒を丁寧にみてくださった SB クリエイティブの岡本晋吾さん、そして、何よりこの本を手に取ってくれた読者の方に感謝いたします。

2013 年 11 月
株式会社シフトブレイン
代表取締役　加藤琢磨

Contents

Appendix

jQuery リファレンス 291

Chapter 02

01 体験することからはじめよう！

前章でも触れましたが、jQuery はとてもシンプルな JavaScript ライブラリであるため、初心者の方でも要点さえ理解すればすぐに使用できるようになります。しかしその反面、多機能でもあるため、基礎知識がない状態で膨大なリファレンスを先頭から読み進めたり、Web 上に散らばっている断片的な情報を拾い読みする方法だけでは、「何をどう使えば良いのか」、「何から覚えたら良いのか」がわかりにくいという一面もあります。初心者の方が効率良く jQuery を習得するためには、**適切な順番で jQuery の機能や使用方法を学んでいくことが必要**です。そこで、まずは以下の 3 つのステップで実際に手を動かしながら、jQuery の基礎とそれに関連する JavaScript の基礎を学んでいきましょう。

STEP 1 **CSS のスタイルを変更してみよう**
STEP 2 **タイミングをコントロールしてみよう**
STEP 3 **アニメーションさせてみよう**

この 3 つのステップではいずれもシンプルな機能しか利用しませんが、これらは jQuery における「**見た目の表現**」に関する基礎となります。そのため、読み進めていくと表現の幅が広がっていく感覚を実感できると思います。

サンプルファイルについて

本章では、前節でダウンロードした本書のサンプルファイルのうち、**[Chapter02]** フォルダ内のファイルを使用します。フォルダ内の **index.html** をブラウザで開くと以下のようなシンプルなページが表示されます。

Fig index.html

index.html をテキストエディタなどで開き、内容を確認しておきましょう。

html サンプルファイルの HTML　index.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<title>Creative jQuery</title>
<link rel="stylesheet" href="./css/normalize.css">
<link rel="stylesheet" href="./css/main.css">

<script src="./js/vendor/jquery-1.10.2.min.js"></script>
<script src="./js/vendor/jquery-ui-1.10.3.custom.min.js"></script>
<script src="./js/main.js"></script>
</head>
<body>

<header class="page-header" role="banner">
    <h1>Creative jQuery Sample</h1>
</header>

<div class="page-main" role="main">
    <div id="typo">
        <div class="inner">Creative jQuery</div>
    </div>
</div>

<footer class="page-footer" role="contentinfo">
    <small class="copyright">COPYRIGHT &copy;
        <a href="http://www.shiftbrain.co.jp" target="_blank">
            SHIFTBRAIN Inc.</a>
    </small>
</footer>

</body>
</html>
```

❶ ❷

head 要素内で複数の CSS ファイルを読み込んでいますが❶、**normalize.css** はクロスブラウザ対応のためのファイルです。このファイルについては P.307 を参照してください。本章を読み進めるうえでは特に詳細を把握しておく必要はありません。

また、3 つの JavaScript ファイルを読み込んでいますが❷、これらのうち、本章でみなさんが使用するのは **main.js** のみです[*1]。**main.js** は、みなさんが本書を読み進めながらプログラムを記述していくために用意した空のファイルです（何も記述されていません）。

＊1 **main.js** よりも先に（上部で）読み込んでいる 2 つのファイルは、jQuery のファイルと jQuery の拡張ライブラリである「jQuery UI」です。jQuery UI については、これを使用するタイミングで別途説明します。

本章の解説の流れ

本章では、基本的に以下の流れで解説を進めていきます。

（1）main.js をエディタで編集する
（2）編集後にブラウザで開き、動作確認を行う

書籍を読むだけ、サンプルを見るだけではなかなか身につきません。そのため、本章の **main.js** は、あえて空っぽの状態でスタートします。入力するのが面倒だと感じる方もいるかもしれませんがそれがプログラムというものです。きちんと技術を身につけるためにも、実際に手を動かしてプログラムを書いていくことをお勧めします。それでははじめましょう！

Point! 本章のサンプルファイルの対応ブラウザ

本章のサンプルについては、IE8 などの古いブラウザではサポートされていない CSS プロパティが一部に含まれています。そのため、動作を確認する際はなるべく新しいブラウザを使用してください。なお、5 章以降で解説するサンプルでは一部の古いブラウザにも対応しています。

Column

JavaScript の記述場所

JavaScript は、**HTML 内に直接記述する方法**と**外部ファイルに記述して読み込む方法**がありますが、本書では外部ファイル（**main.js**）に記述して読み込む方法を採用しています。

HTML に記述する方法は手軽ではあるのですが、HTML と JavaScript が 1 つのファイルにまとまってしまうため、Web サイトが複雑になればなるほど、管理性や更新性が損なわれてしまいます。現在では外部ファイルに記述して読み込む方法が一般的に採用されています。

なお、外部ファイルに記述して読み込む方法の場合は、js ファイルの読み込む順番に注意が必要です。上記では、jQuery の js ファイルを先に読み込んでから、**main.js** を読み込んでいるのがわかると思います。このように、先に jQuery を読み込んでおかないと、**main.js** のプログラムから jQuery を使用することはできません（エラーになります）。

STEP 1

02 CSS のスタイルを変更してみよう

まずは、jQuery を使用して **CSS のスタイル**を変更してみます。

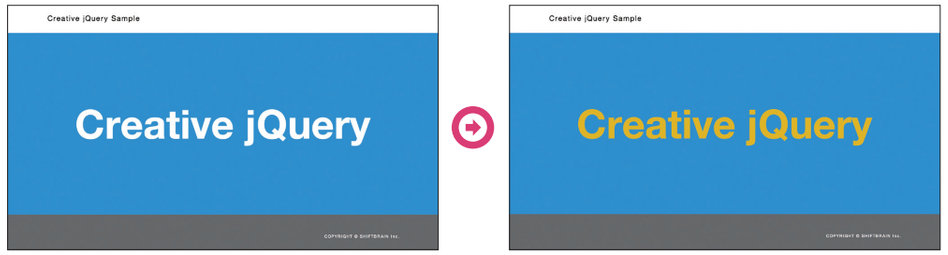
main.js をエディタで開き、以下のコードを追加してください。この段階ではコードの意味を理解する必要はありません。後で詳しく解説します。

JavaScript CSS のスタイルの変更（#typo の要素の変更） main.js

```
$(function(){
    $('#typo').css('color', '#ebc000');
});
```

上記のコードを追加したら **main.js** を保存して、ブラウザをリロードします。すると下図のように **#typo** の要素の文字色が変わります[*1]。

Fig 「Creative jQuery」と書かれた箇所の変更



コードの解説

それでは先ほど追加したコードの内容を見ていきましょう。理解しやすいようにコードを以下の 2 つのパートに分けます。

JavaScript 1 つめのパート main.js

```
$(function(){
    //2つめのパートが記述されている
});
```

＊1 **#typo** の要素が変化しない場合は、記述ミスをしている可能性があります。ブラウザの開発者ツールを起動してコンソールを確認してみましょう（P.32）。

JavaScript 2 つめのパート　main.js

```
    $('#typo').css('color', '#ebc000');
```

1 つめのパートのコードの意味は後で説明します（⇒ P.46）*2。ここでは 2 つめのパートを見てください。CSS をご存じの方であれば、コードの細かい意味はわからなくても、どのような処理をしているのか想像できたのではないでしょうか。このコードの内容は以下の通りです。

Fig スタイルを変更するコードの内容

```
$('セレクタ').css('プロパティ', '値');
```

Table 書き換え可能な箇所

要素	説明
セレクタ	ここには CSS と同じセレクタを指定できる。サポートされていないものも多少あるが、特定の要素を指定するのに困ることはない
プロパティ	CSS のプロパティを指定する
値	プロパティに設定する値を指定する

つまり、このコードは「**“セレクタ” に該当する要素の CSS の “プロパティ” を “値” に変更する**」という処理を行います。そうです、CSS の知識をそのまま使用できるのです*3。世の中のインタラクティブな動きのある Web サイトは、基本的にはこのように「JavaScript で CSS を随時書き替える」ことで実現されています*4。このコードはそのほんの入口ですが、とても重要です。

サンプルコードを書き替えてみよう！

本章のサンプルは HTML の要素が少ないため、あまりいろいろな実験はできませんが、ここで実際に以下のような感じでセレクタやプロパティ、値を自由に書き替えてみてください。HTML 内の各要素を jQuery で操作できることがわかります。

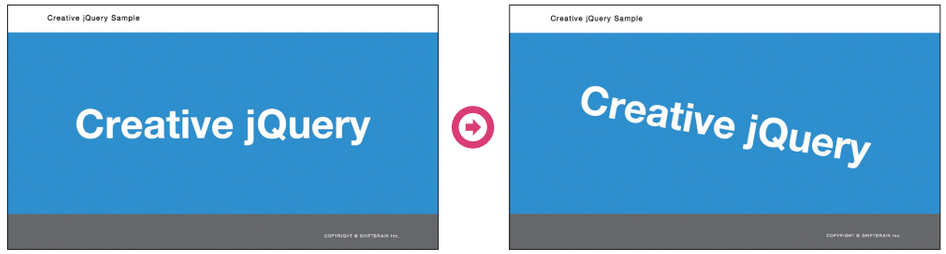
JavaScript 変更例①　main.js

```
$(function(){
    $('#typo .inner').css('transform', 'rotate(10deg)');
});
```

＊2 1 つめのパートのコードにはコメントアウトが含まれています。// 以降の文字列は行末までコメントアウトされます。複数行のコメントアウトは CSS と同じ /*《コメント》*/ です。
＊3 jQuery では、該当する各要素の style 属性の情報を書き換えることで CSS を適用します。そのため、CSS で「!important」を使用している項目は、その設定が優先されるので注意してください。
＊4 Flash や CSS3、canvas などによる表現は除きます。

上記のコードを保存してブラウザをリロードすると以下のようになります。

Fig 変更例①の表示



このように、CSS3 プロパティである **transform** プロパティの **rotate()** 関数も問題なく使用できます*5。

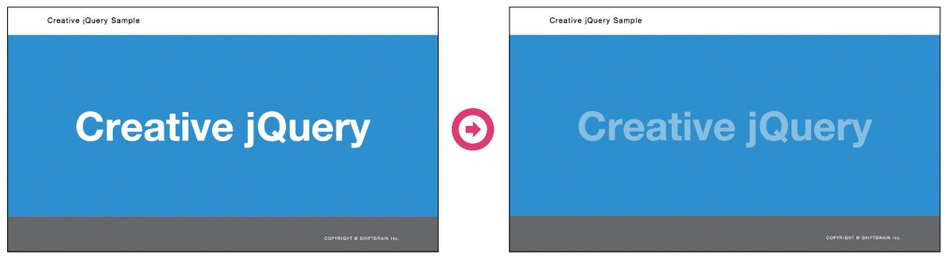
もう 1 つ例を見てみましょう。

JavaScript 変更例②　main.js

```
$(function(){
    $('.page-main > div:nth-child(1) .inner').css('opacity', 0.5);
});
```

上記では無意味に長いセレクタで［変更例①］と同じ **#typo .inner** を選択しています。ご覧の通り、jQuery は古いブラウザでは使えない **:nth-child()** などの CSS3 セレクタにも対応しています。ここでは **opacity**（要素の不透明度）を **0.5**（50%）に指定しています*6。上記のコードを保存してブラウザをリロードすると以下のようになります。

Fig 変更例②の表示



＊5 **transform** プロパティには通常はベンダープレフィックスが必要ですが（執筆時点）、プロパティのベンダープレフィックスは jQuery が自動的に付加してくれます。ただし、「値」にベンダープレフィックスが必要な場合は付加した文字列を渡す必要があります。
＊6 IE8 以下では、**opacity**（不透明度）の指定方法が異なりますが、jQuery が自動的に差異を吸収してくれるので統一的に「**opacity**」と 0 ～ 1 の数値で指定できます。

Column

数値と文字列の違いに注意する

上記でいくつかの指定例を示しましたが、値の指定方法が２通りあることに気づいたでしょうか。最初の例では「`css('color', '#ebc000')`」のように値を「`'`」（シングルクォーテーション）で囲っていますが、２つめの例では「`css('opacity', 0.5)`」のように直接数値を指定しています。これらの見た目はそれほど変わりませんが、プログラム的には大きく異なるので注意してください。

jQuery では「`'`」や「`"`」には以下のような役割があります。

> **重要** 「`'`」（シングルクォーテーション）または「`"`」（ダブルクォーテーション）で囲まれた箇所は**文字列**として認識されます。そのため文字列以外、例えば**数値**を指定する場合は「`'`」や「`"`」で囲まずに値を指定することが必要です。

JavaScript はとても寛容なプログラム言語なので、数値を指定すべき箇所に「`'`」を記述して指定しても（文字列を指定しても）自動的に数値に変換して処理してくれる場合があるので、必ずしもエラーになるわけではありませんが、エラーになる場合もあるので、なるべく上記のルールを守り、数値を指定すべきところには数値を指定し、文字列を指定すべきところには文字列を指定するように心がけてください。

jQuery の最も基本的な構文の解説

CSS のスタイルを変更する方法が何となくわかった所で、コードの内容をもう少し掘り下げて見ていきます。ここからは少しボリュームがありますが、少しずつ理解を深めていきましょう。

まずは先ほどパート分けした２つめのパートを使用して、**jQuery の基本的な構文と考え方**を解説します。

JavaScript ２つめのパート　main.js

```
$('#typo').css('color', '#ebc000');
```

これを jQuery 的に解釈すると「`$('#typo')` というモノ（**命令の対象**）に対して、`css('color', '#ebc000')` という命令を指示している（**命令の内容**）」と捉えることができます。なぜこのような解釈になるのでしょうか。１つずつ見ていきましょう。

命令の内容：jQuery メソッドと引数

まずは**命令の内容**の部分を見てみます。

JavaScript 命令の内容　main.js

```
css('color', '#ebc000');
```

コードの解説で触れた通り、`css()` では「`()`」（カッコ）の中に記述した `'color'` が CSS のプロパティ、`'#ebc000'` がそのプロパティの値に相当し、「**CSS の color プロパティに #ebc000 を指定せよ**」という命令であることがわかります。

この `css()` という命令は、JavaScript に定義されている命令ではなく、jQuery 独自の命令です。jQuery の js ファイルの中に `css()` という命令が定義されており、その命令を `main.js` から利用しています。このような jQuery 独自の命令のことを「**jQuery メソッド**」と呼びます（以降は「メソッド」と記述します）。

また、`css()` の「`()`」の中に記述した CSS プロパティや値のように、メソッドに引き渡されるデータを「**引数**」と呼びます[*7]。

まずは、上記のコードが **jQuery のメソッド**（`css()`）と **２つの引数**（`'color', '#222'`）で構成されているということを把握しておいてください。

Fig スタイルを変更するコードの内容

```
.css('color', '#222');
 ↑       ↑
jQueryメソッド　2つの引数
```

命令の対象：$() 関数と jQuery オブジェクト

さてここで `css()` というメソッドの「**命令の対象**」になっているのは「**id="typo" が指定されている HTML 要素**」ですが、jQuery では HTML 要素に対して直接 `css()` メソッドを実行することはできません。メソッドを使用するためには、jQuery 独自の形式で命令の対象を指定することが必要です。それが以下の部分になります。

JavaScript 命令の対象部分　main.js

```
$('#typo')
```

＊7 メソッドに引数を指定することを「**引数を渡す**」といい、メソッドを実行することを「**メソッドを呼ぶ（呼び出す）**」といいます。

`$()` と CSS のセレクタ（上記では `'#typo'`）の関係は、先述した `css()` メソッドと引数（CSS プロパティおよび値）の関係と同じです。セレクタは `$()` の引数として渡されています。

Fig 命令対象の内容

```
$('#typo');
```

関数　1つの引数

`$()` は「**HTML 要素をもとにして jQuery 独自の要素を生成する**」という機能を持っている**関数**です。HTML 要素を指定する方法として、CSS のセレクタを指定できます。この `$()` 関数によって jQuery の形式に変換された要素のことを「**jQuery オブジェクト**」と呼びます。

Point! メソッドと関数

`$()` と `css()` は、引数を伴うという意味では類似していますが、`$()` はメソッドとはいわず、「**関数**」といいます。この辺りが少しややこしいのですが、現時点では、**命令文の前に「.」（ドット）がついているものは「メソッド」、ついていないものは「関数」**だと理解しておいてください。

今回使用したコードを用いて解説すると、「`$('#typo')`」の指定によって `index.html` 内の全 HTML のうち、以下の赤字の箇所の要素のみが jQuery オブジェクトに変換されます❶。このことから「`$('#typo')`」に続いて指定されているメソッドは、この jQuery オブジェクトに対して実行されると理解できます。

html jQuery オブジェクトとして変換される要素（コード内の赤字の箇所） index.html

```
//省略
<body>

<header class="page-header" role="banner">
    <h1>Creative jQuery Sample</h1>
</header>

<div class="page-main" role="main">
    <div id="typo">
        <div class="inner">Creative jQuery</div>
    </div>
</div>

<footer class="page-footer" role="contentinfo">
    <small class="copyright">COPYRIGHT &copy;
        <a href="http://www.shiftbrain.co.jp" target="_blank">
            SHIFTBRAIN Inc.</a>
    </small>
```

❶


```
</footer>

</body>
//省略
```

他の例を挙げると、例えば「`$('h1')`」を指定した場合は、以下の要素が jQuery オブジェクトに変換され、これが命令の対象になります❷。

html jQuery オブジェクトとして変換される要素（コード内の赤字の箇所） index.html

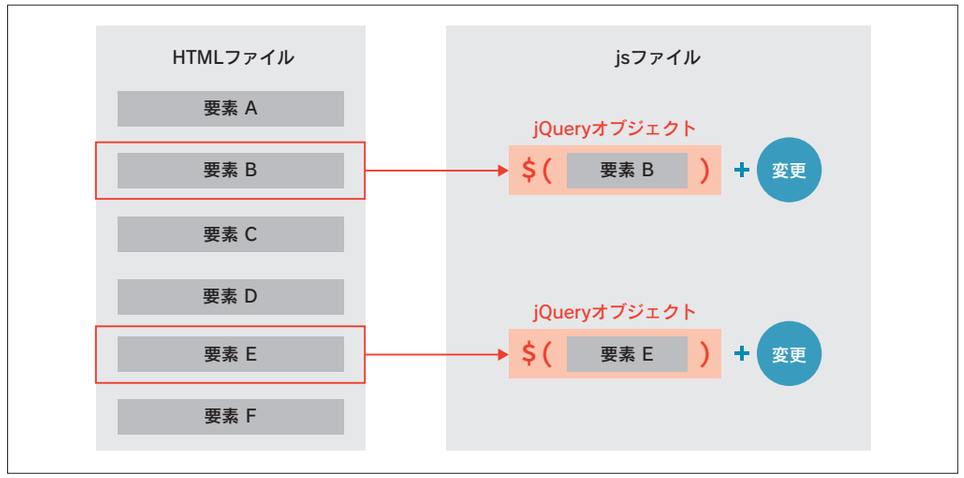
```
//省略
<body>

<header class="page-header" role="banner">
    <h1>Creative jQuery Sample</h1>
</header>

<div class="page-main" role="main">
//省略
```

❷

Fig HTML 要素と jQuery オブジェクトの関係

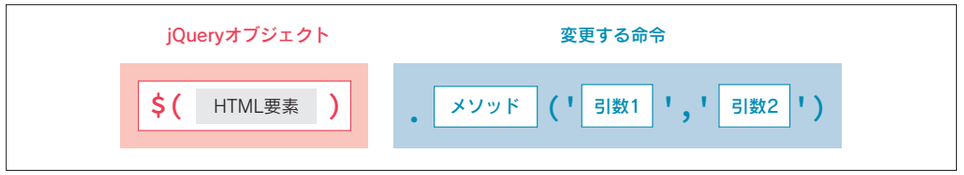


また、処理の流れも基本的に以下の通りです。

（1）`$()` 関数で命令の対象となる HTML 要素を jQuery オブジェクトに変換し、
（2）その jQuery オブジェクトに対してメソッドを呼び出して変更を加える

jQuery には、`css()` メソッド以外にも、たくさんのメソッドが用意されていますが、どのメソッドの場合でもこの流れは変わりません。これが jQuery の基本です。ここで登場した各用語の関係は次の図のように表されます。これらが jQuery において最も重要な登場人物になりますので、しっかりと関係性を把握しておいてください。

Fig 主な登場人物の関係図



「.」(ドット) と「;」(セミコロン) の役割

今回解説に使用しているコードでは、**$()** 関数 (jQuery オブジェクト) に続けて **css()** メソッドが指定されていました。この関数とメソッドの間にある「**.**」(ドット) には **jQuery オブジェクトと jQuery メソッドをつなぐ役割**があります。

また、行末の「**;**」(セミコロン) は、JavaScript の「文」の終わりを表す記号です。日本語の文における「。」にあたるものです。いくつかの例外もあるのですが、現時点では **JavaScript の 1 つの命令が終わったら「;」をつける**と覚えておいてください。

Fig 「.」(ドット) と「;」(セミコロン) の役割

HTML の準備が整うのを待つ仕組み

さて、後に回していたサンプルコードの 1 つめのパートについて解説します。

JavaScript 1 つめのパート — main.js

```
$(function(){
    //2つめのパートが記述されている
});
```

このコードの役割を確認するために、1 つめのパートだけを削除した状態でブラウザをリロードしてみてください。すると**スクリプトが動作していない**ことが確認できるはずです。なぜこうなってしまうのでしょうか。これには HTML の構造が関係しています。

もう一度サンプルの **index.html** のコードを見てみましょう。

html サンプルの HTML — index.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<title>Creative jQuery</title>
<link rel="stylesheet" href="./css/normalize.css">
<link rel="stylesheet" href="./css/main.css">

<script src="./js/vendor/jquery-1.10.2.min.js"></script>
<script src="./js/vendor/jquery-ui-1.10.3.custom.min.js"></script>
<script src="./js/main.js"></script> ❶
</head>
<body>

    //省略

</body>
</html>
```

ブラウザは基本的に、HTML を「**コードが記述された順番通り**」に処理していきます。つまりブラウザは、1 行目にある **<!DOCTYPE html>** から順に、**html** 要素、**head** 要素、**<meta charset="UTF-8">**…という順で認識していきます。

そして、このブラウザの処理が **<script src="./js/main.js"></script>** の部分に到達すると、ブラウザは **main.js** を読み込みます❶。そして、**main.js** の読み込みが完了するとその内容をすぐに実行します。

この際、**main.js** にはその後に続く HTML の要素を操作する命令が書かれています。しかしブラウザはこの時点ではまだ **head** 要素の途中までしか認識していないため (それ以降の要素はまだ認識していないため)、処理を実行することができません。

この問題を解決するための仕組みが、1 つめのパートにあります。以下のように **$(function(){ });** を記述すると、ブラウザはその中に記述された処理をいったん**予約状態**にしておき、HTML を最後まで読み込んだ時点でそれらを実行するようになります。

JavaScript ブラウザが HTML を最後まで読み込むのを待つための仕組み

```
$(function(){
    //HTMLがロードした後に実行する処理
});
```

jQuery を利用したプログラムでは、最初に HTML の要素を選択することがとても多いため、ほぼすべてのコードが上記のような構成になります。本書でもすべてのサンプルにこのコードが記述されています。

ここまでのまとめ

ここでいったんこれまでの解説を振り返ってみます。これまでの解説で以下のことを学びました。これらの内容はすべて jQuery の最も基本的な項目であり、いずれもとても重要です。現時点では概要的な内容しか解説していない箇所もありますが、しっかりと理解してから次に進んでください。

- jQuery で CSS のスタイルを変更する方法
- 数値と文字列の違い
- jQuery メソッドや $() 関数、jQuery オブジェクトの概要と関係性
- 「.」(ドット) と「;」(セミコロン) の役割
- HTML の準備が整うのを待つ仕組み

これらについてきちんと理解できているか振り返ってみてください。なお、次項の解説も上記と同様に以下の流れで学習していきます。

(1) main.js にサンプルコードを記述する
(2) ブラウザで index.html の表示内容を確認する
(3) 記述したサンプルコードの解説
(4) コードを自由に書き替えてみる
(5) ブラウザで index.html の表示内容を確認する
(6) サンプルコードを使用した jQuery の基本解説