

Webデザイン実習3B

2019/6/03

Kazuma Sekiguchi

class@cieds.jp

jQuery

- 最近のウェブ上での表現はDOMを利用して見た目を変化させるのが多い
 - CSSでできるだけ表現を行う
 - アニメーションもCSSTransitionなどを利用する
- JSをネイティブで記述するのが一番良いが、記述量が多い
 - できるだけ記述量を減らしたりDOMへのアクセスを簡単にできた方が楽
 - 良く使用するような機能を予め用意しておく
- 代表的なJSのライブラリがjQuery
 - ライブラリ＝誰かが作成したある機能を利用するために簡単に利用できるように機能をまとめたもの

jQuery

- 表現はDOMを通じたCSSの改変などが多い
 - バックエンドシステムと連動してユーザビリティを向上する目的でJSが利用されるのが最近
 - SPA (SinglePageApplication) など
 - GoogleMap、Gmail、Twitter、Facebookなどのようにリロード無しで機能が利用できるようなもの
 - ほとんどがJSを上手く利用することで実現している
- SPAを実現するライブラリー群とjQueryはイマイチ相性が良くない
 - JSの進化と共にjQueryの機能に近いものがJS自体に搭載され始めている

jQuery

- jQuery不要論の登場
 - ”*You Don't need jQuery*”
<https://github.com/oneuijs/You-Dont-Need-jQuery>
 - jQueryを使うと速度が落ちる（事実）
 - 返って開発効率が落ちる、代替できる手段が提供されてきている
 - jQueryを使わないで、表現を実現したりする
- 現在のウェブは速度重視に移行しつつある
 - 1秒でも早く描画される、利用できる、ユーザビリティが向上できる
というところに力点
 - コーダー（フロントエンドエンジニア）に求められる能力
- できればjQueryに頼らないで記述できるとなお良し

jQuery

- jQueryは比較的枯れた技術
 - 最先端では無いが、未だに利用されるケースは多い
 - ES6, Angular4, React, Vue.jsとかが先端かな
 - 利用される = それなりに便利。恐らく消えて無くなる可能性は低い
 - jQueryベースで書いたものをJSネイティブに移行するのはさほど難しくは無い
- 現在も開発は続いている
- プラグインなどの豊富な資源
 - jQueryをベースにさまざまなプラグインが登場
 - 例：モーダルウィンドウ、スライドショー、パララックス表現などなど
 - 現在もある程度プラグインは利用できる

なぜjQueryを使うか

- jQueryはJavaScriptを簡単に記述するためのもの
 - jQueryを使わなくてもプログラムを記述することは可能
 - jQueryを使うと面倒なところを簡略化できる

文字の色を変えるだけのプログラム

JavaScript

```
document.getElementById("text").style.color="#F90";
```

jQuery

```
$("#text").css("color","#F90");
```

アニメーション

JavaScript

```
var begin = new Date() - 0;
var element = document.getElementById("box");
var id = setInterval(function(){
    var current = new Date() - begin;
    if (current > 1000){
        clearInterval(id);
        current = 1000;
    }
    element.style.top = current / 10 + 'px';
}, 10);
```

jQuery

```
$("#box").animate({"top": "50px"}, 1000);
```

アニメーションみたいな処理

- インタラクティブなコンテンツには何らかのアニメーション的な動きを付けるのが一般的
 - わかりやすさ
 - インパクト
- JavaScript自体にはアニメーションするための仕組みは無い
 - 必要な回数だけ同じ処理を短時間で繰り返してアニメーションを実現する
 - パラパラ漫画と同じ要領
 - 汎用的な処理であるはずなのに、コードにすると長くなってしまおう

アニメーションみたいな処理

- jQueryにはanimateというアニメーションするための仕組みが備わっている
 - 最終的に実現したい状態のCSSを記述すれば、それになるように徐々に変化させていく
 - 一部対応していないCSSプロパティもある（色に関するもの）
- 同じ処理を繰り返すように記述しなくて良いため、調整が楽
 - 他のことに時間を割くことができる
- 最近はCSSTransitionやCSSAnimationを利用するのが一般的で、Class属性をjQueryから操作し、タイミングに合わせてスタートしたり止めたりする
 - CSSの方が早く、スムーズに動作するため

jQuery

- `$(操作する対象).操作(引数);`
 - 基本的な文法としてはこれだけ
 - ドットで連結していく場合もある
- 操作する対象には、CSSのセレクタと同じ記述でOK

`$("#text")` → id="text" に対して何か行う

`$(".photo")` → class="photo" に対して何か行う

`$("a")` → aタグ に対して何か行う

`$("#contents li")` → id="contents" 内にあるliタグ に対して何か行う

jQuery

- いきなり“\$.” という場合もある
 - \$.get();、\$.ajax();など
- 「\$」 = 「jquery」
- 操作部分にはjQueryで決められているメソッドを利用する
 - 数十種類存在
 - 組み合わせて利用することも可能

メソッド (1)

- `.addClass()` DOM要素にクラスを追加する
- `.attr()` DOM要素の属性の読み書きをおこなう
- `.html()` 要素内に含まれるHTMLの取得、設定をおこなう
- `.removeAttr()` 要素の属性を削除する。
- `.val()` value属性の値を取得したり設定したり。
- `.after()` 指定要素の後ろに要素を追加する
- `.append()` DOMツリーに要素を追加
- `.appendTo()` DOMツリーに要素を追加
- `.attr()` DOM要素の静的な属性を取得する
- `.before(content [, content])` 選択されているDOM要素の前にcontentを追加
- `.css()` DOM要素にCSSを設定する
- `.detach([selector])` DOMツリーからselectorを削除する。イベントハンドラやデータは失われない。
- `.empty()` DOMツリーからも実際のデータも削除。
- `.hasClass(className)` DOM要素がclassNameというクラス属性を持っていればtrue
- `.height()` DOM要素の高さを取得したり設定したり。
- `.html()` jQueryオブジェクトが保持している最初のDOM要素に含まれるHTMLを返す。
- `.innerHeight()` DOM要素の高さを取得する。paddingは含むがborderは含まれない。

メソッド (2)

- `.innerWidth()` DOM要素の幅を取得する。paddingは含むがborderは含まれない。
- `.offset()` jQueryオブジェクトが持つ最初のDOM要素(indexが0のもの)のDocumentからの位置を取得する。
- `.prepend()` 選択要素の前に要素を追加する
- `.prependTo(target)` 選択要素を target の前に挿入する
- `.remove([selector])` DOMツリーから selectorを削除する。イベントハンドラやデータも削除されるがDOMそのものは残る。
- `.replaceAll(target)` target を jQueryオブジェクトにセットされている要素と置換する
- `.scrollLeft()` 右にどのくらいスクロールされたか、またはするか
- `.scrollTop()` 下にどのくらいスクロールされたか、またはするか
- `.text()` jQueryオブジェクトにセットされてる要素のテキストデータをセット/ゲットする。
- `.width()` DOM要素の幅を取得したり設定したり。

良く使うメソッド

指定	働き
<code>\$(“ターゲット”).css({“プロパティ名”:“値”})</code>	指定した要素にCSSを適用
<code>\$(“ターゲット”).size()</code>	指定した要素の数を取得
<code>\$(“ターゲット”).animate({“プロパティ名”:“値”},時間)</code>	指定した要素をアニメーションしながらCSSを適用
<code>\$(“ターゲット”).width()</code>	指定した要素の幅を取得
<code>\$(“ターゲット”).height()</code>	指定した要素の高さを取得
<code>\$(“ターゲット”).text()</code>	指定した要素の文字列を取得
<code>\$(“ターゲット”).text(“追加”)</code>	指定した要素のテキストを「追加」に変更する
<code>\$(“ターゲット”).html()</code>	指定した要素のHTMLを取得(タグ付きで取得する)
<code>\$(“ターゲット”).html(“<p>追加</p>”)</code>	指定した要素をHTMLとして指定したテキストに変更する

良く使うメソッド

指定	働き
<code>\$("#ターゲット").remove()</code>	指定した要素を削除する
<code>\$("#ターゲット").detach()</code>	指定した要素を削除する(再利用する場合。通常変数に返り値を入れておく)
<code>\$("#ターゲット").prepend("<p>追加</p>")</code>	指定した要素の直下(直ぐ後ろ)に指定したHTMLを追加する
<code>\$("#ターゲット").append("<p>追加</p>")</code>	指定した要素の終了タグの直前に指定したHTMLを追加する

何かする（イベント）

```
$(“対象”).on(“click”,function(){  
    することを指定  
});
```

クリックしたら何かする例

```
$(“a”).on(“click”,function(){  
    $(“p”).css(“color”:”#F00”);  
});
```

aタグをクリックしたら、pタグの文字色を赤色に変更

```
$(“対象”).on(“resize”,function(){  
    することを指定  
});
```

ブラウザサイズを変えたら何かする例

```
$(window).on(“resize”,function(){  
    $(“div”).css(“top”:”300px”);  
});
```

ブラウザサイズを変えたら、divタグの位置を変更

JSとjQuery

- jQuery自体もJSで書かれているため、JSの構文はそのまま利用可能
 - 変数、制御文などもそのまま利用可能
 - DOMの操作も可能だが、DOMの操作はjQueryの機能を利用した方が便利（そのためにjQueryを利用するので）

jQueryのバージョン

jQuery1.x

- IE8以下をサポートしているjQuery
- 古いプラグインやブラウザで動作させる場合はこれを利用

jQuery2.x

- IE9以上のみサポートしているjQuery
- jQuery1.xよりも高速とされる

jQuery3.x

- jQuery1.x,2.xの後継
- いくつかの機能が削除され、いくつかの機能が変更されている
- 現行開発バージョン

- 基本的には3.x系列を利用するのがベスト
 - ほかは開発が止まっている（セキュリティフィックスのみ）
 - 古いバージョンを利用する場合は、その中での最新版のみを利用すること（古いとセキュリティ上問題がある）

jQueryの利用

- <script>タグでjQueryを読み込んで利用
- 1つ読み込めば充分
 - まれにいくつもjQueryを読み込む人がいますが、単に表示が遅くなるし、表示がおかしくなるなど全くメリット無い
 - プラグインなどをコピペで利用すると2つとか読み込むケースが多いので、きちんと理解して利用すること（学生さんにめちゃくちゃ多いです）
 - コーダー（フロントエンド）重視の人が見るとかな〜〜りマズイ印象を抱く

ダウンロード

- <https://jquery.com>

