ソフトウェア開発

– 授業配布版 –

平野 照比古

(平成 30 年 10 月 18 日改訂)

目 次

第1回　ガイダンス

1.1　受講に関する注意

この授業に関する注意は次の通りである。

- JavaScript を通常の計算機言語として利用するための解説を 10 回の講義で行う。
- 進度が今までのプログラミングの授業より早いので復習をよくすること。
- パソコンを授業に持参す必要がある場合はその旨、前回の授業で指示する。
- 配布された復習用の課題は用紙に解答を直接記入するか、印刷したものを添付して次回の授業の前日の木曜日 10 時までに 6 階の平野のレポートボックスに提出する。
- 復習用課題と同時に配布されるルーブリック評価表も自己評価を付けて提出のこと。
- レポートの内容はルーブリック評価表の「標準的」の項目を参照すること。
- 復習課題の提出方法はメール提出に変更する場合もある。
- 復習課題とそのルーブリック評価表は次回の授業の開始時に添削、評価をしたものを返却する。
- 復習用課題は 10 回程度を予定している。成績評価の 60%を占める。
- 最終回の授業に試験を行う。成績評価の 40%を占める。
- 配布資料等は `http://www.hilano.org/hilano-lab` で公開する予定である。

1.2　参考図書

1. E. Brown, 初めての JavaScript 第 3 版, ES2015 以降の最新ウェブ開発, オライリージャパン
2. David Flanagan, JavaScript 第 6 版, オライリージャパン
3. David Flanagan, JavaScript クイックリファレンス第 6 版, オ ライリージャパン
4. Nicholas C. Zakas, ハイパフォーマンス JavaScript, オライリージャパン
5. Nicholas C. Zakas, メンテナブル JavaScript 読みやすく保守しやすい JavaScript コードの作法, オライリージャパン

1.3　シラバス

この授業のシラバスは次のように予定している。

授業回数	内容
第 1 回	授業のガイダンスとブラウザの開発者モードについて JavaScript の実行環境の確認
第 2 回	JavaScript が取り扱うデータ データの型と演算子に関する注意など
第 3 回	関数の定義方法と変数のスコープ
第 4 回	オブジェクトの定義とクラス
第 5 回	オブジェクトの詳細 関数によるオブジェクトの定義、エラー処理
第 6 回	正規表現と文字列の処理
第 7 回	DOM の利用 HTML 文書、CSS と DOM の基礎
第 8 回	イベント処理 イベントモデルとイベント処理の例
第 9 回	PHP 超入門 PHP に関する簡単なプログラム
第 10 回	サーバーとのデータの交換 PHP 入門の続きとサーバーとのデータ交換の基礎
第 11 回	jQuery DOM の処理を簡単にするライブラリーの紹介
第 12 回	XML ファイルの処理 JavaScript と PHP による XML ファイルの処理の例
第 13 回	クラスの例
第 14 回	システム開発のヒント
第 15 回	最終試験と解説

1.4　JavaScript の実行環境

JavaScript は HTML 文書内で使われる場合が多いが、通常の計算機言語として取り扱うこともできる。この授業では JavaScript の計算機言語としての特徴について述べた後、Web ブラウザ内での処理について解説を行う。しかし、JavaScript を単独で実行する環境はほとんどなく[1]、このテキストではブラウザ内で実行する例だけを取り上げる。

最近のブラウザは JavaScript の実行環境を提供するだけではなく、開発環境も提供している。Google Chrome の「デベロッパーツール」、FireFox の「Web 開発」、Internet Explorer の「開発者ツール」などがそうである。

[1] コマンドラインから JavaScript を実行する環境としては Node.js があるが、新規にインストールする必要がある。

これらのツールは「F12」または「Control+Shift+I」のショートカットキーで表示、非表示ができる。表示されるタブには次のようなものがある[2]。

- **Elements** HTML の構造 (DOM の構造) を表示する。対応する要素の部分が反転する。
- **Console** エラーや警告が表示される。また、JavaScript のプログラムをその場で実行できる。
- **Sources** HTML 文書のソースが表示される。JavaScript のソースの場合にはプログラムの実行を中断するブレイクポイントの設定が可能である。
- **Network** ファイルの取得などの順序やかかった時間などが表示される。
- **Timeline** ブラウザの処理に関する情報が表示される。
- **Profiles** JavaSCript の関数で使われた実行時間などが記録できる。

課題 1.1 各自が使用しているブラウザにおいて JavaScript の文が直接実行できるコンソールが開けること、コンソールで「`1+2`」と入力すると、計算結果が表示されることを確認すること。

1.5 strict モードについて

ECMAScript の改訂版 ECMAScript 2015[3]では `strict` モードという厳密な解釈をするモードが導入された。このモードでは従来見つけにくい単純なバグがエラーとなる。主な違いは次のとおりである。

	非 `strict` モード	`strict` モード
変数の宣言	必要ではない	必要
書き込み不可なプロパティへの代入	エラーが発生しない	エラーが発生
関数の `arguments` オブジェクトの値の変更	可能	不可能
関数の `arguments.caller`	参照可能	エラーが発生
関数の `arguments.callee`	参照可能	エラーが発生
8 進リテラル (`0` で始まる数)	使用可能	エラーが発生

プログラムを `strict` モードにするためには先頭に`"use strict;"` を記述する。また、関数の定義の直後に`"use strict";` を記述するとその関数内は `strict` モードになる。

課題 1.2 次のコードを拡張子が `html` のファイルで作成する。それをブラウザで開き、デベロッパーツールのコンソールから `foo();` と入力したときと、その後、コンソールで `i` と入力したときの結果を確認すること。同様のことを 9 行目の `i` の宣言を省いて行うこと

また、`strict` モードに変更し (7 行目の後に「`"use strict";`」を追加する)、9 行目の変数の宣言の行を取り除くとエラーが発生することを確認すること。

[2] この例は Chrome のものの一部である。
[3] ES6 とも呼ばれる。今後、EcmaScript のバージョンは規格が制定された年号を用いることとなった。

```
 1 <!DOCTYPE HTML>
 2 <html>
 3 <head>
 4 <meta charset="UTF-8"/>
 5 <title>初めての JavaScript</title>
 6 <script type="text/ecmascript">
 7 //<![CDATA[
 8 function foo(){
 9   let i;
10   for(i=1;i<10;i++) {
11     console.log(`${i} ${i*i}`);
12   }
13 }
14 alert("デベロッパーツールからコンソールを開いてコンソールから foo(); と入力してください。");
15 //]]>
16 </script>
17 </head>
18 </html>
```

コードの簡単な説明

- 1 行目 HTML5 における HTML 文書の宣言
- 4 行目 このファイルの文字コード (エンコーディング) を UTF-8 に指定
- 6 行目 スクリプトの開始の要素。プログラミング言語が ECMAScript であることを宣言
- 7 行目 `//`は行末までの部分をコメントにする JavaScript の記法。残りの部分はこれ以降 12 行目までは通常の文字として解釈することを指定 (`CDATA` セクションの開始)。

  7 行目と 13 行目を消去したらどうなるのか確認することまたその理由も考えること[4]。
- 8 行目は関数 `foo()` の宣言。13 行目までがこの関数の定義範囲
- 9 行目は変数 `i` の宣言 (従来の JavaScript では `var` を用いて宣言していたが、この授業では ECMAScript 2015 で新しく導入された `let` だけを用いる。)
- 10 行目 C 言語などでおなじみの繰り返しの指定
- 11 行目 引数内の式をコンソールに出力。ここでは ECMAScript 2015 で導入された文字列の中に式を埋め込み可能なテンプレートリテラル形式 (バッククオート (`@`の上にある文字) で挟む) で記述。
- 14 行目はメッセージボックスに「コンソールを開く」ことを指示。

開発者ツールを開き、コンソールから `foo();` を入力する。

[4]最近のブラウザではエラーが起きないかもしれない。

第2回　JavaScriptのデータとプログラミングの基礎

2.1　データーの型

JavaScript の方には大きく分けてプリミティブ型と非プリミティブ型の 2 種類がある。変数や値の型を知りたいときは `typeof` 演算子を使う。

2.1.1　プリミティブデータ型

表 2.1は JavaScript におけるプリミティブデータ型の一覧である。

表 2.1: プリミティブデータ型

型	説明
`Number`	64 ビット浮動小数点形式だけ
`String`	文字列型、文字のデータ型はない。ダブルクオート (") かシングルクオート (') で囲む。ECMAScript 2015 では文字列内に式の値を埋め込み可能なテンプレートリテラル形式 (バッククオート ‘ ではさむ[1]) が追加された。
`Boolean`	`true` か `false` の値のみ
`undefined`	変数の値が定義されていないことを示す
`null`	`null` という値しか取ることができない特別なオブジェクト

Number 型　数を表現する方法 (数値リテラル) としては次のものがある。

- **整数リテラル** 10 進整数は通常通りの形式である。16 進数を表す場合は先頭に `0x` か `0X` をつける。先頭が `0` でそのあとに `x` または `X` が来ない場合は 8 進数と解釈される場合がある。`strict` モードではこの形式はエラーとなる。

  2 進数は `0b` で、8 進数は `0o` で始まるリテラルが定義されている。

- **浮動小数点リテラル** 整数部、そのあとに必要ならば小数点、小数部そのあとに指数部がある形式である。

Number 型には次のような特別な Number が定義されている。

[1] キーボードの`@`の上にある文字

- `Infinity` 無限大を表す読み出し可能な変数である。オーバーフローした場合や `1/0` など演算の結果としてこの値が設定される。
- `NaN` Not a Number の略である。式が計算ができなかった場合の結果を表す読み出し可能な変数である。文字列を数値に変換できない場合や `0/0` などの結果としてこの値が設定される。

なお、計算の途中結果がこれらの値になってもプログラムの実行は中断されない (問題 2.4を参照)。

String 型　文字列はダブルクオート (") または シングルクオート (') ではさむ。1 文字だけの文字の型はない。テンプレートリテラルは文字列の中に式の値を埋め込むことができる。テンプレートリテラルはバッククオート (‘) ではさむ。埋め込む式は`${2+3}` のように$ の後に { と } の間に式を記述する。たとえば‘`2+3=${2+3}`‘ は `"2+3=5"` という文字列になる。

2 つの文字列をつなげる連接演算子+ついては 2.4節で説明する。

表 2.2は文字列に関する情報 (プロパティ) や操作 (メソッド) の一部である。この表にある正規表現については第 6 回で解説をする。

表 2.2: 文字列のメソッドとプロパティ

メンバー	説明
`length`	文字列の長さ
`indexOf(needle,start)`	文字列内に `needle` が初めて現れる位置を返す。`start` の引数があれば、その位置以降から調べる。見つからない場合は −1 を返す。
`lastIndexOf(needle,start)`	文字列内に `needle` が一番最後に現れる位置を返す。`start` の引数があれば、その位置以前から調べる。見つからない場合は −1 を返す。
`split(separator,limit)`	`separator` の文字列 (または正規表現) で文字列を分割し、配列で返す。`separator` の部分は返されない。2 番目の引数はオプションで、分割する最大数を返す。
`substring(start,end)`	文字列の `start` から `end` の前の位置までの部分文字列を返す。
`slice(start,end)`	文字列の `start` から `end` の前の位置までの部分文字列を返す。値が負のときは文字列末尾から数えた位置を表す。

課題 2.1 次の実行結果を確かめなさい。

1. `"0123456789".indexOf("1");`
2. `"0123456789".indexOf("a");`
3. `"0123456789".indexOf("1",2);`
4. `"0,1,2,3".split(",");`
5. `"0,1,2,3".split(",",2);`
6. `"0123".split("");`

7. `"0123456789".substring(3);`
8. `"0123456789".substring(-3);`
9. `"0123456789".substring(3,5);`
10. `"0123456789".slice(-3);`
11. `"0123456789".slice(3,5);`
12. `"0123456789".slice(3,-3);`

Bool 型 `true` と `false` の 2 つの値をとる。この 2 つは予約語である。論理式の結果としてこれらの値が設定されたり、論理値が必要なところでこれらの値に設定される。詳しくは 2.4を参照のこと。

`undefined` と `null` `undefined` は値が存在しないことを示す読み出し可能な変数である。変数が宣言されたのに値が設定されていないときの初期値や、戻り値が定義されていない関数の戻り値はこの値になる。

`typeof null` の値が `"object"`であることを示すように、オブジェクトが存在しないことを示す特別なオブジェクト値（であると同時にオブジェクトでもある）である。

2.1.2 非プリミティブ型

プリミティブ型以外のデータ型は「オブジェクト」である。JavaScript では次のようなオブジェクトが定義されている。() 内はオブジェクト (を作成するコンストラクタ) の名称である。

- 配列 (`Array`)
- 数学 (`Math`)
- 日時 (`Date`)
- 正規表現 (`RegExp`)
- マップ (`Map`) とウィークマップ (`WeakMap`)
- セット (`Set`) とウィークセット (`WeakSet`)

これらのデータ型のうち、マップ、セット、ウィークセット以外は後の章で解説する。

2.2 変数

JavaScript の変数の特徴は次の通りである。

- 変数名はアルファベットまたは_(アンダースコア) で始まる英数字または_ が続く文字列
- 大文字と小文字は区別される。

- 変数の宣言は ECMAScript 2015 で採用された `let` で行う[2]。宣言時に初期化ができる。
- ECMAScript 2015 で導入された `const` で変数宣言を行うと変数の値は変更できない。
- 非 `strict` モードでは変数の宣言はしなくてもよい。初期化していない変数を演算の対象とするとエラーが起こる。詳しくは後述する。
- 変数に保存するデータの型には制限がない。途中で変更することもできる。

2.3　配列

2.3.1　配列の宣言と初期化

配列を使うためには、変数を配列で初期化する必要がある。変数の宣言と同時に行ってもよい。

```
let a = [];       //空の配列の定義
let b = [1,2,3]; //要素が 3 つある配列
```

`a` は空の配列で、`b` は `b[0]=1,b[1]=2,b[2]=3` となる。次のことに注意する必要がある。

- 実行時に配列の大きさを自由に変えられる。
- 配列の各要素のデータの型は同じでなくてもよい。たとえば配列の要素の一部に配列を置くことができる。

  ```
  let a=[1,[2,3,4],"a"];
  ```

- 配列の変数の代入は参照である。

  ```
  >let a = [0,1,2,3,4];
  >let b = a;  //別の変数への代入
  >b;
  (5) [0, 1, 2, 3, 4]  // a と同じ内容が表示
  >b[3]= b[3]*10;       // 4 番目の要素を 10 倍 [0, 1, 2, 30, 4] となる
  30
  >a;
  (5) [0, 1, 2, 30, 4] // a も b と同じ値の要素を持つ
  ```

  配列の代入が参照なので片方の要素を変化させると、両方とも変わることが分かる。

- 配列の要素の一部をまとめて別の変数に代入する分割代入が ECMAScript 2015 で導入された。分割代入では代入演算子=の左辺を配列の形で記述する。

  代入 `[a,,b,...c] = [0,1,2,3,4,5,6,7];` の結果は次のとおりである。

[2]従来の宣言方法の `var` にはいくつかの問題があるのでこの授業では使用しない。

```
a ==> 0
b ==> 2
c ==> [3, 4, 5, 6, 7]
```

- 変数 a には 0 番目の、b には 3 番目の、c には 4 番目以降の配列の要素が代入される。
- a と b の間に,, があるので 2 番目の要素は代入されない。
- 4 番目以降の要素はまとめて変数 c に代入されている。

配列が関数の戻り値のとき、戻り値の必要なところだけ利用するために便利な機能である。

なお、ここで使用している... は展開演算子と呼ばれ、配列に対して要素を並べたものを表す。

課題 2.2 次の実行結果を確かめなさい。

1. `[,[,a]] = [1,[2,3,4],5]; console.log(a);`
2. `a=10;b=20;[b,a]=[a,b];console.log(a);console.log(b)`

2.3.2 配列のメソッド

表 2.3は配列のメソッドやプロパティをまとめたものである。

表 2.3: 配列のメソッドとプロパティ

メンバー	説明
length	配列の要素の数
join(separator)	配列を文字列に変換する。separator はオプションの引数で、省略された場合はカンマ (,) である。
pop()	配列の最後の要素を削除し、その値を返す。
push(i1,i2,...)	引数で渡された要素を配列の最後に付け加える。
shift()	配列の最初の要素を削除し、その値を返す。
unshift()	配列の最初の要素を削除し、その値を返す。
slice(start,end)	start から end の前の位置にある要素までからなる配列を返す。
splice(start,No,i1,i2,...)	start の位置から No 個の要素からなる配列を返す。元の配列からこれらの要素を取り除き、その位置に i1,i1,... 以下の要素を付け加える。
indexOf(value,start)	配列の要素で value に等しい (===) ものを探し、その位置を返す。見つからない場合は-1 を返す。オプションの引数 start はその位置以降から探すことを指定する。
lastIndexOf(value,start)	配列の要素で value に等しいものを後ろから探し、その位置を返す。見つからない場合は-1 を返す。オプションの引数 start はその位置以前から探すことを指定する。

配列のメソッドとしてはこのほかに、処理をする関数を引数として与えるものがある。それらについては関数のところで解説をする。

課題 2.3 次の実行結果を確かめなさい。なお、2 以降をコンソールで連続して行う場合には 3 以降にある `let` はつけないこと。

1. `[1,2,[],3].length;`
2. `let a=[1,2,3]; console.log(a.pop()); console.log(a.length);a;`
3. `let a=[1,2,3]; a.push(4,5); console.log(a.length);a;`
4. `let a=[1,2,3]; a.shift(4,5); console.log(a.length);a;`
5. `let a=[1,2,3]; a.join(" ");`
6. `let a=[1,2,3,4,5]; console.log(a.slice(1,2)); console.log(a.length);a;`
7. `let a=[1,2,3,4,5]; console.log(a.splice(1,2)); console.log(a.length);a;`
8. `let a=[1,2,3,4,5]; console.log(a.indexOf(3)); console.log(a.indexOf(3,3));`
9. `let a=[3,1,2,3,4,5]; console.log(a.lastIndexOf(3)); console.log(a.lastIndexOf(3,2));`

2.4　演算子

2.4.1　代入、四則演算

数に対しては C 言語と同様の演算子が使用できる。ただし、次のことに注意すること。

- +演算子は文字列の連接にも使用できる。+演算子は左右のオペランドが Number 型のときだけ、和をとる。一方が文字列の場合は数を文字列に直して、文字列として連接を行う。
- そのほかの演算子 (`-*/`) については文字列を数に変換してから数として計算する。
- 文字列全体が数にならない場合には変換の結果が `NaN` になる。

```
2+3  => 5
"2"+3 => "23"
"2"-0 +3 => 5
"2"*3    => 6
"2"*"3"  => 6
"f" *2   => NaN
"0xf"*2  => 30
```

- 整数を整数で割った場合、割り切れなければ小数となる。小数部分を切り捨てたいときは `Math.floor()` を用いる。

```
1/3 => 0.3333333333333333
Math.floor(1/3) =>0
```

課題 2.4 次の実行結果を確かめなさい。

```
console.log(1/0),console.log(1*"a");
```

2.4.2 論理演算子と比較演算子

論理演算子 Boolean 型に対して使用できる演算子は次の 3 つである。

- ! 論理否定
- && 論理積
- || 論理和

論理演算子を Boolean 型でない値を与えると次の値は `false` とみなされる。

- 空文字列 `""`
- `null`
- `undefined`
- 数字の 0
- 数値の `NaN`
- Boolean の `false`

論理和や論理積では左のオペランドの結果により、式の値が決まる場合は右のオペランドの評価は行われない。たとえば、論理和の場合、左の値が `true` であれば右のオペランドの評価が行われない。次の例では変数 `a` の値は 1 のままである。

```
let a=1; true ||(a=3);
```

比較演算子 比較演算子は比較の結果、Boolean の値を返す演算である。C 言語と同様の比較演算子が使用できる。`>,>=,<,<=`などがある。等しいことを比較するためには`==`(等価比較演算子) のほかに 型変換を伴わない等価比較演算子`===` がある。等価比較演算子`==`は必要に応じて型変換を行う。特別な事情がない限り、`==`や`!=`は使用しないこと。

```
0 == "0" => true
0 === "0" => false
```

同様に非等価比較演算子`!=`にも型変換を伴わない非等価演算子`!==`がある。

`NaN === NaN` の結果は `false` である。値が `NaN` であるかを調べるには関数 `isNaN()` を用いる。

2.5　制御構造

JavaScript でも C 言語などと同様に `if` 文、`for` 文、`while` 文、`switch` 文がある。使い方も同様である。また、Java や C++のように `for` 文の初期化のところで変数を `let` を用いて宣言することができる。このように宣言された変数に関する JavaScript 特有の注意は次回解説をする。

2.6　組み込みオブジェクト

2.6.1　組み込み関数

表 2.4は JavaScript に組み込まれている代表的な関数である。

表 2.4: JavaScript における組み込み関数

名称	説明
`parseInt(string,radix)`	`string`(文字列) と `radix`(基数、省略したときは 10) をとり、先頭から見て正しい整数表現のところまで整数に変換する。
`parseFloat(string)`	`string`(文字列) をとり、先頭から見て正しい浮動小数点表現のところまで浮動小数に変換する。
`isNaN(N)`	`N` が数であれば `false`、そうでなければ `true` を返す。
`isFinite(N)`	`N` が数値または数値に変換できる値でかつ `Infinity` または `-Infinity` でないときに `true`、そうでないとき、`false` を返す。
`encodeURIComponent(string)`	`string` を URI エンコードする。ブラウザの日本語を含む字句検索の結果のアドレスバーに%で始まる文字列が URI エンコードした結果である。
`decodeURIComponent(string)`	`encodeURIComponent(string)` の逆の操作をする。
`encodeURI(string)`	`string` を URI エンコーディングする。この関数はプロトコル部分などはエンコードしない。
`decodeURI(string)`	`encodeURI(string)` の逆の操作をする。

このほかにもいくつか関数があるが、非推奨であるか使い方に注意するものなので省略する。

2.6.2 Math オブジェクト

Math オブジェクトには数学的な定数の定義 (円周率など) や三角関数などの関数が定義されている。表 2.5はそれらのプロパティや関数をまとめたものである。

表 2.5: JavaScript における Math オブジェクトの種類

名称	種類	説明
Math.E	定数	自然対数の底 (2.71828182...)
Math.LN10	定数	$\log_e 10$
Math.LN2	定数	$\log_e 2$
Math.LOG2E	定数	$\log_2 e$
Math.LOG10E	定数	$\log_{10} e$
Math.PI	定数	円周率 ($\pi = 3.141592\ldots$)
Math.SQRT1_2	定数	$\frac{1}{2}$ の平方根 $\sqrt{\frac{1}{2}}$
Math.SQRT2	定数	2 の平方根 $\sqrt{2}$
Math.abs(x)	関数	x の絶対値
Math.acos(x)	関数	逆余弦関数 $\arccos x$
Math.asin(x)	関数	逆正弦関数 $\arcsin x$
Math.atan(x)	関数	逆正接関数 $\arctan x$
Math.atan2(y, x)	関数	$\arctan \frac{y}{x}$ を計算。x が 0 のときでも正しく動く
Math.ceil(x)	関数	x 以上の整数で最小な値を返す。
Math.cos(x)	関数	余弦関数 $\cos x$
Math.exp(x)	関数	指数関数 e^x
Math.floor(x)	関数	x の値を超えない整数
Math.log(x)	関数	自然対数 $\log x$
Math.max([x1, x2, ... , xN])	関数	与えられた引数のうち最大値を返す
Math.min([x1, x2, ... , xN])	関数	与えられた引数のうち最小値を返す
Math.pow(x, y)	関数	指数関数 x^y
Math.random()	関数	0 と 1 の間の擬似乱数を返す
Math.round(x)	関数	x の値を四捨五入する
Math.sin(x)	関数	正弦関数 $\sin x$
Math.sqrt(x)	関数	平方根を求める。 $\sqrt{x}$
Math.tan(x)	関数	正接関数 $\tan x$

いくつか注意をしておく。

- 配列の要素の最大値や最小値を求めるために Math.max や Math.min を直接使用できない。展開演算子... を用いると計算ができる。次の実行例を参照のこと。

```
>a = [3,5,23,1,4];
(5) [3, 5, 23, 1, 4]
>Math.max(a);
NaN
>Math.max(...a);
23
```

- EcmaScript 2016 ではべき乗の演算子**が導入されたので Math.pow() は使わなくてもよい。

詳しい解説は [1, 第 16 章] にある。

2.6.3　Date オブジェクト

Date オブジェクトは日付や時間に関するデータを扱うコンストラクタ関数である。その実体は 1970 年 1 月 1 日午前 0 時 (世界標準時) [3]からのミリ秒単位の経過時間である。

引数の数や型によりいくつかの異なる日付と日時のオブジェクトが作成できる。

- 引数なしの場合は現在の日付と日時を用いて作成する。

  ```
  >new Date()
  Thu Oct 01 2015 13:17:46 GMT+0900 (東京 (標準時))
  ```

- 引数が数値の場合は世界標準時 (UTC) において 1970 年 1 月 1 日 0 時からの経過時間をミリ秒単位で作成する。

- 引数が文字列の場合は日時を示す文字列に基づいて作成する (次の例を参照)。

- 引数の数が 2 つ以上 7 つの場合は年、月、日、時、分、秒、ミリ秒の順が指定されたとして作成する。ない引数は 0 と解釈される。

  また、月の値は 0 が 1 月を表す。

  ```
  >new Date(2015,10,2,10,10,20,500);
  Mon Nov 02 2015 10:10:20 GMT+0900 (東京 (標準時))
  ```

  上記の例では月の値 10 となっているので 11 月 (Nov) に設定されている。

  引数の値が範囲を外れる (日に対して 0 を指定するなど) 場合でも、正しく作成される。たとえば日の値が 0 のときは指定した月の 1 日の 1 日前、つまり前の月の最終日と解釈される。

[3] この時刻を UNIX エポックという。

表 2.6は Date オブジェクトのメソッドを示したものである。

表 2.6: JavaScript における Date オブジェクトのメソッド

名称	説明
getTime()	日付オブジェクトのタイムスタンプを得る
setTime(time)	日付オブジェクトのタイムスタンプを time に設定する
getFullYear()	西暦の年の値を得る。getYear() もあるが、2000 年問題の影響があるので使わないこと
setFullYear(year,month,date)	year 年 month＋ 1 月 date 日設定する。
getMonth()	月の値を得る。0 が 1 月を表すことに注意
setMonth(month,date)	month+1 月 date 日に設定する。
getDate()	日の値を得る。
setDate(date)	date 日に設定する。
getHours()	時間の設定を行う
setHours(hour,min,sec,ms)	hour 時 min 分 sec 秒 ms ミリ秒に設定する
getMinutes()	分の値を得る
setMinutes(min,sec,ms)	min 分 sec 秒 ms ミリ秒に設定する
getSeconds()	秒の値を得る
setMinutes(sec,ms)	sec 秒 ms ミリ秒に設定する
getMilliseconds()	ミリ秒の値を得る
setMilliseconds(ms)	ms ミリ秒に設定する
getTimezoneOffset()	ローカルタイムと UTC の時差を分単位で返す
getDay()	曜日 (0 が日曜日) を得る

詳しい解説は [1, 第 15 章] にある。

課題 2.5 次の日時を求める式を答えよ。与えられた日時は変数 theDay に Date オブジェクトとして与えられているものとする。

1. 与えられた日時から 1 週間後の日時
2. 与えられた日時の翌月の 1 日
3. 与えられた日時の前の月の最終日
4. 与えられた日時の月の第 1 月曜日

課題 2.6 次の式の評価結果を求めなさい。

式	結果	理由
`4+"5"`	`"45"`	右のオペランドが文字列なので左の数は文字列に変換され、それらが連接される。
`4-"5"`	-1	演算子が`-`なので右の文字列が数に変換される。
`4+"ff"`	`"4ff"`	前と同様
`4+"0xff"`	`"40xff"`	前と同様
`4-"0xff"`	-251	`"0xff"`が 255 と解釈されて計算される。
`4+parseInt("ff")`	`NaN`	文字列内に数として正しく変換されるものがないので `parseInt()` の戻り値が `NaN` となり、これ以降の数の演算は `NaN` となる。
`4+parseInt("0xff")`	259	`parseInt()` は正しく 16 進数として解釈するので $4 + 255 = 259$ となる。
`4+parseInt("ff",16)`	259	基数を 16 と指定しているので、正しく 255 と解釈される。
`4+"1e1"`	`"4+1e1"`	1 番目と同じ
`4+parseInt("1e1")`	5	”1e1” は数値リテラルとしては $1 \times 10^1 = 10$ を表すが、`parseInt()` は整数リテラル表記しか扱わないので、数の変換は `e` の前で終わる。 1 の値が戻り値となる。
`4+parseFloat("1e1")`	14	`parseInt()` と異なり、`parseFloat()` の戻り値は 10。
`"4"*"5"`	20	文字列の間では`*`の演算が定義されていないので両方とも数に変換されて計算される。
`"4"/"5"`	0.8	上と同様
`[].length`	0	配列の要素がないので長さは 0 となる。
`[[]].length`	1	長さを求める配列は空の配列一つを要素に持つ。
`0 == "0"`	`true`	文字列`"0"`が数 0 に変換されて比較される。
`0 == []`	`true`	空の配列が空文字`""`に変換されたのち、数との比較なので数 0 に変換される。
`"0" == []`	`false`	空の配列が空文字`""`に変換され、文字列の比較となる。
`![]`	`false`	空の配列はオブジェクトとして存在するので `true` と解釈され、否定演算子で `false` になる。
`false == []`	`true`	`[]` が空文字`""`に変換されたのち、`0` に変換される。
`false == undefined`	`false`	`false` が数に 0 に変換され、数と `undefined` は等しくない。
`[] == []`	`false`	配列はオブジェクトであり、二つの空の配列は別物とみなされる。
`typeof []`	`"object"`	配列はオブジェクトである。
`null == undefined`	`true`	この比較は `true` と定義されている。
`a=[], b=a, a==b;`	`true`	同じメモリーにあるオブジェクトを参照している。

第3回　関数

3.1　今回の内容

JavaScript では関数はいろいろな目的のために利用される。今回は次のことを取り扱う。

- 関数の定義方法と使用法
- 関数への引数の渡し方
- 関数の戻り値 (数や配列のほかに関数も戻り値にできる)
- JavaScript における変数のスコープ

さらに次のような JavaScript 特有の事柄についても説明する。

- 無名関数とコールバック関数
- 自己実行関数
- 関数内で定義された関数
- クロージャ

3.2　関数の定義方法と呼び出し

3.2.1　簡単な関数の例

次の例は関数 `sum()` を定義している。

```
function sum(a,b) {
  let c = a + b;
  return c;  // return a + b;  でもよい。
}
```

関数の定義は次の部分から成り立っている。

- `function` **キーワード** 戻り値の型を記す必要はない。
- **関数の名前** `function` の後にある識別名が関数の名前になる。ここでは `sum` が関数の名前である。

- **引数のリスト** 関数名の後に () 内にカンマで区切られた引数を記述する。この場合は変数 a と b が与えられている。引数はなくてもよい。
- **関数の本体であるコードブロック** {}で囲まれた部分に関数の内容を記述する。
- **return キーワード**
  関数の戻り値をこの後に記述する。戻り値を明示的に示さない場合は undefined が返される。

次にこの関数の実行例を掲げる。

```
>sum(1,2)  // 戻り値は 3
>sum(1)    // 戻り値は NaN
>sum(1,2,3)// 戻り値は 3
```

- 引数に 1 と 2 を与えられているので期待通りの結果が得られる。
- 引数に 1 だけを与えた場合、エラーが起こらず、NaN となる。これは、不足している引数 b には undefined が渡され、戻り値は 1+undefined の計算結果 (NaN) となる。
- 引数を多く渡してもエラーが発生しない。無視されるだけである。

これらのことから JavaScript の関数はオブジェクト指向で使われるポリモーフィズムをサポートしていない。さらに、次の問題で見るように同じ関数を定義してもエラーにならない。後の関数の定義が優先される。

課題 3.1 次の順に 2 つの関数 sum(a,b) と sum(a,b,c) 定義する。定義した直後に sum(1,2,3) と sum(1,2) をそれぞれ実行した結果を確認し、どちらの関数が実行されているか 確認しなさい。

```
function sum(a, b){
  return a+b;
}
function sum(a, b, c){
  return a+b+c;
}
```

3.2.2　仮引数への代入

仮引数に値を代入してもエラーとはならない。仮引数で渡された値がプリミティブなときとそうでないときとでは呼び出し元における変数の値が異なる。

実行例 3.1 次の例は呼び出した関数の中で仮引数の値を変化させたときの例である。

```
function func1(a){
  a = a*2;
  return 0;
}
```

func1() では仮引数 a の値を 2 倍している。次のように実行する。

```
>a = 4;
>func1(a); // 戻り値は 0
>a;
4          //a の値は変化しない
```

この結果から呼び出し元の変数の値には変化がないことがわかる。つまり、プリミティブデータ型の引数は値そのものが渡される (値渡し)。

次の例は配列を引数に渡す関数の例である。

```
function func2(a){
  a[0] *=2;
  return 0;
}
```

func2() の仮引数は配列が想定してあり、その配列の先頭の値だけ 2 倍する関数である。

```
>a = [1,2,3];
>func2(a);// 戻り値は 0
>a;
[2, 2, 3] //配列の先頭の値が 2 倍されている
```

この関数の実行後、配列の先頭の値が変化している。この結果、プライミティブ型以外の引数の渡し方が参照渡しであることがわかる。

3.2.3 可変引数をとる関数

前節の引数の和を計算する関数で、引数の数を固定しないものを定義するには仮引数に展開演算子をつける[1]。

実行例 3.2 次で定義された関数は任意個数の引数を取る関数の例である。

```
1 function sumN(...args) {
2     let S = 0;
3     for(let i=0;i<args.length;i++) {
4         S += args[i];
5     }
6     return S;
7 }
```

- 1 行目で関数の仮引数 args に展開演算子... をつけている。

[1] これまでの JavaScript では、関数の引数を表す配列のような性質を持つ arguments オブジェクトが用意されていたが、ECMAScript 2015 以降非推奨となった。

- args は配列となるのでその大きさは args.length でわかる。
- これを用いて総和のプログラムが 2 行目から 5 行目に記述されている。

実行例は次のとおりである。

```
>sumN(1,2,3,4);  //戻り値は 10
>sumN(1,2,3,4,5);//戻り値は 15
```

課題 3.2 実行例 3.2 で定義した関数 sumN について次の問いに答えよ。

1. sumN() の結果はなにか。
2. 配列 a の要素がすべて数値のとき、sumN() を用いて a の要素の総和を求める方法を述べよ。

3.2.4　変数のスコープ

変数のスコープとはある場所で使われている変数がどこで定義されているものかという概念である。JavaScript では次の特徴がある。

1. 非 strict モードでは変数は宣言しなくても使用できる。
2. 関数内で let により明示的に宣言された変数はブロック内で有効となる。ブロックとは{と}で囲まれた部分である。ただし、宣言よりも前では使用できない。同じブロック内で let による同一変数名の宣言はエラーになる。
3. 変数を var で宣言すると関数の途中で宣言しても、関数の先頭で宣言したと同じ効果を持つ。これを変数の巻き上げという。関数の外で var で宣言された変数は、プログラムの先頭で宣言したものとみなされる。ただし、初期化は宣言された位置で行われる。同じスコープ内で var による同一変数名の宣言はエラーにならない。
4. 関数の外で宣言された変数や宣言されないで使用された変数はすべてどこからからでも参照できるグローバルとなる。

関数やブロックの中で関数を定義したりブロックを作成すると、その内側の関数やブロック内で let で宣言された変数のほかに、一つ上の関数やブロックで利用できる (スコープにある) 変数が利用できる。これがスコープチェインである。例を挙げる。

```
let G1, G2;
function func1(a) {
  let b, c;
  function func2() [
    let G2, c;
   ...
  }
}
```

- 関数 func1() 内ではグローバル変数 G1 と G2、仮引数 a とローカル変数 b と c が利用できる。
- 関数 func2() 内ではグローバル変数 G1、func1() の仮引数 a と func1() のローカル変数 b、func2() のローカル変数 G2 と c が利用できる。
- func2() では同名の G2 と c が宣言されているため、グローバル変数 G2 と func1() で定義された c は参照できない。

このように内側で定義された関数は自分自身の中で定義されたローカル変数があるかを探し、見つからない場合には一つ上のレベルでの変数を探す。これがスコープチェインである。JavaScript の関数のスコープは関数が定義されたときのスコープチェインが適用される (次の実行例 3.3の 14 行目参照)。これをレキシカルスコープと呼ぶ。レキシカルスコープは静的スコープとも呼ばれる。これに対して実行時にスコープが決まるものは動的スコープと呼ばれる。

実行例 3.3 変数のスコープを次の例で確かめる。

```
 1  let S = "global";
 2  function func1(){
 3      console.log(S);
 4      return 0;
 5  }
 6  function func2(){
 7      console.log(S);
 8      let S = "local";
 9      console.log(S);
10      return 0;
11  }
12  function func3(){
13      let S = "local";
14      func1();
15      return 0;
16  }
17  function func4(){
18      let S = "local in func4";
19      func5 = function(){
20          console.log(S);
21          return 0;
22      };
23      console.log(S);
24      return 0;
25  }
```

このリストは次のようになっている。

- func1() から func4() まで 4 つの関数が定義されている。
- 1 行目ではグローバル変数 S が宣言されて、文字列"global" で初期化されている。
- 2 行目から 5 行目で func1() が定義されている。
  - 3 行目で S の値をコンソールに出力している。
  - この関数内で変数 S の宣言がないので 1 行目のグローバル変数 S が参照される。

```
>func1();
global
0
```

- 6 行目から 11 行目で func2() が定義されている。
  - 7 行目と 9 行目で S の値をコンソールに 2 回出力している。
  - この関数内で変数 S は 8 行目で let を用いてローカル変数として宣言されている。
  - let で宣言された変数はブロックで宣言される前に使用できないので実行時にエラーが発生する。

```
>func2();
VM362:2 Uncaught ReferenceError: S is not defined
    at func2 (<anonymous>:2:17)
    at <anonymous>:1:1
```

- 12 行目から 16 行目で func3() が定義されている。
  - 13 行目でローカル変数 S を定義して、初期値を"local"としている。
  - 14 行目で初めに定義した関数 func1() を呼び出している。
  - func1() の実行時は、この関数が定義された時の変数 S(1 行目) が参照される。

```
>func3();
global
0
```

- 17 行目から 25 行目で func4() が定義されている。
  - 18 行目でローカル変数 S の値を設定している。
  - 23 行目の出力は 18 行目での値となる。

```
>func4();
local in func4
0
```

  - 19 行目では関数オブジェクトを変数 func5 に代入している。これにより func5() という関数が定義される。

- let による変数の宣言がないので変数 func5 はグローバル変数となる (20ページの変数のスコープの 4参照)。
- func5() 内では変数 S の内容を出力が定義されている。
- func4() を実行した後では func5() が実行できる。
- 関数 func5() が定義された段階では変数 S が、この関数を定義している func4() の中にあるので、18 行目の変数 S が参照される。

```
func5();
local in func4
0
```

課題 3.3 実行例 3.3 における変数の宣言をすべて let から var に変えて func1() から func5() まで順に関数を実行した結果を記せ。

3.3 JavaScript における関数の特徴

3.3.1 関数もデータ

JavaScript では関数もデータ型のひとつなので次のようなことが可能となっている。

- 関数の定義を変数に代入したり (実行例 3.3の 19 行目以降参照) 関数の引数として渡す。
- 代入はいつでもできるので、実行時に関数の定義を変える。
- 関数の戻り値として関数自体を返す。

3.3.2 コールバック関数

実行例 3.3の 19 行目以降の関数オブジェクトでは function の後には関数名がない。このような関数は無名関数と呼ばれる。HTML 文書などでは、イベント (マウスがクリックされた、一定の時間が経過した) が発生したときに、その処理を行う関数を登録する必要がある。関数に引数として渡される関数をコールバック関数という。

実行例 3.4 次の例は、一定の経過時間後にある関数を呼び出す window オブジェクトの setTimeout() メソッドの使用例である。

```
1  let T = new Date();
2  window.setTimeout(
3    function callMe(){
4      let NT = new Date();
5      if(NT.getTime()-T.getTime()<10000) {
6        console.log(Math.floor((NT.getTime()-T.getTime())/1000));
7        window.setTimeout(callMe,1000);
```

```
8       }
9     },1000);
```

- 1 行目では実行開始時の時間を変数 T に格納している。
- このメソッドは一定時間経過後に呼び出される関数と、実行される経過時間 (単位はミリ秒) を引数に取る。
- 実行する関数は 3 行目から 9 行目で定義されている。
- この関数内で一定時間経過後にこの関数を呼び出す。この関数に名前は callMe である (3 行目)[2]。
- 4 行目で呼び出されたときの時間を求め、経過時間が 10000 ミリ秒以下であれば (5 行目)、経過時間を秒単位で表示する (6 行目)。
- さらに、自分自身を 1 秒後に呼び出す (7 行目)。

setTimeout() メソッドのコールバック関数の引数に値を渡したいときは、setTimeout() メソッドの 3 番目以降の引数に記述する。

課題 3.4 次のリストは上記の例のコールバック関数を引数付にしたものである。

```
let T = new Date();
window.setTimeout(
  function callMe(L){
    let NT = new Date();
    if(NT.getTime()-T.getTime()<10000) {
      console.log(Math.floor((NT.getTime()-T.getTime())/1000));
      L +=1000;
      window.setTimeout(callMe,L,L);
    }
   },1000,1000);
```

コンソールの出力結果と動作を確認しなさい。

3.3.3　自己実行関数

関数を定義してその場で直ちに実行することができる。次のコードは課題 1.2の関数の中身である。

```
let i;
for(i=1;i<10;i++) {
  console.log(`${i} ${i*i}`);
}
```

[2] function オブジェクトの引数を表す配列のようなオブジェクト arguments のプロパティ callee を使うのが従来の方法である。ECMAScript 2015 では arguments が非推奨となったため、コールバック関数に名前を付けている。

このプログラムを実行すると 1 から 9 までの値とそれの 2 乗の値がコンソールに出力される。実行後に、コンソールに `i` と入力すれば `10` が出力される。

一方、課題 1.2では関数が定義されただけで何も出力されないので、`foo()` と入力して関数を実行する必要がある。この場合、変数 `i` は関数内のローカル変数なので関数実行後、変数 `i` の値は参照できない (`undefined` が出力される)。

前者の場合は変数 `i`、後者の場合は関数名 `foo` がグローバル変数が残っている。JavaScript では定義した関数をその場で実行する方法があり、これを利用すると不要なグローバル変数を残さないことができる。

次の例はこれを実現している。

```
(function(){
  let i;
  for(i=1;i<10;i++) {
    console.log(`${i} ${i*i}`);
  })();
```

関数の定義を全体で `()` で囲み、そのあとに関数の呼び出しを示すための `()` を付ける。

この技法は、初期化の段階で 1 回しか実行しない事柄を記述し、かつグローバルな空間に余計な変数などを残さない手段として JavaScript のライブラリーで用いられる。

3.4 クロージャ

関数内部で宣言された変数は、その外側から参照することができない。つまり、その関数は関数内のローカル変数を閉じ込めている。しかし、関数内部で定義された関数を外部に持ち出す (グローバルな関数にする) と、持ち出された関数のスコープチェイン内に定義された親の関数のスコープを引き継ぐので、親の関数のローカル変数の参照が可能となる。

関数とその依存する環境 (変数や呼び出せる関数などのリスト) を合わせたものをその関数のクロージャと呼ぶ。

3.4.1 クロージャの例 – 変数を隠す

ここで上げる例は実用に乏しいと思われるかもしれないが、この後に出てくるオブジェクトの項の例でより実用的なものと理解できるであろう。

実行例 3.5 次の関数を考える。

```
1 function f1() {
2   let n=0;
3   return function() {
4     return n++;
5   };
6 }
```

- ローカル変数 n を定義し、初期値を 0 としている (2 行目)。
- 戻り値はローカル変数 n の値を返す関数である (3 行目から 5 行目)。
- この戻り値の関数はローカル変数の値を 1 増加させている (5 行目)。

これを次のように実行する。

- 変数 ff に関数 f1() で返される関数オブジェクトを代入する。これにより変数 ff は関数オブジェクトになる。

```
>ff= f1();  //関数の定義が表示される
```

- f1() 内のローカル変数は参照できない。

```
>n;
VM351:2 Uncaught ReferenceError: n is not defined(…)
```

- 戻り値の関数は変数 ff を用いて参照できる。

```
>ff(); // 戻り値は 0
>ff(); // 戻り値は 1
>ff(); // 戻り値は 2
```

何回か実行すると戻り値が順に増加している。つまり、ローカルには変数 n が存在している。

- もう一度関数 f1() を実行すると新しい関数が得られる。

```
>ff2=f1();  // 関数の定義が表示
>ff();  // 戻り値は 3
>ff2(); // 戻り値は 0
```

3.4.2　クロージャの例 – 無名関数をその場で実行する

実行例 3.5で定義された関数 f1 を一度だけ実行して、それがこれ以上実行されないようにするためにはこの関数を無名関数としてその場で実行すればよい。

実行例 3.6 次のリストは、使い捨ての関数の例である。このような方法を用いるとオブジェクトに共通の変数を持つことができる。

```
let foo = (function () {
  let n=0;
  return function() {
    return n++;
  };
})();
```

- 無名関数を定義した部分を () でくくり、引数リストをその後の () に記述する。ここでは、引数がないので中は空である。
- 戻された関数オブジェクトを変数 foo に代入する。
- 前と同じように実行できる。

```
>foo();  // 戻り値は 0
>foo();  // 戻り値は 1
>foo();  // 戻り値は 2
```

実行例 3.7 次の例は関数内のローカル変数の値を単純に返すと、不都合が起こる例である[3]。

```
1  function f2() {
2    let a = [];
3    let i;
4    for(i=0; i<3; i++) {
5      a[i] = function() {
6        return i;
7      };
8    }
9    return a;
10 }
```

- 変数 a を空の配列で初期化する (2 行目)。
- この配列に配列の添え字の値を返す関数を定義する (4 行目から 8 行目)
- 配列全体を戻り値として返す (9 行目)。

これを実行すると次のようになる[4]。

```
>funcs=f2()
[function a.(anonymous function)(), function a.(anonymous function)(),
                function a.(anonymous function)()]
>funcs[1]() // 戻り値は 3
```

これは、f2() を実行した後では、変数 i の値が 3 となっており、それぞれの関数が実行されるときにはこの値が参照されるためである。

実行例 3.8 この不具合は、for の制御変数 i を for 文の初期化のところで宣言すれば発生しない[5]。

[3] この例は Stoyan Stefanov(水野貴明、渋川よしき訳) オブジェクト指向 JavaScript、アスキーメディアワークス、2012 年の 103 ページ以降から取りました。
[4] ここでの出力例は Chrome によるものである。ブラウザによっては出力が異なる。
[5] ECMAScript 2015 では for 文の初期化内で let で宣言された変数のスコープ規則が別に定められている。

```
1  function f2() {
2    let a = [];
3    for(let i=0; i<3; i++) {
4      a[i] = function() {
5        return i;
6      };
7    }
8    return a;
9  }
```

これを実行すると、期待した結果が得られる。

```
>funcs2 = f3();
[function anonymous(), function anonymous(), function anonymous()]
>funcs2[0]();  // 戻り値は 0
>funcs2[1]();  // 戻り値は 1
```

この形で let による変数の宣言を var に変えると変数の巻き上げにより正しく動作しない。正しく動作させるためには値を関数の引数に (値渡しで) 渡すことで、スコープチェーンを切ればよい。

```
1  function f3() {
2    let a = [], i;
3    for(i=0; i<3; i++) {
4      a[i] = (function(x){
5        return function() {
6          return x;
7        }
8      })(i);
9    };
10   return a;
11 }
```

- 5 行目から 9 行目で、その場で実行される、引数を取る無名関数を用意している。
- この戻り値は、5 行目の無名関数の引数の値を返す無名関数 (6 行目から 8 行目) である。
- 仮引数には、実行されたときの i のコピーが渡されるので、その後、もとの変数の値が変わっても呼び出されたときの値が仮引数の x に保持される。

let を利用することで記述が分かりやすくなっていることが分かる。

課題 3.5 実行例 3.7において、for 文の代わりに while 文に書き換えたときに正しく動作するようにせよ。ただし、直前のリストのように即時実行関数を用いないで実現すること。

第4回　オブジェクト

4.1　配列とオブジェクト

配列はいくつかのデータをまとめて一つの変数に格納している。各データを利用するためには `foo[1]` のように数による添え字を使う。これに対し、オブジェクトでは添え字に任意の文字列を使うことができる。

実行例 4.1 次の例はあるオブジェクトを定義してそれの各データにアクセスする方法を示している。

```
let person = {
  name : "foo",
  birthday :{
    year : 2001,
    month : 4,
    day : 1
  },
  "hometown" : "神奈川",
}
```

- オブジェクトは全体を `{}` で囲む。
- 各要素はキーと値の組で表される。両者の間は `:` で区切る。
- キーは任意の文字列でよい。
- キーの文字列が変数名の約束事に合わないときはキー全体を `""` で囲う必要がある。
- 値は JavaScript で取り扱えるデータなあらば何でもよい。上の例ではキー `birthday` の値がオブジェクトとなっている。値が関数であるキーはそのオブジェクトのメソッドと呼ばれる。
- 各要素の値を取り出す方法は 2 通りある。
  - `.` 演算子を用いてオブジェクトのキーをそのあとに書く。

    ```
    >person.name;
    "foo"
    ```
  - 配列と同様に `[]` 内にキーを文字列として指定する。

    ```
    >person["name"];
    "foo"
    ```

- オブジェクトの中にあるキーをすべて網羅するようなループを書く場合や変数名として利用できないキーを参照する場合には後者の方法が利用される。
- キーの値が再びオブジェクトであれば、前と同様の方法で値を取り出せる。

```
>person.birthday;
Object {year: 2001, month: 4, day: 1}
>person.birthday.year;
2001
>person.birthday["year"];
2001
```

この例のように取り出し方は混在してもよい。

- キーの値は代入して変更できる。

```
>person.hometown;
"神奈川"
>person.hometown="北海道";
"北海道"
>person.hometown;
"北海道"
```

- 存在しないキーを指定すると値として `undefined` が返る。

```
>person.mother;
undefined
```

- 存在しないキーに値を代入すると、キーが自動で生成される。

```
>person.mother = "aaa";
"aaa"
>person.mother;
"aaa"
```

- オブジェクトのキーをすべて渡るループは `for-in` で実現できる。
  - `for( v in obj)` の形で使用する。変数 `v` はループ内でキーの値が代入される変数、`obj` はキーが走査されるオブジェクトである。[1]
  - キーの値は `obj[v]` で得られる。

```
>for(i in person) { console.log(‘i ${person[i]}‘);};
name foo
birthday [object Object]
```

[1] 1ページで挙げた文献 [1,162 ページ] にはオブジェクトの独自プロパティだけを見たい場合には `hasOwnProperty()` を使うことが推奨されている。

```
hometown 北海道
mother aaa
undefined
```

最後の `undefined` は `for` ループの戻り値である。

- `Object.keys()` にオブジェクトを渡すと、キー名の配列が得られる。

```
>Object.keys(person)
(3) ["name", "birthday", "hometown"]
```

これを用いてオブジェクトの値を得ることも可能である。子の利用法は後の回で解説をする。

オブジェクトを`{}`の形式で表したものをオブジェクトリテラルとよぶ。

課題 4.1 `window` オブジェクトにはどのようなプロパティがあるか調べよ。2 つ以上のブラウザで実行し、比較すること。

4.2 オブジェクトリテラルと JSON

JSON(JavaScript Object Notation) はデータ交換のための軽量なフォーマットである。形式は JavaScript のオブジェクトリテラルの記述を文字列として表現したものである。

- 正しく書かれた JSON フォーマットの文字列をブラウザとサーバーの間でデータ交換の手段として利用できる。
- JavaScript 内で、JSON フォーマットの文字列を JavaScript のオブジェクトに変換できる。
- JavaScript 内のオブジェクトを JSON 形式の文字列に変換できる。

JavaScript のオブジェクトと JSON フォーマットの文字列の相互変換の手段を提供するのが `JSON` オブジェクトである。

実行例 4.2 次の例は 2 つの同じ形式のオブジェクトを通常の配列に入れたものを定義している。

```
let persons = [{
  name : "foo",
  birthday :{ year : 2001, month : 4, day : 1},
  "hometown" : "神奈川",
},
{
  name : "Foo",
  birthday :{ year : 2010, month : 5, day : 5},
  "hometown" : "北海道",
}];
```

次の例はこのオブジェクトを JSON に処理させたものである。

```
>s = JSON.stringify(persons);
"[{"name":"foo","birthday":{"year":2001,"month":4,"day":1},
"hometown":"神奈川"},
{"name":"Foo","birthday":{"year":2010,"month":5,"day":5},
"hometown":"北海道"}]"
>s2 = JSON.stringify(persons,["name","hometown"]);
"[{"name":"foo","hometown":"神奈川"},{"name":"Foo","hometown":"北海道"}]"
>typeof s;
"string"
>o = JSON.parse(s2);
[Object, Object]
>o[0];
Object {name: "foo", hometown: "神奈川"}
```

- JavaScript のオブジェクトを文字列に変更する方法は JSON.stringify() を用いる。このまま見ると"がおかしいように見えるが表示の関係でそうなっているだけである。なお、結果は途中で改行を入れているが実際は一つの文字列となっている。
- JSON.stringigy() の二つ目の引数として対象のオブジェクトのキーの配列を与えることができる。このときは、指定されたキーのみが文字列に変換される。
- ここでは、"name" と "hometown"が指定されているので"birthday"のデータは変換されていない。
- JSON データを JavaScript のオブジェクトに変換するためのメソッドは JSON.parse() である。
- ここではオブジェクトの配列に変換されたことがわかる。
- 各配列の要素が正しく変換されていることがわかる。

課題 4.2 実行例 4.2において、

```
s3 = JSON.stringify(persons,["year"]);
```

としたときの結果はどうなるか調べなさい。

4.3　クラス

4.3.1　クラスの宣言とインスタンスの生成

前節で同じキーを持つオブジェクトを複数作成したい場合に同じようなコードを繰り返すのは問題である。また、キーの追加をするときの修正の手間がかかりすぎてプログラムのメンテナンスが面倒になる。

そこで、オブジェクトのひな形を作成し、それをもとにオブジェクトを構成することが考えられる。このオブジェクトのひな形は通常クラスと呼ばれる。ECMAScript 2015 ではクラスの宣言ができる。クラスから作成されたオブジェクトはこのクラスのインスタンスと呼ばれる。

実行例 4.3 実行例 4.1をクラスを用いて書き直すと次のようになる。

```
1  class Person{
2    constructor(name, year, month, day, hometown="神奈川"){
3      this.name = name;
4      this.birthday = {
5        year : year,
6        month : month,
7        day : day
8      };
9      this["hometown"] = hometown;
10   }
```

- 1 行目がクラスの宣言である。キーワード `class` の後にクラス名 `Person` を記述する。通常、クラス名は大文字で始まる。関数の時と同様にクラスの宣言を変数に代入することもできる。
- その後の`{`と`}`内にクラスの記述を行う。
- クラスの定義には初期化を行う `constructor()` が必須である。ここでは 5 つの引数を取るコンストラクタが定義されている。5 番目の引数はデフォルトの値が指定されている。
- キーワード `this` はクラスから作成されたインスタンスを指す。
- 3 行目から 9 行目で、そのインスタンスのオブジェクトのキーはプロパティと呼ばれる。プロパティの名前は前のオブジェクトと同じである。
- 9 行目の定義はコンストラクタの最後の引数はデフォルトの値が与えられている。
- クラス内のコードは `strict` モードで実行される。

クラスの定義からインスタンスを作成するためにはキーワード `new` をつけてクラスを呼び出す。上の例ではコンストラクタが引数を必要としているので次のようになる。

```
>p = new Person("foo",2001,8,18);
Person {name: "foo", birthday: {…}, hometown: "神奈川"}
>p.birthday
{year: 2001, month: 8, day: 18}
>p.hometown;
"神奈川"
```

コンストラクタの最後の引数にはデフォルト値が指定されているので、この実行例のように値が指定されていない場合にはデフォルト値に設定されていることが分かる。

課題 4.3 実行例 4.1と同様に、`class` を用いて作成されたインスタンスについてプロパティの値の変更、プロパティの追加ができるか確認しなさい。

4.3.2　クラスメソッド

クラス内では constructor のほかにメソッドと呼ばれる関数で定義されたプロパティがある。また、メソッドにはアクセッサプロパティと呼ばれるオブジェクトに値を渡すセッター、値を得るゲッターの 2 種類を指定することもできる。ECMAScript 2015 ではゲッターやセッターにはキーワード get と set を用いる。

実行例 4.4 次のリストは Person に文字列に変換する toString() と、実行時における年令を返すゲッター age を追加したものである。

```
1  class Person{
2    constructor(name, year, month, day, hometown = "神奈川"){
3      this.name = "foo";
4      this.birthday = {
5        year : year,
6        month : month,
7        day : day
8      };
9      this["hometown"] = hometown;
10   }
11   toString() {
12     return `${this.name}さんは`+
13       `${this.birthday.year}年${this.birthday.month}月${this.birthday.day}日に` +
14      `${this.hometown}で生まれました。`;
15   }
16   get age() {
17     let today = new Date();
18     let age = today.getFullYear() - this.birthday.year;
19     if(today.getTime() <
20          new Date(today.getFullYear(),
21                   this.birthday.month-1,
22                   this.birthday.day).getTime()) age--;
23     return age;
24   }
25 }
```

11 行目から 15 行目でメソッド toString() が定義されている。

一般に toString() メソッドはすべてのオブジェクトに定義されていて、文字列が必要な時に呼び出される。ここでは「~ さんは~ 年~ 月~ 日に~ で生まれました。」という表示を返す。

実行例は次のとおりである。文字列が必要なときに呼び出されることがわかる。

```
>p.toString();//toString() の明示的呼び出し
"foo さんは 2001 年 4 月 1 日に神奈川で生まれました。"
```

```
>‘${p}‘;        //暗黙の toString() 呼び出し
"foo さんは 2001 年 4 月 1 日に神奈川で生まれました。"
```

16 行目から 24 行目でゲッター age が定義されている。

- 16 行目でキーワード get をつけて、ゲッター age() を宣言している。メソッドは関数なので () をつける必要がある。ただし、ゲッターに仮引数をつけることはできない。
- 17 行目でアクセス時の時間を変数 today に保存している。
- 18 行目でアクセス時の年から誕生日の年を引いている。
- 19 行目から 22 行目で、今年の誕生日が過ぎているかどうかをチェックしている。アクセス時の年と誕生日の月を日をもとに日付を作成し、その時間 (getTime() を利用) を比較することでチェックができる。
- 誕生日前ならば 18 行目で求めた値を 1 減少させる。

実行例は次のとおりである。

```
>p.age;
16
>p.age = 50;
50
>p.age;
16
```

- 定義の方はにはメソッドであることを示す () があるが、利用するときはプロパティと同じようになる。() を付けるとエラーになる。
- 代入の式はエラーがなく実行できるが、ゲッターの機能は変わらない。代入はセッターの呼び出しが行われる。

```
p.age(10);
VM87:1 Uncaught TypeError: p.age is not a function
    at <anonymous>:1:3
```

課題 4.4 指定された日付における年令を求めるメソッドを作成しなさい。求める年齢は次の条件を満たすこと。

- 引数がない場合には age と同じ
- 年しかない場合にはその年の 1 月 1 日現在
- 年と月しかない場合にはその年月の 1 日現在

課題 4.5 実行例 4.4のリストにあるプロトタイプメソッド age() の前にある get を省略して通常のメソッドとして定義しときの実行方法について報告せよ。

課題 4.6 実行例 4.4において、age プロパティがセッターとして使われたときには注意を促すメッセージを表示するようにしなさい。

4.3.3　継承

すでにあるクラスをもとに機能の追加や変更を加えた新しいクラスを作ることができる。新しいクラスはもとになるクラスを継承しているという。もとのクラスは新しいクラスのスーパークラス、新しいクラスはもとになるクラスのサブクラスという。JavaScript では複数のクラスから同時に継承する多重継承はサポートされていない。

実行例 4.5 次の例は実行例 4.4のクラス Person を継承して新しいクラス Student を作成するものである。

```
1  class Student extends Person {
2    constructor(name, id, year, month, day, hometown) {
3      super(name, year, month, day, hometown);
4      this.id = id;
5    }
6  }
```

- クラスを継承するためにはクラス名の後に、キーワード extends を付けて継承するクラス名を書く (1 行目)。
- Student クラスのコンストラクタには Person クラスで必要なパラメタのほかに id が付け加えている (2 行目)。
- 親クラス (スーパークラス) のコンストラクタを呼び出すために super() を実行する (3 行目)
- 4 行目で追加のプロパティの設定を行っている。

実行例は次のとおりである。

```
>s = new Student("foo",1523999,2001,4,1);//Person のデフォルト引数値が設定されている
Student {name: "foo", birthday: {…}, hometown: "神奈川", id: 1523999}
> `${s}` //toString() も Person で定義されたものが使われる
"foo さんは 2001 年 4 月 1 日に神奈川で生まれました。"
>s2 = new Student("foo",1523999,2001,4,1,"厚木");//デフォルト以外の設定もできる
Student {name: "foo", birthday: {…}, hometown: "厚木", id: 1523999}
>`${s2}`;
"foo さんは 2001 年 4 月 1 日に厚木で生まれました。"
```

課題 4.7 Student クラスにプロパティid も表記するような toString() メソッドを定義すると、Person の toString() が上書きされることを確認しなさい。

4.3.4　instanceof 演算子

instanceof 演算子はオブジェクトを生成したコンストラクタ関数が指定されたものかを判定する。

実行例 4.6 次の結果は Chrome で、実行例 4.4の例である。

```
>p instanceof Person
true
>p instanceof Object;
true
>p instanceof Date;
false
```

Person オブジェクトが Object を継承しているので p instanceof Object が true となっている。

4.3.5 静的メソッド

クラスに対して、インスタンスを作成しないで使用できる静的メソッドを定義できる。静的メソッドはキーワード static を付ける。インスタンス化されたオブジェクトからは使用できない。

Math オブジェクトにある各関数が Math オブジェクトの静的メソッドの例になっている。

実行例 4.5のプロパティid を重複がなくかつ連続な値に自動で設定しようとするとクラスにおいて変数を管理する変数 (クラス変数) が必要となる。

実行例 4.7 クラスメソッドを用いて連続した id を作成する方法は次のようになる。

```
1  class Student extends Person {
2    static getNextId(){
3      return Student.nextId++;
4    }
5    constructor(name, year, month, day, hometown) {
6      super(name, year, month, day, hometown);
7      this.id = Student.getNextId();
8    }
9  }
10 Student.nextId = 10000;
```

- 2 行目から 4 行目で次の id を求めるクラスメソッドを定義している。
- ここではクラス変数 nextId を 1 増加させている (3 行目)
- nextId の初期化は 10 行目で行っている。
- 初期化の方法からも推察されるように、この値は途中で変更が可能である。

実行例は次のとおりである。

```
>s1 = new Student("foo1",2001,4,1);
Student {name: "foo", birthday: {…}, hometown: "神奈川", id: 10000}
>s2 = new Student("foo2",2001,5,1);
```

```
Student {name: "foo", birthday: {…}, hometown: "神奈川", id: 10001}
Student.nextId = -100;
-100
>s3 = new Student("foo2",2001,6,1);
Student {name: "foo", birthday: {…}, hometown: "神奈川", id: -100}
```

3 つ目のインスタンスを作成する前にこの値を上書きしているので、最後のインスタンスは前の 2 つと異なるものに設定できていることが分かる。

この欠点を解消するにはクラスメソッドで定義する代わりにクラスをクロージャの中で定義して、その中に変数を置く方法がある。

実行例 4.8 次の例は即時実行関数内に id を管理する変数を用意して、その関数がクラス式を返すようにしたものである。

```
1  const Student = (function(){
2    let id = 10000;
3    return class extends Person {
4      constructor(name, year, month, day, hometown) {
5        super(name, year, month, day, hometown);
6        this.id = id++;
7      }
8    }
9  })();
```

- 即時実行関数の戻り値はクラス式であり、それを変数 Student に代入している。同じ名前のクラスが再定義されないようにするため変数を const で宣言している。
- 2 行目で次に設定する id を保存する変数を初期化している。
- 関数の戻り値としてクラス式を返す。クラスの定義は以前のものとほとんど同じである。
- 違いはインスタンスに設定した後で id を管理する変数を増加させているところ (6 行目の右辺)。

実行例は次のとおりである。

```
>s = new Student("foo",2001,4,1);  //初めのインスタンスの id は 10000
{name: "foo", birthday: {…}, hometown: "神奈川", id: 10000}
>`${s}`;
"foo さんは 2001 年 4 月 1 日に神奈川で生まれました。"
>s2 = new Student("foo",2001,4,1,"厚木");  //次のインスタンスの id は 10001
{name: "foo", birthday: {…}, hometown: "厚木", id: 10001}
>`${s2}`;
"foo さんは 2001 年 4 月 1 日に厚木で生まれました。"
```

第5回　オブジェクトの詳細

5.1　class の実体

ECMAScript 2015 で導入されたキーワード class の実体はこれまでの ECMAScript に新しい機能を追加したわけではなく、従来のオブジェクトや継承に関する記述を簡単にしたものに過ぎない[1]。実行例 4.3において typeof Person の結果は次のとおりである。

```
>typeof Person;
"function"
```

従来の ECMAScript では関数を用いてクラスを定義する。class でもその手法が使われていることを示す。

実行例 5.1 次の例はコンストラクタ関数を用いて、前の例と同じオブジェクトを作成している。ここでは実行例 4.3の例で確認する。

```
function PersonF(name, year, month, day, hometown = "神奈川"){
    this.name = name;
    this.birthday = {
      year : year,
      month : month,
      day : day
    };
    this["hometown"] = hometown;
}
```

この定義では constructor の内容をそのまま関数にしているに過ぎない。なお、ここではメソッドなどが定義されていないが、それについては後で解説をする。

```
>p = new PersonF("foo",2001,4,1);
PersonF {name: "foo", birthday: {…}, hometown: "神奈川"}
```

5.2　オブジェクト属性

JavaScript の関数オブジェクトには prototype、class と extensible という 3 つの属性がある。prototype 属性はオブジェクトの継承に関係するので、最後に説明をする。

[1] このことを syntactic sugar(糖衣構文) と呼ぶ。

5.2.1　class 属性

オブジェクトの class 属性はオブジェクトの型情報を表す文字列である。最新の JavaScript でもこの属性を設定する方法はない。直接値を取得する方法もない。クラス属性は間接的にしか得られない。組み込みのコンストラクタで生成されたオブジェクトではそのクラス名が間接的に得られるが、独自のコンストラクタ関数では、"Object"しか得られない。

実行例 5.2 Object から継承した toString() を直接呼び出すと次のような結果になる。

```
>p = new PersonF("foo",2001,4,1);
PersonF {name: "foo", birthday: {…}, hometown: "神奈川"}
>`${p}`;
"[object Object]"
```

ここで、Person() コンストラクタには toString() メソッドが定義されていないので、もともとの Object で定義されている toString() が呼び出されている。

5.2.2　extensible 属性

extensible はオブジェクトに対してプロパティの追加ができるかどうかを指定する。JavaScript ではこの属性の取得や設定ができる関数が用意されている。

この属性の取得は Object.isExtensible() に調べたいオブジェクトを引数にして渡す。オブジェクトのプロパティを拡張できなくするためには Object.preventExtension() に引数として設定したいオブジェクトを渡す。

実行例 5.3 実行例 4.1の例で確認する。

```
>p.mother = "Alice";
 Alice
>p.grandmother = "Old Alice";
 "Old Alice"
>Object.preventExtensions(p);
 {name: "foo", birthday: {…}, hometown: "神奈川", mother: "Alice", grandmother: "Old Alice"}
>p.father = "Bob";
 "Bob"
>p.father;
 undefined
>delete p.mother;
 true
>p.mother;
 undefined
```

- Object.preventExtensions(p) を実行する前では存在しない属性の追加ができている (p.mother)。

- Object.preventExtensions(p) を実行後は、新しい属性が定義できていない (p.father)。
- Object.preventExtensions(p) では属性の削除までは禁止できない (delet p.mother が成功していて、値を参照すると undefined が返る)。

属性の削除まで禁止したい場合には Object.seal() を用いる。一度この関数を実行されたオブジェクトは解除できない。
引き続いて実行すると次のようになる。

```
>Object.seal(p);
 {name: "foo", birthday: {…}, hometown: "神奈川", grandmother: "Old Alice"}
>Object.isSealed(p);
 true
>delete p.grandmother;
 false
>p.grandmother;
 Old Alice
>p.grandmother = "Very Old Alice";
 "Very Old Alice"
>p.grandmother;;
 "Very Old Alice"
```

- すでに seal された状態になっているかどうかは、Object.isSealed() で調べられる。
- seal された状態ではプロパティを削除できない (delete p.grandmother の結果が false)。
- seal された状態ではプロパティの値を変えることができる (p.grandmother の値の書き換えができている)。

オブジェクトを最も強く固定するためには Object.freeze() を用いる。また、この状態を確認するためには Object.isFrozen() を用いる。引き続いて Object.freeze() を実行すると次のようになる。

```
>Object.freeze(p);
 {name: "foo", birthday: {…}, hometown: "神奈川", grandmother: "Very Old Alice"}
>p.grandmother = "Very Very Old Alice";
 "Very Very Old Alice"
>p.grandmother;
 "Very Old Alice"
```

p.grandmother の値が設定できていないことがわかる。このような状態で属性値を変えたい場合には、属性にたいするセッターメソッドを定義することになる。
このオブジェクトのプロパティbirthday の値はオブジェクトである。このオブジェクトは seal されていない。

```
>p.birthday;
 {year: 2001, month: 4, day: 1}
>delete p.birthday;
 false
>p.birthday = {};
 {}
>p.birthday;
 {year: 2001, month: 4, day: 1}
>delete p.birthday.year;
 true
>p.birthday;
 {month: 4, day: 1}
        undefined
```

p.birthday は削除できないし、値の変更もできない。一方で、p.birthday.year は削除できる。

課題 5.1 実行例 4.3のクラスの記述に対して次のことを行いなさい。

1. 実行例 5.3のようにオブジェクトの extensible 属性を変更したときの結果を比較せよ。
2. クラス Person を freeze できるか確認せよ。また、freeze 後にインスタンスを作成したとき、インスタンスは freeze されているか確認せよ。

JavaScript におけるクラスとそれから生成されるインスタンスは通常のオブジェクト指向言語と異なり、構造を変えることができる。固定化はインスタンスごとに行う必要がある。この手間を省く方法を 5.4節で解説する。

5.2.3 prototye 属性

オブジェクトの prototype 属性の値は、同じコンストラクタ関数で生成された間で共通のものになっている。オブジェクトリテラルで生成されたオブジェクトは Object.prototype で参照できる。new を用いて生成されたオブジェクトはそのコンストラクタ関数の prototype を参照する。このコンストラクタ関数の prototype もオブジェクトであるから、それの prototype も存在する。この一連の prototype オブジェクトをプロトタイプチェーンとよぶ。

5.2.4 prototype の使用例

実行例 5.4 次の例は実行例 5.1の属性を省略し、いくつかのメソッドを prototype に追加したものである。

```
1  PersonF.prototype = {
2    toString:function() {
3      return `${this.name}さんは`+
```

```
4       `${this.birthday.year}年${this.birthday.month}月${this.birthday.day}日に` +
5     `${this.hometown}で生まれました。`;
6   },
7   get age(){
8     let today = new Date();console.log(x);
9     let age = today.getFullYear() - this.birthday.year;
10    if(today.getTime() <
11         new Date(today.getFullYear(),
12                  this.birthday.month-1,
13                  this.birthday.day).getTime()) age--;
14    return age;
15  },
16  constructor: PersonF
17 }
```

これらの定義は前と同じである。オブジェクトリテラルの形式で prototype に代入している。異なるところは constructor プロパティに自分自身を定義しているところである。

なお、get キーワードの代わりに set キーワードを用いるとセッターを定義できる。

課題 5.2 実行例 4.4の Person クラスの prototype を調べよ。

5.3 関数によるオブジェクトの継承

JavaScript では prototype を用いることでメソッドの継承が可能となっている。

実行例 5.5 次のリストは実行例 5.4の Person を継承して、学籍番号を追加のプロパティとする Student オブジェクトのコンストラクタ関数である。

```
1  const StudentF = (function(){
2    let id = 10000;
3    return function(name, year, month, day, hometown = "神奈川") {
4      this.name = name
5      this.birthday = {
6        year : year,
7        month : month,
8        day : day
9      };
10     this["hometown"] = hometown;
11     this.id = id++;
12     }
13   })();
```

```
14 StudentF.prototype = new PersonF();
15 StudentF.prototype.constructor = StudentF;
```

- 関数を用いた継承では super が使えないので、継承元のクラスで定義されたプロパティ(name など) は継承先で定義する必要がある (4 行目から 10 行目)。
- 一方、メソッドはインスタンスで共有されるので継承元で定義をする必要はないが、参照できるようにする必要がある。そのために 14 行目で継承元のクラスを作成して継承するクラスの prototype を書き直している (14 行目)。
- このままでは constructor プロパティが PersonF になるので、StudentF.prototype.constructor を継承するクラスに置き換える (15 行目)

5.4　WeakMap によるインスタンスの安全性の確保

クラスのクロージャ内に変数を閉じ込めることで、クラス全体に共通のデータを保護することはすでに解説をした。これを応用してクラスのインスタンスのデータを安全に管理する方法としては ECMAScript 2015 で導入された WeakMap を用いると簡単にできる。

WeakMap はオブジェクトと同じようにキーと値の集まりである[2]。オブジェクトのキーは文字列しか使用することはできない[3]が、WeakMap のキーはオブジェクトしか使用できない。

実行例 5.6 次の例は実行例 4.4のクラスを WeakMap を利用して書き直したものである。

```
 1 const Person = (function() {
 2   const properties = new WeakMap();
 3   return class {
 4     constructor(name, year, month, day, hometown="神奈川"){
 5       properties.set(this,
 6         {
 7           "name" : name,
 8           "birthday" :{
 9             "year" : year,
10             "month": month,
11             "day" : day
12           },
13           "hometown" : hometown
14         });
15     }
16     get name(){
```

[2]ECMAScript 2015 では Map オブジェクトも定義されている。WeakMap 方がメモリーの使用状況が改善するような仕様であるのでここでは紹介しない。詳しくは [1] を参照のこと。

[3]実際には文字列以外にこの授業では紹介しない SYMBOL も利用できる。

```
17        return properties.get(this).name;
18      }
19      get birthday(){
20        return properties.get(this).birthday;
21      }
22      get hometown(){
23        return properties.get(this).hometown;
24      }
25      toString() {
26        return `${this.name}さんは`+
27          `${this.birthday.year}年${this.birthday.month}月${this.birthday.day}日に` +
28          `${this.hometown}で生まれました。`;
29      }
30      get age(){
31        let today = new Date();
32        let age = today.getFullYear() - this.birthday.year;
33        if(today.getTime() <
34           new Date(today.getFullYear(),
35                    this.birthday.month-1,
36                    this.birthday.day).getTime()) age--;
37        return age;
38      }
39    }
40  })()
```

- 2 行目で Person クラスのインスタンスの (隠された) プロパティ値を格納するための WeakMap のインスタンスを作成している。
- 4 行目から 15 行目で constructor を定義している。
  - 6 行目から 14 行目で定義されているオブジェクトは実行例 4.3で定義しているオブジェクトと同じものになっている。
  - そのオブジェクトをインスタンス (this) をキーにして、2 行目で作成した WeakMap のインスタンスに set メソッドを用いて登録している。
- 各インスタンスのプロパティを読み出すために、name、birthday、hometown の 3 つのメソッド定義している (16 行目から 24 行目)。
- インスタンスの (隠し) プロパティを保存している WeakMap から値を取り出すために、get メソッドを使用している。
- 25 行目から 29 行目の toString() メソッドと 30 行目から 38 行目の設定プロパティage は以前の定義と同じである。

次のリストはこのクラスの実行例である。

```
>p = new Person("foo",2001,4,1);
{}
```

インスタンスのプロパティは設定していないため空である。結果を展開するとメソッドが見える。
　インスタンスのそれぞれのプロパティは読み出すことができる。

```
>p.name;
"foo"
>p.birthday;
{year: 2001, month: 4, day: 1}
>p.hometown;
"神奈川"
```

メソッドも正しく動作する。

```
>p.age;
16
>‘${p}‘;
"foo さんは 2001 年 4 月 1 日に神奈川で生まれました。"
```

プロパティの値は変更できない。

```
>p.name = "Alice"; //name の値を "Alice" にする試み
>p.name;
"foo"               //name の値は変更されていない。
```

プロパティは消去できない。

```
>delete p.name;  //プロパティ name の消去の試み
true             //成功?
>p.name
"foo"            //プロパティ name は消去されていない。
```

p.birthday のプロパティの値は変更できる。

```
>p.birthday.year = 2010;          // 戻り値 2010
>p.birthday;
{year: 2010, month: 4, day: 1}   //year の値が変更されている。
```

最後の 2 つの結果の理由は問題としておく。

課題 5.3 実行例 5.6において次の問いに答えよ。

1. typeof p.name の値を確かめよ。
2. delete p.name の結果が true であるのに p.name がその後も参照できる理由は何か。

3. p.birthday にプロパティは追加できるか。また、プロパティを消去できるか確かめよ。

4. p.birthday.year の値が書き直せる理由は何か。

5. p.birthday.year の値を書き直せないように birthday メソッドを書き直せ。

5.5 エラーオブジェクトについて

エラーオブジェクトとはエラーが発生したことを知らせるオブジェクトである。通常は計算の継続ができなくなったときにエラーオブジェクトをシステムに送る操作が必要である。これを通常、エラーを投げる (throw する) という。エラーオブジェクトには表 5.1のようなプロパティがある。

表 5.1: エラーオブジェクトのプロパティ

プロパティ	説明
message	エラーに関する詳細なメッセージ。コンストラクタで渡された文字列か、デフォルトの文字列
name	エラーの名前。エラーを作成したコンストラクタ名になる

5.5.1 エラー処理の例

エラー処理の例としてインスタンス作成時に引数の妥当性をチェックする例を挙げる。

実行例 5.7 次のリストは、実行例 4.4において、コンストラクタに与えられた引数をチェックして不正な値の場合にはエラーを投げるように書き直したものである。

なお、このリストでは Person の toString() と age、継承するクラス Student の内容が以前のものと同じなので ... で省略している。

```
1  class Person{
2    static checkName(name) {
3      if(name === "") throw new Error("名前がありません");
4      return name;
5    }
6    static checkDate(y, m, d) {
7      if(m<1 || m>12) throw new Error("月が不正です");
8      let date = new Date(y,m,0);
9      if(d<1 || d>date.getDate()) throw new Error("日が不正です");
10     return {year: y, month: m, day: d};
11   }
12   constructor(name, year, month, day, hometown="神奈川"){
```

```
13      this.name = Person.checkName(name);
14      this.birthday = Person.checkDate(year, month, day);
15      this["hometown"] = hometown;
16    }
17    toString() {
18    ...
19    }
20    get age(){
21      ...
22    }
23  }
24  const Student = (function(){
25    ...
26  })();
```

- 2行目から5行目で、name が空文字であればエラーを発生させ、正しければ値をそのまま返すクラスメソッドを定義している。
- 6行目から11行目で正しい日付でないとエラーを発生させ、正しいときは日付のオブジェクトを返すクラスメソッドを定義している。
  - 7行目は月の値の範囲をチェックしている。
  - 8行目では、与えられた年と月からその月の最終の日を求めている。Date.getMonth() の戻り値が 0(1月) から 11(12月) になっているので、new Date(y,m,0) により翌月の1日の1日前、つまり、問題としている月の最終日が設定できる。
  - 9行目で与えられた範囲に日が含まれていなければエラーを発生させている。
  - 10行目 f で日付のオブジェクトを作成し、戻り値として返している。
- 13行目と14行目で、名前と日付を指定したプロパティにクラスメソッドからの戻り値で設定している。

これをいくつかのデータで実行した結果は次のようになる。

- 通常の日時ならば問題なく、オブジェクトが構成される。

```
>p = new Person("foo",1995,4,1);
Person {name: "foo", year: 1995, month: 4, day: 1}
```

- うるう年の1996年には2月29日あるので、エラーは起こらずオブジェクトが作成できる。

```
>p = new Person("foo",1996,2,29);
Person {name: "foo", year: 1996, month: 2, day: 29}
```

- うるう年ではない1995年には2月29日がないので、エラーが起きる。

```
>p = new Person("foo",1995,2,29);
 Uncaught Error: 日が不正です (…)
```

- 不正な月や日では当然、エラーが起こる。

```
>p = new Person("foo",1995,13,29);
 Uncaught Error: 月が不正です (…)
>p = new Person("foo",1995,12,0);
 Uncaught Error: 日が不正です (…)
```

課題 5.4 Person を継承した Student クラスでエラーチェックができていることを確認しなさい。

課題 5.5 実行例 5.7においてエラーチェックが完全ではない点を指摘し、その部分を改良しなさい。

5.5.2 エラーからの復帰

前節の例ではエラーが発生するとそこでプログラムの実行が止まってしまう。エラーが発生したときに、投げられた (throw された) エラーを捕まえる (catch する) ことが必要である。このためには try{...}catch{...} 構文を使用する。

- try のブロック内にエラーが発生する可能性があるコードを記述する。
- エラーが発生したときは throw を用いてエラーを発生させる。
- エラーが発生すると catch(e) ブロックが実行される。catch の後にある e は投げられたエラーオブジェクトが渡される。この変数のスコープは catch 内である。
- finally{} を付けることもできる。try や catch の処理がが実行後、必ず呼び出される。これは try や catch の部分が return 文、break 文、continue 文、return 文や新しい例外を投げたとしても呼び出される。
- try{...}catch{...} 構文は入れ子にできる。投げられたエラーに一番近い catch にエラーが捕まえられる。

実行例 5.8 次のリストは実行例 5.7において try{...}catch{...} 構文を用いてオブジェクトが正しくできるまで繰り返すようにしたものである。なお、このリストはブラウザで実行することを想定している。

```
1  function test() {
2    for(;;) {
3      try {
4        let y = Number(prompt("生まれた年を西暦で入力してください"));
5        console.log(‘年:${y}‘)
6        let m = Number(prompt("生まれた月を入力してください"));
7        console.log(‘月:${m}‘)
```

```
 8        let d = Number(prompt("生まれた日を入力してください"));
 9        console.log(‘日:${d}‘)
10        return new Person("foo", y, m, d);
11      } catch(e) {
12        console.log(e.name+":"+e.message);
13      } finally{
14        console.log("finally");
15      }
16    }
17  }
```

- テストを繰り返す関数 `test()` が定義されている。
- 2 行目では無限ループが定義されている。正しいパラメータが与えられたときに 7 行目で作成されたオブジェクトを戻り値にして関数の実行が終了する。
- `try{}` 内にはエラーが発生するかもしれないコードを中に含める。
  - 年、月、日の入力を `prompt` によるダイアログボックスから入力させている。
  - 戻り値は文字列なので、`Number` で数に直している。
  - 代入された値が正当であればその値をコンソールに出力している。
- 与えられた入力が正しくなければエラーが投げられ、`catch(e)` の中に制御が移る。
- `catch(e)` における `e` には発生したエラーオブジェクトが渡されるので、コンソールにその情報を出力する (12 行目)。

```
>p = test();
 年:2010
 月:4
 日:31
 Error:日が不正です
 finnally
 年:2010
 月:4
 日:30
finnally
Person {name: "foo", birthday: {…}, hometown: "神奈川"}
```

- 200 年 4 月 31 日は不正な日付なので、9 行目の `console.log()` の出力はない。
- その代り、エラーの内容が出力されている。
- また、エラー出力後は `finally` ブロックが実行されていることも分かる。
- 入力データが正しい場合でも `finally` ブロックが実行されている (下から 2 行目)。

第6回　正規表現

正規表現とは、一定の規則にあてはまる文字列のパターンを記述する文字列のことである。正則表現とも呼ばれる。JavaScript では RegExp クラスが正規表現を表す。

6.1　正規表現オブジェクトの記述方法

正規表現のオブジェクトは RegExp() コンストラクタで生成できるが、正規表現リテラルを使って記述することもできる。正規表現リテラルは文字列をスラッシュ(/) で囲んで記述する。

次の正規表現はどちらも s で始まる文字列を表す。これを s で始まる文字列にマッチするという。

```
let pattern = new RegExp("^s");
let pattern = /^s/;
```

リテラル文字　正規表現では次の文字は特別な意味を持つ。これらはメタ文字と呼ばれる。

```
^ $ . * + ? = ! | \ / ( ) [ ] { }
```

これらの文字をそのまま文字として使いたい場合はその前にバックスラッシュ(\) を付ける。英数字の前にバックスラッシュを付けると別な意味になる場合がある (表 6.1)。

表 6.1: 正規表現のリテラル文字

文字	意味
英数字	通常の文字
\0	NULL 文字 (\u0000)
\t	タブ (\u0009)
\n	改行 (\u000A)
\v	垂直タブ (\u000B)
\f	改ページ (\u000C)
\r	復帰 (\u000D)
\xnn	16 進数 nn で指定された ASCII 文字 (\x0A は\n と同じ)
\uxxxx	16 進数 xxxx で指定された Unicode 文字 (\u000D は\r と同じ)
\cX	制御文字 (\cC は\u0003 と同じ)

文字クラス　個々のリテラル文字を `[]` で囲むことでそれぞれの文字の一つにマッチする。このほかにもよく使われる文字クラスには特別なエスケープシークエンスがある (表 6.2)。

表 6.2: 文字クラス

文字	意味
`[...]`	`[]` 内の任意の 1 文字
`[^...]`	`[]` 内以外の任意の 1 文字
`.`	改行 (Unicode の行末文字) 以外の任意の 1 文字 `[^\n]` と同じ
`\w`	任意の単語文字。`[A-Za-z0-9]` と同じ
`\W`	任意の単語文字以外の文字。`[^A-Za-z0-9]` と同じ
`\s`	任意の Unicode 空白文字
`\S`	任意の Unicode 空白文字以外の文字
`\d`	任意の数字。`[0-9]` と同じ
`\D`	任意の数字以外の文字。`[^0-9]` と同じ
`\[\b]`	リテラルバックスペース

たとえば、 `\d\d` は 2 桁の 10 進数にマッチする。先頭に `0` が来てもよい。先頭に `0` が来る場合を除くのであれば `[1-9]\d` となる。一般には、同じパターンの繰り返しが必要になることが多い。

繰り返し　正規表現の文字の繰り返しを指定できる (表 6.3)。

表 6.3: 繰り返しの指定

文字	意味
`{m,n}`	直前の項目の `m` 回から `n` 回までの繰り返し
`{m,}`	直前の項目の `m` 回以上の繰り返し
`{m}`	直前の項目の `m` 回の繰り返し
`?`	直前の項目の 0 回 (なし) か 1 回の繰り返し。`{0,1}`と同じ
`+`	直前の項目の 1 回以上の繰り返し。`{1,}`と同じ
`*`	直前の項目の 0 回以上の繰り返し。`{0,}`と同じ

- 4 桁の 10 進数のパターンは`\d\d\d\d` で表されるが、繰り返しを使うと`\d{4}`で表される。
- 1 桁以上の 10 進数は`\d+`で表される。先頭が `0` でないようにすると、`[1-9]\d*`と表される。

非貪欲な繰り返し　通常、正規表現において繰り返しはできるだけ長く一致するように繰り返しが行われる。これを貪欲な繰り返しという。たとえば、`"aaaaab"`という文字列に対し、正規表現 `/a+/` がマッチする部分は `aaaaa` の長さ 5 の文字列になる (もっともここではこのパターンが含まれることしかわからない)。

これに対し、できるだけ短い文字列のマッチで済ませる繰り返しを、非貪欲な繰り返しという。これを指定するには繰り返し指定の後に?を付ける。つまり、??、+?、*?、{1,5}? などのように記述する。

したがって、"aaaaab"という文字列に対し、正規表現 /a+?/ がマッチする部分は a の長さ 1 の文字列になる。しかし、正規表現 /a+?b/ がマッチする部分は全体の"aaaaab" になる。これはマッチが開始する位置が、この文字列の先頭から始まるためである。詳しくは実行例 6.5を参照のこと。

選択、グループ化、参照 正規表現にはいくつかのパターンの選択や、表現のグループ化ができる。また、グループ化したところに一致した文字列を後で参照する (前方参照) ことができる (表 6.4)[1]。

表 6.4: 選択、グループ化、参照の指定

文字	意味
|	この記号の左右のどちらかを選択する
(...)	正規表現のグループ化をする。これにより、*,+,| などの対象がグループ化されたものになる。また、グループに一致した文字列を記憶して後で参照できる。
(?:..)	グループ化しか行わない。一致した文字列を記憶しない。
\n	グループ番号 n で指定された部分表現に一致する。グループ番号は左から数えた (の数である。ただし、(?は数えない。

JavaScript では文字列は"ではさむものか’ ではさむことになっている。グループ番号を使うと文字列のパターンのチェックができる。

実行例 6.1 選択、グループ化の例を挙げる。

- (J|j)ava(S|s)cript は JavaScript, Javascript, javaScript, javascript の 4 つにマッチする。
- [+-]?\d+ は符号つき (なくてもよい)10 進数とマッチする。[+-]?により、符号がなくてもよいことに注意すること。

一致位置の指定 正規表現には文字列が現れる位置を指定することができる (表 6.5)。

最後の 2 つは Java にはマッチさせたいが JavaScript にはマッチさせたくないときなどに使用できる。

フラグ 正規表現にはいくつかのフラグを指定できる (表 6.6)。フラグを書く位置は正規表現リテラルを表す/..../ の後に書く。

[1] 元来の正規表現では前方参照ができない。この機能は正則言語のより広い範囲の言語であることを示している

表 6.5: 一致位置の指定

文字	意味
^	文字列の先頭
$	文字列の最後
\b	単語境界。\w と\W の間の位置。[\b] との違いに注意
\B	単語境界以外
(?=p)	後に続く文字列が p に一致することが必要。
(?!p)	後に続く文字列が p に一致しないことが必要。

表 6.6: フラグの指定

文字	意味
i	大文字と小文字を区別しない
g	グローバル検索をする。初めに一致したものだけでなくすべてを検索する。
m	複数行モードにする。^は文字列の先頭だけでなく、行の先頭に。$ は文字列の末尾と行の末尾に一致する。

g のフラグはパターンマッチした部分文字列を置き換えるメソッド内でしか意味を持たない。たとえば、/javascript/ig である。これは JaVaSCRipt などにマッチする。

課題 6.1 次の文字列にマッチする正規表現を作れ。

1. C 言語の変数名の命名規則に合う文字列
2. 符号付小数。符号はなくてもよい。整数の場合は小数点はなくてもよい。また、小数点はあっても小数部はなくてもよい。整数部分には数字が少なくとも一つはあること。たとえば-1. にはマッチするが、-.0 には整数部分がないのでマッチしない。. のエスケープを忘れないようにすること。
3. 前問の正規表現を拡張して、指数部が付いた浮動小数にマッチするものを作れ。指数部は E または e で始まり、符号付き (なくてもよい) 整数とする。
4. 24 時間生の時刻の表し方。時、分、秒はすべて 2 桁とし、それらの区切りは:とする。たとえば午後 1 時 10 分 6 秒は 13:10:06 である。また、13:10:66 は秒数が 60 以上になっているのでマッチしてはいけない。
5. ファイルの拡張子が.html であるファイル名

6.2 文字列のパターンマッチングメソッド

表 6.7は正規表現による文字列のパターンマッチングが使用できる String オブジェクトのメソッドの一覧である。

表 6.7: 文字列で正規表現が使えるメソッド

文字	引数	意味
match()	正規表現	引数の正規表現にマッチした部分を文字列の配列で返す。見つからない場合は null が変える。g フラグがない場合の補足は下の例を参照。
replace()	正規表現､置換テキスト	g フラグがあれば一致した部分すべてを、ない場合は、はじめのところだけ置換した文字列を返す。 置換文字列の中でグループ化した部分文字列を$1,$2,... で参照できる。
search()	正規表現	正規表現に一致した位置を返す。見つからない場合は −1 を返す。g フラグは無視される。
split()	正規表現、分割最大数	正規表現のある位置で文字列を分割する。2 番目の引数はオプション

これらをまとめた実行例を示す。

実行例 6.2 4 桁の数字にマッチする正規表現オブジェクトを作成する。

```
let ex = /\d\d\d\d/;
```

この正規表現に対し、それぞれのメソッドを適用させる。

```
>"20144567".search(ex);          //戻り値は 0
>"20144567".match(ex);           //戻り値は ["2014"]
>"20144567".replace(ex,"AA");    //戻り値は "AA4567"
```

- 検索対象の文字列はすべて数字からなるのでこの正規表現にマッチする位置は 0 である。
- マッチした文字列は先頭から 4 文字となる。
- "AA"で置き換えると先頭の 4 文字が置き換えられる。

正規表現に g フラグをつけて同じようなことをする。

```
>let exg = /\d\d\d\d/g;
>"20144567".search(exg);       //戻り値は 0
>"20144567".match(exg);        //戻り値は ["2014", "4567"]
>"20144567".replace(exg,"AA");//戻り値は "AAAA"
```

g フラグがあるので match() や replace() が複数回実行されていることがわかる。

実行例 6.3 次の例はマッチした文字列を置換する。

```
>"aaa bbb".replace(/(\w*)\s*(\w*)/,"$2,$1"); //戻り値は "bbb,aaa"
```

英数字からなる 2 つの文字列 ((\w*)) の順序を入れ替えて、その間に , を挿入する。$1 は文字列 aaa に、$2 は文字列 bbb にマッチしている。

実行例 6.4 次の例は文中の Java を JavaScript に変える。ここでは JavaScript を JavaJavaScript にしないために (?!p) を用いる。

```
>"Java と JavaScript は全く違う言語です。".replace(/Java(?!Script)/,"JavaScript");
"JavaScript と JavaScript は全く違う言語です。"
```

実行例 6.5 貪欲さと非貪欲さの確認を行う。

```
>"aaaaab".match(/a+/);     //戻り値は ["aaaaa"]
>"aaaaab".match(/a+?/);    //戻り値は ["a"]
```

上の例は貪欲なので a の繰り返しの部分を最大限の位置でマッチしている。下の例は非貪欲なので最小限の長さの部分にしかマッチしていない。

```
>"aaaaab".match(/a+?b/);  //戻り値は ["aaaaab"]
```

初めに先頭の a にマッチしたので b が来るところまでマッチする。

実行例 6.6 次の例は前方参照を行っている。

```
>"abcdbcc".search(/((.)\2).*\1/);  //戻り値は -1
```

- (.) の部分は左かっこが 2 番目にあるので、\2 で参照できる。(.)\2 は同じ文字が 2 つ続く意味になる。この部分全体が再び () でくくられているので、その部分は\1 で参照できる。
- したがって、この正規表現は、同じ文字の繰り返しが 2 回現れる文字列にマッチする。
- 文字列"abcdbcc"には同じ文字が連続して現れるのが末尾の"cc"しかないので、この文字列にはマッチしない (戻り値が -1 である)。

```
>"abccbcc".search(/((.)\2).*\1/); //戻り値は 2
>"abccbcc".match(/((.)\2).*\1/);  //戻り値は ["ccbcc", "cc", "c"]
```

- この文字列では cc という同じ文字を繰り返した部分が 2 か所あるのでマッチする。はじめの cc の位置が先頭から 2 番目なので戻り値が 2 となっている。
- match() を行うと、cc ではさまれた部分文字列が戻り値の配列の先頭に、以下、\1 と\2 にマッチした部分文字列が配列に入る。

```
>"abccbcckkccaaMMaa".match(/((.)\2).*\1/);//戻り値は ["ccbcckkcc", "cc", "c"]
```

- この例では cc が部分文字列に 3 か所現れている。
- 貪欲なマッチのため、マッチした部分は 1 番目と 3 番目の cc にはさまれた部分である。g フラグがないので、aa ではさまれた部分文字列はマッチしない。
- 配列の残りの成分には\1 と\2 が入っている。

```
>"abccbcckkccaaMMaa".match(/((.)\2).*\1/g); //戻り値は ["ccbcckkcc", "aaMMaa"]
```

g フラグを付けるとマッチした部分文字列の配列が戻ってくるが、\1 などの情報は得られない。

```
"abccbcckkccaaMMaa".match(/((.)\2).*?\1/g);//戻り値は ["ccbcc", "aaMMaa"]
```

この例は非貪欲でグローバルなマッチである。非貪欲にすると一番目と 2 番目の cc にはさまれた部分と aa ではさまれた部分がそれぞれマッチする。

次の例は前方参照を利用してデータなどに含まれる文字列の引用符が対応している (シングルクオート同士、ダブルクオート同士) かを確認している。

```
>'\'abcd"df"\''.search(^/(['"]).*\1$/); //戻り値は 0
>'\'abcd"df"\''.match(/^(['"]).*\1$/);  //戻り値は ["'abcd"df"'", "'"]
```

- \' は文字列データとして与えられたことを仮定している。
- 文字クラス ['"] が () で囲まれているのでここで一致した文字が\1 で参照できる。
- したがって、文字列の先頭と最後に同じ引用符が来る場合に文字は見つかる。
- match() の戻り値は配列で、正規表現に g フラグがないときは、初めがマッチした文字列、以下順に\1,\2,.. にマッチした文字列が入っている。

実行例 6.7 文字列を指定した文字列で分割する split() の分割する文字列にも正規表現が使える。

```
>" 1, 2 , 3   ,  4".split(/\s*,\s*/);//戻り値は [" 1", "2", "3", "4"]
```

0 個以上の空白、,、0 個以上の空白の部分で分割している。1 の前にある空白が除去できていない。

```
>" 1, 2 , 3   ,  4".split(/\W+/);  //戻り値は ["", "1", "2", "3", "4"]
```

非単語文字の 1 個以上の並びで分割している。先頭の空白で分割されているので、分割された初めの文字列は空文字列""となっている。

```
>" 1, 2 , 3   ,  4".replace(/\s/,"").split(/\W+/);
  //戻り値は ["1", "2", "3", "4"]
```

先頭の分割文字列が空文字になるのを防ぐために、初めに空白文字を空文字に置き換えて (取り除いて) いる。その文字列に対し非単語文字列で分割しているので空文字が分割結果に表れない。

この様に文字列に対してメソッドを順に続けて記述することができる。

課題 6.2 次の実行結果がどうなるか答えよ。理由も述べること。

1. `"aaaabaaabb".match(/.*b/);`
2. `"aaaabaaabb".match(/.*b/g);`
3. `"aaaabaaabb".match(/.*?b/);`
4. `"aaaabaaabb".match(/.*?b/g);`
5. `"abccbcckkccaaMMaacc".match(/((.)\2).*\1/);`
6. `"abccbcckkccaaMMaacc".match(/((.)\2).*\1/g);`
7. `"abccbcckkccaaMMaacc".match(/((.)\2).*?\1/);`
8. `"abccbcckkccaaMMaacc".match(/((.)\2).*?\1/g);`
9. `"abccbcckkccaaMMaa".match(/((.)\2).*\1/);`
10. `"abccbcckkccaaMMaa".match(/((.)\2).*\1/g);`
11. `"abccbcckkccaaMMccaa".match(/((.)\2).*\1/g);`
12. `"abccbcckkccaaMMccaa".match(/((.)\2).*?\1/g);`

課題 6.3 実行例 5.8 において `prompt()` の戻り値は文字列である。これを利用して入力値を正規表現を用いてチェックするように直せ。

第7回　DOMの利用

この節からは HTML 文書の取り扱いを始める。この授業では 2014 年 10 月 28 日に W3C の Recommendation となった HTML5 を用いる[1]。

7.1　HTML 文書の構成

実行例 7.1 次のリストは Google Maps を利用して地図を表示するものである[2]

```
 1 <!DOCTYPE html>
 2 <html>
 3 <head>
 4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
 5 <title>初めての GoogleMaps</title>
 6 <script type="text/javascript"
 7           src="http://maps.google.com/maps/api/js?sensor=false"></script>
 8 <script type="text/javascript">
 9 window.onload = function() {
10   let latlng = new google.maps.LatLng(35.486210,139.341443);
11   let myOptions = {
12     zoom: 10,
13     center: latlng,
14     mapTypeId: google.maps.MapTypeId.ROADMAP
15   };
16   let mapCanvas = document.getElementById("map_canvas")
17   let map = new google.maps.Map(mapCanvas, myOptions);
18 }
19 </script>
20 <link rel="stylesheet" type="text/css" href="map.css" />
21 </head>
22 <body>
23   <div id="map_canvas" ></div>
```

[1] http://www.w3.org/TR/2014/REC-html5-20141028/

[2] Google Maps API3 の公開当初は API キーは存在しなかったが、その後 API キーが必要となった。2018 年 7 月に API 利用のルールが変更され、API の利用が有償となった。一定の利用以下であれば無料であるが、支払方法の登録は必要である。

```
24 </body>
25 </html>
```

ここではこの HTML 文書の構成について解説をする。

- 1 行目は HTML 文書の DOCTYPE 宣言である。この形は HTML5 におけるもので、以前のものと比べて記述が簡単になっている。
- 2 行目はこの HTML 文書のルート要素と呼ばれるものである。最後の 25 行目の </html>までが有効となる。すべての要素はこの範囲になければならない。
- 3 行目から始まる<head>はブラウザに表示されない、いろいろな HTML 文書の情報を表す。
  - 4 行目はこの文書の形式や文字集合を記述している。ここでは内容は text/html の形式、つまり、テキストで書かれた html の形式で書かれていることを表す。[3]
  - 5 行目の<title>はブラウザのタブに表示される文字列を指定している。
  - 6 行目から 7 行目は Google Maps のライブラリーを読み込むためのものである。このように JavaScript のプログラムは外部ファイルとすることができる。
  - 8 行目から 19 行目は HTML 文書内に書かれた JavaScript である。詳しい解説は後の授業で行う。
  - 20 行目は HTML 文書の見栄えなどを規定する CSS ファイルを外部から読み込むことをしている。
- HTML 文書で実際にブラウザ内で表示される情報は<body>要素内に記述する。
- このリストでは Google Maps を表示するための 要素が一つあるだけである。このとき、 は<body>の子要素であるといい、<body>は の親要素という。
- 各要素名または要素の終了を示すタグ (<...>) の間に文字列がある場合、その部分はテキストノードと呼ばれるノードが作成されている。

各要素は<との中に現れる。初めに現れる文字列が要素名であり、そのあとに属性と属性値がいくつか並ぶ。

- 属性とその属性値は=で結ばれる。
- 属性値は ” ではさまれた文字列として記述する。
- 6 行目の<script>要素では属性 type と src が設定されている。
- 23 行目の 要素では属性 id に属性値 map_canvas を設定している。なお、この要素は CSS によっても属性が定義されている。

次のリストは 20 行目で参照している CSS ファイルの内容である。

[3] このような方法でファイルのデータ形式を表すことを MIME(Multipurpose Internet Mail Extension) タイプと呼ぶ。元来、テキストデータしか扱えない電子メールに様々なフォーマットのデータを扱えるようにする規格である。

```
1 #map_canvas{
2     width:500px;
3     height:500px;
4     float:left;
5     margin:5px 10px 5px 10px;
6 }
```

- CSS の各構成要素は HTML 文書の要素を選択するセレクタ (ここでは`#map_canvas`) とそれに対する属性値の並び ([属性]:[属性値];) からなる。
- `#`で始まるセレクタはそのあとの文字列を`<id>`の属性値に持つ要素に適用される。
- したがって、ここの規則は 23 行目の`<div>`要素に適用される。
- その内容は Google Maps が表示される画面の大きさ (`width` と `height`)、配置の位置 (`float`) と要素の外に配置される空白 (`margin`) を指定している。

7.2 CSS の利用

カスケーディングスタイルシート (CSS) は HTML 文書の要素の表示方法を指定するものである。CSS は JavaScript からも制御できる。

文書のある要素に適用されるスタイルルールは、複数の異なるルールを結合 (カスケード) したものである。スタイルを適用するためには要素を選択するセレクタで選ぶ。

表 7.1は CSS3 におけるセレクタを記述したものである[4]。

表 7.1: CSS3 のセレクタ

セレクタ	解説
*	任意の要素
E	タイプが E の要素
E[foo]	タイプが E で属性 "foo" を持つ要素
E[foo="bar"]	タイプが E で属性 "foo" の属性値が"bar"である要素
E[foo~="bar"]	タイプが E で属性 "foo" の属性値がスペースで区切られたリストでその一つが "bar"である要素
E[foo^="bar"]	タイプが E で属性 "foo" の属性値が"bar"で始まる要素
E[foo$="bar"]	タイプが E で属性 "foo" の属性値が"bar"で終わる要素
E[foo*="bar"]	タイプが E で属性 "foo" の属性値が"bar"を含む要素
E[foo|="en"]	タイプが E で属性 "foo" の属性値がハイフンで区切られたリストでその一つが "en"で始まる要素

次ページへ続く

[4] http://www.w3.org/TR/selectors/より引用。

表 7.1: CSS3 のセレクタ (続き)

セレクタ	解説
E:root	document のルート要素
E:nth-child(n)	親から見て n 番目の要素
E:nth-last-child(n)	親から見て最後から数えて n 番目の要素
E:nth-of-type(n)	そのタイプの n 番目の要素
E:nth-last-of-type(n)	そのタイプの最後から n 番目の要素
E:first-child	親から見て一番初めの子要素
E:last-child	親から見て一番最後の子要素
E:first-of-type	親から見て初めてのタイプである要素
E:last-of-type	親から見て最後のタイプである要素
E:only-child	親から見てただ一つしかない子要素
E:only-of-type	親から見てただ一つしかないタイプの要素
E:empty	テキストノードを含めて子要素がない要素
E:link, E:visited	まだ訪れたことがない (:link) か訪れたことがある (visited) ハイパーリンクのアンカーである要素
E:active, E:hover, E:focus	ユーザーに操作されている状態中の要素
E:target	参照 URI のターゲットである要素
E:enabled, E:disabled	使用可能 (:enable) か使用不可のユーザーインターフェイスの要素
E:checked	チェックされているユーザーインターフェイスの要素
E::first-line	要素のフォーマットされたはじめの行
E::first-letter	要素のフォーマットされたはじめの行
E::before	要素の前に生成されたコンテント
E::after	要素の後に生成されたコンテント
E.warning	属性 class が ”warning” である要素
E#myid	属性 id の属性値が ”myid” である要素
E:not(s)	単純なセレクタ s にマッチしない要素
E F	要素 E の子孫である要素 F
E > F	要素 E の子である要素 F
E F+	要素 E の直後にある要素 F
E ~ F	要素 E の直前にある要素 F

いくつか注意する点を挙げる。

- 属性 id の属性値の前に#をつけることでその要素が選ばれる。
- 属性 class の属性値の前に . をつけることでその要素が選ばれる。
- nth-child(n) には単純な式を書くことができる。詳しくは実行例 7.1を参照のこと。このセ

レクタは複数書いてもよい。

- E F と E > F の違いを理解しておくこと。たとえば div div というセレクタは途中に別の要素が挟まれていてもよい。また、 要素が 3 つ入れ子である場合にはどのような 2 つの組み合わせも対象となる。

課題 7.1 次の HTML 文書において以下の問いに答えなさい。

```
 1 <!DOCTYPE html>
 2 <html>
 3 <head>
 4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
 5 <title>nth-child のチェック</title>
 6 <style type="text/css" >
 7 li:nth-child(n){
 8   background:yellow;
 9 }
10 </style>
11 </head>
12 <body>
13   <ol>
14     <li>1 番目</li>
15     <li>2 番目</li>
16     <li>3 番目</li>
17     <li>4 番目</li>
18     <li>5 番目</li>
19     <li>6 番目</li>
20   </ol>
21 </body>
22 </html>
```

ここで<ol>は箇条書きの開始を示す要素であり、<li>は箇条書きの各項目を示す要素である。

1. nth-child の () 内に次の式を入れたとき、各項目の背景色がどうなるか報告しなさい。
   (a) n(ここでのリストの設定)
   (b) 2n
   (c) n+3
   (d) -n+2
2. 背景色が次のようになるようにそれぞれ CSS を設定しなさい。
   (a) 偶数番目が黄色、奇数番目がオレンジ色

(b) 1 番目、4 番目、... のように 3 で割ったとき、1 余る位置が明るい灰色

(c) 4 番目以下がピンク

(d) 下から 2 番目以下が緑色

7.3　DOM とは

Document Object Model(DOM) は HTML 文書などの要素をノードとしたツリー構造で管理する方法である。DOM のメソッドやプロパティを使うことで各要素にアクセスしたり、属性値やツリーの構造を変化させることができる。DOM の構造は開発者ツールなどで見ることができる。

- Google Chrome では開発者ツールの Elements タブで確認できる。要素上で右クリックして「Edit as HTML」を選択するとテキストとして編集できる。
- FireFox では開発ツールの「インスペクタ」タブで確認できる。要素上で右クリックから「HTML として編集」とするとテキストとして編集できる。

7.4　DOM のメソッドとプロパティ

DOM では DOM ツリーを操作するメソッドやプロパティが規定されている。これらの手段を用ると DOM の構造を変化させて、結果として HTML 文書では表示の変更が可能となる。

7.4.1　DOM のメソッド

表 7.2は DOM のメソッドのリストである。`document` だけに適用できるものと、すべての要素に適応できるものとがある。一番右の項目「PHP」は PHP の `DOMDocument` クラスにおけるサポート状態を示す。◯ はサポートされていて × はサポートされていないことを示す。なお、結果が要素のリストであるものについては通常の配列と同様に `[ ]` によりそれぞれの要素を指定できる。

なお、表中の名前空間 (Namespace) とは、指定した要素が定義されている規格を指定するものである。一つの文書内で複数の規格を使用する場合、作成する要素がどこで定義されているのかを指定する。これにより、異なる規格で同じ要素名が定義されていてもそれらを区別することが可能となる。ルート要素 (HTML 文書では `html`) が通常は指定される。HTML 文書では `http://www.w3.org/1999/xhtml` を指定する[5]。

[5] `http://dev.w3.org/html5/spec-LC/namespaces.html`

表 7.2: DOM のメソッド

メソッド名	対象要素	説　　明	PHP
getElementById(id)	document	属性 id の値が id の要素を得る。	○
getElementsByTagName(Name)	対象要素	要素名が Name の要素のリストを得る。	○
getElementsByClassName(Name)	対象要素	属性 class の値が Name の要素のリストを得る。	×
getElementsByName(Name)	document	属性 name が Name である要素のリストを得る。	×
querySelector(selectors)	対象要素	selectors で指定された CSS のセレクタに該当する一番初めの要素を得る。	×
querySelectorAll(selectors)	対象要素	selectors で指定された CSS のセレクタに該当する要素のリストを得る。	×
getAttribute(Attrib)	対象要素	対象要素の属性 Attrib の値を読み出す。得られる値はすべて文字列である。	○
setAttribute(Attrib,Val)	対象要素	対象要素の属性 Attrib の値を Val にする。数を渡しても文字列に変換される。	○
hasAttribute(Attrib)	対象要素	対象要素に属性 Attrib がある場合は true を、ない場合は false を返す。	○
removeAttribute(Attrib)	対象要素	対象要素の属性 Attrib を削除する。	○
createElement(Name)	document	Name で指定した要素を作成する。	○
createElementNS(NS,Name)	document	名前空間 NS にある要素 Name を作成する。	○
createTextNode(text)	document	text を持つテキストノードを作成する。	○
cloneNode(bool)	対象要素	bool が true のときは対象要素の子要素すべてを、false のときは対象要素だけの複製を作る。	○
appendChild(Elm)	対象要素	Elm を対象要素の最後の子要素として付け加える。Elm がすでに対称要素の子要素のときは最後の位置に移動する。	○
insertBefore(newElm, PElm)	対象要素	対象要素の子要素 PElm の前に newElm を子要素として付け加える。Elm がすでに対称要素の子要素のときは指定された位置に移動する。	○
removeChild(Elm)	対象要素	対象要素の子要素 Elm を取り除く。	○
replaceChild(NewElm, OldElm)	対象要素	対象要素の子要素 OldElm を NewElm で置き換える。	○

実行例 7.2 次の例は 1 月から 12 月までを選択できるプルダウンメニューを作成するものである。この様なプルダウンメニューをテキストエディタで入力すると次のようになる。

```
1  <!DOCTYPE html>
2  <head>
3  <meta charset="UTF-8"/>
4  <title>プルダウンメニュー</title>
```

```
 5  </head>
 6  <body>
 7    <form id="menu">
 8      <select>
 9        <option value="1">1 月</option>
10        <option value="2">2 月</option>
11        <option value="3">3 月</option>
12        <option value="4">4 月</option>
13        <option value="5">5 月</option>
14        <option value="6">6 月</option>
15        <option value="7">7 月</option>
16        <option value="8">8 月</option>
17        <option value="9">9 月</option>
18        <option value="10">10 月</option>
19        <option value="11">11 月</option>
20        <option value="12">12 月</option>
21      </select>
22    </form>
23  </body>
24  </html>
```

- ユーザからの入力を受け付ける要素は通常、<form>要素内に記述する (7 行目)。
- プルダウンメニュー は8行目の<select>で作成している。選択する内容はその子要素の<option>要素で指定する。
- <option>要素の属性 value の値が選択した値として利用できる。
- <option>要素内の文字列 (テキストノード) がプルダウンメニューに表示されるものになる。
- <select>は<form>の子要素であり、各<option>は<select>の子要素となっている。

この HTML 文書では option 要素が数字の連番からなるので、プログラムで作成することも可能である。次のリストは JavaScript でファイルのロード後に option 要素を作成するものである。

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
5 <title>プルダウンメニューの作成</title>
6 <script type="text/ecmascript">
7 //<![CDATA[
8   window.onload = function(){
```

```
 9     let Select = document.createElement("select");
10     document.getElementById("menu").appendChild(Select);
11     for(let i=1;i<=12;i++) {
12       let Option = document.createElement("option");
13       Option.setAttribute("value",i);
14       Select.appendChild(Option);
15       Option.appendChild(document.createTextNode(i+"月"));
16     }
17   }
18 //]]>
19 </script>
20 </head>
21 <body>
22   <form id="menu"></form>
23 </body>
24 </html>
```

- 8 行目の window.onload はファイルのロードが終わった後に発生するイベントの処理をする関数を表す。イベントの詳細については後で解説する。
- 10 行目では<select>要素を作成している。
- 11 行目では 10 行目で作成した<select>要素を getElementById("menu") で得た<form>要素の子要素に設定している。
- 12 行目からの for ループで 12 個の<option>要素を作成し、<select>要素の子要素としている。
  - 13 行目で<option>要素を新規に作成し、14 行目で、その要素の属性 value に値を設定している。
  - 15 行目ではその<option>要素を<select>要素の子要素としている。
  - さらに、15 行目では表示する文字列をもつテキストノードを作成し、16 行目でそれを<option>要素の子要素としている。

課題 7.2 実行例 7.2の JavaScript でプルダウンメニューを作成するリストに対して次の問に答えなさい。

1. ブラウザの「ページのソースを表示」を選択して、ソースコードがどのようになっているか確認する。
2. DOM ツリーが実行例 7.2のリストと同じようにできていることを確認する。
3. <select>要素を構成する部分を関数化して、月だけでなく日 (1 日から 31 日) が選べるプルダウンメニューも作成できるようにしなさい。関数の引数を工夫すること。

7.4.2　DOM の要素のプロパティ

表 7.3は DOM の要素に対するプロパティである。最後の「PHP」の項目は表 7.2と同様の意味である。

表 7.3: DOM 要素に対するプロパティ

プロパティ名	説　　明	PHP
firstChild	指定された要素の先頭にある子要素	○
lastChild	指定された要素の最後にある子要素	○
nextSibling	指定された子要素の次の要素	○
previousSibling	現在の子要素の前にある要素	○
parentNode	現在の要素の親要素	○
hasChildNodes	その要素が子要素を持つ場合は true 持たない場合は false である。	○
nodeName	その要素の要素名前	○
nodeType	要素の種類 (1 は普通の要素、3 はテキストノード)	○
nodeValue	(テキスト) ノードの値	○
childNodes	子要素の配列	○
children	子要素のうち通常の要素だけからなる要素の配列	×
firstElementChild	指定された要素の先頭にある通常の要素である子要素	×
lastElementChild	指定された要素の最後にある通常の要素である子要素	×
nextElementSibling	指定された子要素の次の通常の要素	×
previousElementSibling	現在の子要素の前にある通常の要素	×

- これらのプロパティのうち、nodeValue を除いてはすべて、読み取り専用である。
- ある要素に子要素がない場合にはその要素の firstChild や lastChild は null となる。
- ある要素に子要素がある場合、その firstChild.previousSibling や lastChild.nextSibling も null となる。firstElementChild などでも同様である。

プロパティに関する例はイベント処理のところで示す。

課題 7.3 65ページから始まるリストと 66ページから始まるリストにおける<select>要素の子要素の数を調べよ。異なる場合にはその理由を述べよ。

document.getElementsByTagName("select")[0].childNodes.length を使うかコンソールに出力して調べるのもよい。また、document.getElementsByTagName("select")[0].childNodes[0].nodeType も調べるとよいかもしれない。また、childNodes の代わりに children を用いたらどうなるかも調べるとよい。

第8回　イベント

8.1　イベント概説

イベントとはプログラムに対して働きかける動作を意味する。Windows で動くプログラムにはマウスボタンが押された (クリック) などのユーザからの要求、一定時間経ったことをシステムから通知される (タイマーイベント) などありとあらゆる行為がイベントという概念で処理される。プログラムが開始されるということ自体イベントである。このイベントはプログラムの初期化をするために利用される。

イベントの発生する順序はあらかじめ決まっていないので、それぞれのイベントを処理するプログラムは独立している必要がある。このようにイベントの発生を順次処理していくプログラムのモデルをイベントドリブンなプログラムという。

8.2　イベント処理の方法

イベントは各要素内で発生するので、イベントを処理する関数を適当な要素に登録する。この関数をイベントハンドラーという。イベントハンドラーの登録には表 8.1にある属性に関数を登録する方法と、メソッドを用いて要素に登録する方法がある。最近の傾向としては、HTML 文書には表示するものだけを記述し、イベントハンドラーなどの JavaScript のプログラムに関することは書かないということがある。この授業ではメソッドを通じて登録する方法だけを紹介する。

要素にイベントハンドラーを登録するメソッドは次の `document` オブジェクトのメソッドを使う。

`addEventListner(string` *type* `,functiion` *listener* `,[boolean` *useCapture* `])`

引数の意味は次のとおりである。

- *type* はイベントの種類を示す文字列であり、表 8.1におけるイベントの属性名から `on` を取り除いたもの
- *listener* はイベントを処理する関数 (イベントハンドラー)
- *useCapture* はイベントの処理順序 (詳しくは後述)

要素からイベントハンドラーを削除する `document` オブジェクトのメソッドは次のとおりである。

`removeEventListener(string` *type* `,functiion` *listner* `,[boolean` *useCapture* `])`

イベントハンドラーが削除されるためには登録したときと同じ条件を指定する必要がある。

通常、イベントハンドラーの関数は通常、イベントオブジェクトを引数に取るように定義する。

8.3　DOM Level 2 のイベント処理モデル

DOM Level 2 のイベント処理モデルでは、あるオブジェクトの上でイベントが発生すると次の順序で各オブジェクトにイベントの発生が伝えられる。

1. 発生したオブジェクトを含む最上位のオブジェクトにイベントの発生が伝えられる。このオブジェクトにイベント処理関数が定義されていなければなにも起きない。
2. 以下順に DOM ツリーに沿ってイベントが発生したオブジェクトの途中にあるオブジェクトにイベントの発生が伝えられる (イベントキャプチャリング)。
3. イベントが発生したオブジェクトまでイベントの発生が伝えられる。
4. その後、DOM ツリーに沿ってこのオブジェクトを含む最上位のオブジェクトまで再びイベントの発生が伝えられる (イベントバブリング)。

DOM Level 2 のモデルではイベントが発生したオブジェクトは渡されたイベントオブジェクトの `target` プロパティで、イベントを処理している関数が登録されたオブジェクトは `currentTarget` で参照できる。また、イベントが発生したオブジェクトにイベントハンドラーが登録されていなくてもその親やその上の祖先にイベントハンドラーが登録されていると、イベントが発生する。

`addEventListner()` の 3 番目の引数を `false` にするとイベント処理はイベントバブリングの段階で、`true` にするとイベント処理はイベントキャプチャリングの段階で呼び出される。このイベントの伝播を途中で中断させるメソッドが `stopPropagation()` である。

通常、ブラウザの画面で右クリックをすると、ページのコンテキストメニュが表示される。このようなデフォルトの操作を行わせないようにする方法が `preventDefault()` である。

8.4　HTML における代表的なイベント

8.4.1　ドキュメントの `onload` イベント

Web ブラウザは HTML 文書を次のような順序で解釈し、実行する[1]。

1. 初めに `document` オブジェクトを作成し、Web ページの解釈を行う。
2. HTML 要素やテキストコンテンツを解釈しながら、要素やテキスト (ノード) を追加する。この時点では、`document.readystate` プロパティは`"loading"`という値を持つ。
3. `<script>`要素が来ると、このこの要素を `document` に追加し、スクリプトを実行する。したがって、この時点で存在しない要素に対してアクセスすることはできない。つまり、`<script>`要素より後に書かれている要素のはアクセスできない。したがって、この段階では後で利用するための関数の定義やイベントハンドラーの登録だけをするのがふつうである。
4. ドキュメントが完全に解釈できたら、`document.readystate` プロパティは`"interactive"`という値に変わる。

[1] ここの説明は少し簡略化している。詳しくは [2](1ページ) の 13.3 節を参照のこと

5. ブラウザはドキュメントオブジェクトに対し DOMContentLoaded イベントを発生させる。この時点では画像などの追加コンテンツの読み込みは終了していない可能性がある。

6. それらのことが終了すると、document.readyState プロパティの値が"complete"となり、window オブジェクトに対して load イベントを発生させる。

8.4.2 マウスイベント

表 8.1 は HTML 文書で発生するマウスイベントを記述したものである。

表 8.1: マウスイベントの例

イベントの発生条件	イベントの属性名
ボタンがクリックされた	onclick
ボタンが押された	onmousedown
マウスカーソルが移動した	onmousemove
マウスボタンが離された	onmouseup
マウスカーソルが範囲に入った	onmouseover
マウスカーソルが範囲から出た	onmouseout

表 8.2はマウスイベントのプロパティを表したものである。

表 8.2: マウスイベントのプロパティ

プロパティ	型	意味
target	EventTarget	イベントが発生したオブジェクト
currentTarget	EventTarget	イベントハンドラーが登録されているオブジェクト
screenX	long	マウスポインタのディスプレイ画面における x 座標
screenY	long	マウスポインタのディスプレイ画面における y 座標
clientX	long	マウスポインタのクライアント領域における相対的な x 座標 (スクロールしている場合には window.pageXOffset を加える)
clientY	long	マウスポインタのクライアント領域における相対的な y 座標 (スクロールしている場合には window.pageYOffset を加える)
pageX	long	マウスポインタの document 領域における相対的な x 座標
pageY	long	マウスポインタの document 領域における相対的な y 座標
ctrlKey	boolean	cntrl キーが押されているか
shiftKey	boolean	shift キーが押されているか

次ページへ続く

表 8.2: マウスイベントのプロパティ(続き)

プロパティ	型	意味
altKey	boolean	alt キーが押されているか
metaKey	boolean	meta キーが押されているか
button	unsigned short	マウスボタンの種類、0 は左ボタン、1 は中ボタン、2 は右ボタンを表す。
eventPhase	unsigned short	イベント伝播の現在の段階を表す。次の値がある。Event.CAPUTURING_PHASE(1)、Event.AT_TARGET(2)、Event.BUBBLING_PHASE(3)

8.5　イベント処理の例

実行例 8.1 次の例は、form 要素内でユーザの入力を取り扱う代表的な input 要素の処理方法を示したものである。

このページでは大きさが指定された div 要素が 3 つ並び、その横にプルダウンメニュー、ラジオボタン、テキスト入力エリアと設定ボタンが並んでいる。

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8"/>
<title>イベント処理</title>
<link rel="stylesheet" type="text/css" href="event.css"/>
<script type="text/ecmascript" src="event.js"></script>
</head>
<body>
  <div class="block" id="Squares">
    <div></div><div></div><div></div>
  </div>
  <form>
    <select id="select">
      <option value="red">赤</option>
      <option value="yellow">黄色</option>
      <option value="blue">青</option>
    </select>
    <div id="radio">
      <div><input type="radio" value="red" name="color">赤</input></div>
      <div><input type="radio" value="yellow" name="color">黄</input></div>
      <div><input type="radio" value="blue" name="color">青</input></div>
    </div>
```

```
23     <div>
24       色名<input type="text" size="10" id="colorName"></input>
25       <input type="button" value="設定" id="Set"></input>
26     </div>
27     <div id="position">
28       <div>クリック位置</div>
29       <div><div>clientX</div><input type="text" size="3" class="click">
30            <div>clientY</div><input type="text" size="3" class="click"></div>
31       <div><div>pageX</div><input type="text" size="3" class="click">
32            <div>pageY</div><input type="text" size="3" class="click"></div>
33       <div><div>pageXOffset</div><input type="text" size="3" class="click">
34            <div>pageYOffset</div><input type="text" size="3" class="click"></div>
35      <div>from Boundary</div>
36       <div><div> Left</div><input type="text" size="3" class="click">
37            <div>Top</div><input type="text" size="3" class="click"></div>
38       </div>
39   </form>
40 </body>
41 </html>
```

このリストではCSSファイル、JavaScriptファイルが外部ファイルとなっている(5行目と6行目)。

このページでは左にある3つの領域でマウスをクリックすると、クリックした位置の情報が右側下方の8つのテキストボックスに表示される。

- 上の6つはイベントオブジェクトのプロパティをそのまま表示している(表7.3参照)。
- 最下部の値はそれぞれの領域から見た相対位置を表している。

プルダウンメニュー、ラジオボタンで選択された色、テキストボックスでは最後にクリックされた領域の背景色を変えることができる。プルダウンメニュー、ラジオボタンでは値に変化があったときに設定が行われる。テキストボックスではCSS3で定義された色の表示形式が使用できる。設定するためには隣の「設定」ボタンを押す必要がある。

次のリストは上のHTMLファイルから読み込まれるCSSファイルである。

```
1 .block {
2   display : inline-block;
3   vertical-align : middle;
4 }
5 .block > div {
6   width : 200px;
7   height : 200px;
8   display : inline-block;
9 }
```

```
10 #radio {
11   width:80px;
12 }
13 .click {
14   text-align:right;
15 }
16 form {
17   display : inline-block;
18   vertical-align : middle;
19 }
20 input, select{
21   font-size:25px;
22 }
23 #position > div {
24   text-align:center;
25   font-size: 20px;
26 }
27 #position > div > div {
28   display:inline-block;
29   width: 90px;
30   text-align: right;
31   font-size: 18px;
32 }
```

このスタイルシートは各要素の大きさや配置について指定している。

- 左側の 3 つの領域は id が block である要素の直下の div 要素であることから 5 行目のセレクタが適用される。
  - これらの 3 つの部分は大きさが 200px の正方形となっている (6 行目と 7 行目)。
  - 横に並べるようにするため、display を inline-block に指定している。
  - 背景色 (background) はプログラムから設定される。
- これらの領域全体を囲む div の垂直方向の配置の位置が 3 行目で定められている。
- 右上方のプルダウンメニューのフォントの大きさは 20 行目からのセレクタで定義されているが、文書のロード時に 30px に変更している。
- ラジオボタンの要素の幅は 10 行目からのセレクタで定められている。
- クリックしたときの位置情報を示す部分の CSS は 23 行目以下で定められている。
  - テキストボックスがない行は 23 行目から 26 行目が適用されている。

– テキストボックスがある行は、テキストボックスの位置をそろえるために、その前の文字列を div 要素の中に入れ、幅 (29 行目)、テキストの配置位置 (30 行目) とフォントの大きさ (31 行目) を指定している。

```
 1 window.onload = function() {
 2   let Squares   = document.getElementById("Squares");
 3   let Select    = document.getElementById("select");
 4   let ColorName = document.getElementById("colorName");
 5   let Radio     = document.getElementById("radio");
 6   let Set       = document.getElementById("Set");
 7   let Inputs    = document.getElementsByClassName("click");
 8   let lastClicked = Squares.children[1];
```

ここでは id 属性を持つ要素すべてを得ている。また、8 行目で、最後にクリックされた正方形のオブジェクトを記憶する変数を初期化している。

```
 9   Squares.children[0].style.background = "red";
10   Squares.children[1].style.background = "yellow";
11   Squares.children[2].style.background = "blue";
12   Select.style.fontSize = "30px";
```

- id が Squares である div 要素の下には 3 つの div 要素がある。これらの要素の背景色を指定するために、children プロパティを用いて参照している。。
- スタイルを変更するためには style プロパティの後に属性を付ける。
- 9 行目から 11 行目では左の 3 つの領域の背景色 (background) を設定している。
- フォントの大きさを指定する CSS 属性は font-size であるが、これは - を含んでいるのでそのままでは JavaScript の属性にならない。このような属性は-を省き、次の単語を大文字で始めるという規約がある。したがって、この場合には fontSize となる。

次のリストは正方形の領域がクリックされたときのイベントハンドラーを定義している部分である。イベントハンドラーは 3 つの正方形を含んだ div 要素につけている。

```
13   Squares.addEventListener("click",function(E){
14     Inputs[0].value = E.clientX;
15     Inputs[1].value = E.clientY;
16     Inputs[2].value = E.pageX;
17     Inputs[3].value = E.pageY;
18     Inputs[4].value = window.pageXOffset;
19     Inputs[5].value = window.pageYOffset;
20     let R = E.target.getBoundingClientRect();
21     Inputs[6].value = E.pageX-R.left;
```

```
22     Inputs[7].value = E.pageY-R.top;
23     ColorName.value = E.target.style.background;
24     lastClicked = E.target;
25   },false);
```

- 表示するためのテキストボックスのリストは 7 行目で得ている。
- 14 行目から 19 行目でそれぞれのテキストボックスに該当する値を入れている。
- 21 行目と 22 行目では該当する領域からの相対位置を求めている。そのためにはそれぞれの領域がページの先頭からどれだけ離れているかを知る必要がある。その情報を得るメソッドが `getBoundingClientRect()` である。

  このメソッドは次のようなプロパティを持つ `ClientRect` オブジェクトを返す。

プロパティ	解説
`top`	領域の上端の Y 座標
`bottom`	領域の下端の Y 座標
`left`	領域の左端の X 座標
`right`	領域の右端の X 座標
`width`	領域の幅
`height`	領域の高さ

- これらの値とイベントが起きた位置の情報から領域内での位置が計算できる。
- 24 行目ではクリックした正方形のオブジェクトを更新している。

ここでは 3 つの入力方法に対して、イベントハンドラーを定義している。

```
26   Select.addEventListener("change", function(){
27       lastClicked.style.background = Select.value; },false);
28   Radio.addEventListener("click", function(E){
29       alert(E.target.tagName);
30       if(E.target.tagName === "DIV") {
31         E.target.firstChild.checked = true;
32       }
33       console.log("----" + Radio.value);
34       lastClicked.style.background = Radio.querySelector("input:checked").value;
35     },false);
36   Set.addEventListener("click", function(){
37       lastClicked.style.background = ColorName.value; }, false);
38 }
```

- 26 行目から 27 行目ではプルダウンメニューに `change` イベントのハンドラーを定義している。直前にクリックされた部分の背景色を変えている。

- 28 行目から 34 行目はラジオボタンのあるところがクリックされたときのイベントハンドラーを設定している。ラジオボタンは name 属性が同じ値のものが一つのものとして扱われる (どこか一つだけオンになる)。
  - 29 行目ではクリックされた場所の要素名を表示している (通常は必要ない)。HTML の要素名は小文字で書かれていても大文字に変換されることが確認できる。
  - HTML のリストの 28 行目にある div 要素にクリックのイベントハンドラーを登録する (28 行目から 29 行目) ので、クリックされた場所がラジオボタンの上でなくても、この範囲内であればイベントは発生する。
  - クリックされた場所がラジオボタンの上でなければ (ラジオボタンの親要素の div 要素が E.target となる) のでその firstChild が値を変えるラジオボタンになる (31 行目)。
  - ラジオボタンの集まりには value プロパティがない。チェックされている場所を探すために querySelector("input:checked") を用いて、チェックされている要素を探す (33 行目の右辺)。
  - 色の設定はテキストボックスの値を代入する (36 行目)。

課題 8.1 次のようなオブジェクトをもとにプルダウンメニューを作成する JavaScript の関数を作成しなさい。

関数に渡すオブジェクト:`{"red": "赤", "orange":"橙", "yellow":"黄色", "green":"緑"}`
作成される DOM

```
<select>
  <option value="red">赤</option>
  <option value="orange">橙</option>
  <option value="yellow">黄色</option>
  <option value="green">緑</option>
</select>
```

また、18 行目から 22 行目にあるようなラジオボタンを作成する関数も作成しなさい。

実行例 8.2 次の例は 3 つのプルダウンメニューが順に年、月、日を選択できるものである。年や月が変化 (change イベントが発生) すると日のプルダウンメニューの日付のメニューがその年月にある日までに変わるようになっている。

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
5 <title>日付</title>
6 <script type="text/ecmascript">
7 //<![CDATA[
8   window.onload = function(){
```

```
 9    function makeSelectNumber(from, to, prefix, suffix, id, parent){
10      let Select = makeElm("select", {"id":id}, parent);
11      for(let i=from; i<=to; i++) {
12        let option = makeElm("option",{ "value":i}, Select);
13        makeTextNode(prefix+i+suffix,option);
14      }
15      return Select;
16    };
```

このプログラムは 8 行目から始まる window.onload のイベントハンドラーの中にある。

9 行目から 16 行目では指定された範囲の値を選択できるプルダウンメニューを作成する関数 makeSelectNumber() を定義している。

- 引数は順に、下限値 (from)、上限値 (to)、数字の前後に付ける文字列 (prefix と suffix)、プルダウンメニュー の id 属性の属性値 (id) と親要素 (parent) である。
- 10 行目で select 要素を作成している。作成のために makeElm 関数 (18 行目から 26 行目で定義) を呼び出している。
- 11 行目から 14 行目で option 要素を順に作成している。
- 13 行目でプルダウンメニューに表示される文字列をテキストノードを作成する関数 makeTextNode() (25 行目から 27 行目) を呼び出している。

```
17    function makeElm(name, attribs, parent) {
18      let elm = document.createElement(name);
19      for(let attrib in attribs) {
20        elm.setAttribute(attrib,attribs[attrib]);
21      };
22      if(parent) parent.appendChild(elm);
23      return elm;
24    }
```

17 行目から 24 行目で与えられた要素を作成する関数 makeElm() を定義している。

- 引数は順に、要素名 (name)、属性値のリスト (attribs) と親要素の指定である。
- 指定された要素を作成し (18 行目) て、与えられた属性とその属性値がメンバーになっているオブジェクトの値を順に選んで属性値を設定する (19 行目から 21 行目)。
- 親要素が指定されている場合には、作成した要素を子要素として付け加える (22 行目)。
- 作成した要素を戻り値にする (23 行目)。

```
25    function makeTextNode(text,parent) {
26        parent.appendChild(document.createTextNode(text));
27    };
```

関数 makeTextNode() は指定された文字列を基にテキストノードを作成し、指定された親要素の子要素に設定している。

```
28    let Form  = document.getElementById("menu");
29    let Year  = makeSelectNumber(2000,2020,"","年","year", Form);
30    let Month = makeSelectNumber(1,12,"","月","month", Form);
31    let Days = [];
32    for(let d=28; d<=31; d++) {
33      Days[d] = makeSelectNumber(1,d,"","日","day");
34    }
```

56 行目の form 要素内にプルダウンメニューを作成する。プルダウンメニューで表示される日付は実行当日になるように設定される。

- 28 行目では form 要素を得ている。
- 29 行目と 30 行目ではそれぞれ年、月のプルダウンメニューを作成して form 要素の子要素にしている。
- 28 日から 31 日まであるプルダウンメニューを 4 つ作成して配列に格納している (32 行目から 34 行目)。ここでの関数呼び出しは最後の親要素が省略されてので、別の要素の子要素としては登録されない。

```
35    let today = new Date();
36    Year.value = today.getFullYear();
37    Month.value = today.getMonth()+1;
38    let d = new Date(Year.value, Month.value,0).getDate();
39    Form.appendChild(Days[d]);
40    Form.children[2].value = today.getDate();
```

ここでは、実行時の日付がプルダウンメニューの初期値として表示されるようにしている。

- Date() オブジェクトを引数なしでよぶと、実行時の時間が得られる (35 行目)。
- 36 行目で年を得ている。Date オブジェクトの getYear() メソッドは西暦の下 2 桁しか返さないので使わないこと。
- 37 行目で月を得ている。getMonth() メソッドの戻り値は 0(1 月) から 11(12 月) である。
- 39 行目と 40 行目では、得られた年と月をそれぞれのプルダウンメニューに設定している。月の値を 1 増やしていることに注意すること。
- Date() オブジェクトを年、月、日 (さらに、時、分、秒もオプションの引数として与えられる) を与えて、インスタンスを作成できる。実行当日の翌月の 0 日を指定することで翌月の

1 日の 1 日前の日付に設定できる。その日付から getDate() メソッドで現在の月の最後の日が得られる[2](38 行目)。

なお、Date オブジェクトの Day() は曜日を得るメソッドで 0(日曜日) から 6(土曜日) の値が返る。

- 39 行目で、この日数を持つプルダウンメニューを子要素として追加し、40 行目で、日の選択する値を設定している。

```
41     Form.addEventListener("change", function(){
42       let d2 = Form.children[2].value
43       d = new Date(Year.value, Month.value, 0).getDate();
44       if( d != Form.children[2].children.length) {
45         Form.replaceChild(Days[d],Form.children[2]);
46         Form.children[2].value = Math.min(Form.children[2].length, d2);
47       }
48     },false);
49   }
50 //]]>
51 </script>
```

ここではプルダウンメニューの値が変化したイベント (change) ハンドラーを登録している。

- イベントハンドラーは form 要素につけている。
- 42 行目で、現在表示されている日を保存している。
- 43 行目で、現在、プルダウンメニューで設定されている年月の日数を求めている。
- この日数と現在表示されている日数のプルダウンメニューの日数が異なる場合には (44 行目) 子ノードを入れ替える (45 行目)。さらに、初期値を現在の値と月の最終日の小さいほうに設定する (46 行目)。

```
53 <body>
54   <form id="menu"></form>
55 </body>
56 </html>
```

ここでは HTML 文書内で表示される要素を記述している。ここでは body 要素内に form 要素があるだけである。

[2] ネットの情報を参考にした。Date オブジェクトのコンストラクタは範囲外の日時を指定しても正しい日時に解釈してくれる。

第9回　PHPの超入門

9.1　Web サーバーの基礎知識

9.1.1　HTTP プロトコルの基礎

Web ページとサーバー間の通信には `http`(hypertext transfer protocol) または `https` が用いられる。これらのプロトコルには次のようなメソッドが定義されている。

表 9.1: http のよく使われるメソッド

`GET`	指定されたリソースを要求する。
`POST`	指定された URI のリソースを作成する。
`PUT`	指定されたリソースを置き換えたり、集めたりする。
`DELETE`	指定したリソースを消去する。
`HEAD`	指定したリソースの HTTP ヘッダーの部分だけ要求する。

後で見るように、これらのメソッドの定義を無視してサーバーのデータを操作したり、クライアント側に返すことも可能である。サーバーとクライアント側のやり取りを提供する Web サービス (Web API) ではメソッドに即した設計が求められている。このような設計思想で作られた API は RESTfull な API と呼ばれているようである[1]。

9.1.2　サーバーの重要な設定項目

Web サーバーは HTTP プロトコルを利用して、クライアントプログラム (Web ブラウザなど) からの要求を処理する。Apache はこのサービスを提供するものとして有名である。Apache の設定ファイル `httpd.conf` の中で設定される重要な項目は次のとおりである。

- ポート番号 : TCP/IP のサービスを識別するための番号で、次の 3 種類に分けられる[2]。

種類	範囲	内容
System Ports(Well Known Ports)	1 番 ～ 1023 番	assigned by IANA
User Ports, (Registered Ports)	1024 番 ～49151 番	assigned by IANA
Dynamic Ports(Private or Ephemeral Ports)	49152 番 ～65535 番	never assigned

[1] `https://ja.wikipedia.org/wiki/Representational_State_Transfer`
[2] `https://tools.ietf.org/html/rfc6335#section-6`

IANA(Internet Assigned Numbers Authority) は DNS Root, IP アドレスやインターネットプロトコルのポート番号などを管理している団体である。

- ドキュメントルート : Apache では Web サーバーに直接要求できるファイルはこのフォルダ (ディレクトリ) の下にあるものだけである。

なお、HTTP のポート番号は 80 番、HTTPS のポート番号は 443 番となっている[3]。

9.1.3　CGI

Common Gateway Interface(CGI) とは Web コンテンツを動的に生成する標準の方法で、Web サーバー上では Web サーバーと Web コンテンツを生成するためのインターフェイスを与える。この授業では CGI のプログラムは PHP(PHP: Hypertext Preprocessor) を使用する。

9.1.4　Web サーバーのインストール

Web サーバー (Apache) や PHP をインストールし、HTTP のサーバー上で PHP と連携させるためにはいくつかの設定をする必要がある。これらのインストールを一括で行うことができるパッケージも存在する。

XAMPP　XAMPP は Apache、MySQL、Perl と PHP を一括してインストールするパッケージである。インストール時にはいくつかのパッケージをインストールしない選択も可能である。XAMPP をインストールしたときに注意する点は次のとおりである。

- インストール時に Apache をサービスとして実行するとパソコンを立ち上げた時に Apache を起動させる手間が省ける。
- 標準の文字コードは UTF-8 である。
- PHP の設定でタイムゾーンがヨーロッパ大陸になっている。タイムスタンプを利用するプログラムがうまく動かない場合あがあるので注意すること[4]。

9.2　PHP とは

日本 PHP ユーザー会のホームページ[5]には次のように書かれている。

> PHP は、オープンソースの汎用スクリプト言語です。 特に、サーバサイドで動作する Web アプリケーションの開発に適しています。 言語構造は簡単で理解しやすく、C 言語の基本構文に多くを拠っています。 手続き型のプログラミングに加え、(完全ではありませんが) オブジェクト指向のプログラミングも行うことができます。

[3] ポート番号の一覧は次のところにある。
`https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml`

[4] `date_default_timezone_get()` 関数で調べることができる。

[5] http://www.php.gr.jp

なお、PHP は Web アプリケーションのためではなく、通常のプログラミング言語としても使用できる。通常のプログラミング言語として使用するためには、コマンドプロンプトから PHP が実行できるように、環境変数 `PATH` に `php.exe` があるフォルダを追記しておくとよい。

また、コマンドプロンプトから `php -S localhost:8080` を実行すると起動したフォルダをドキュメントルートとするポート番号が `8080` のサーバーが立ち上がる。

なお、このテキストの PHP の仕様に関する説明は日本 PHP ユーザー会から多く引用している。

9.3 PHP プログラムの書き方

通常のサーバーの設定では PHP のプログラムの拡張子は `php` である。Web アプリケーションではクライアントから拡張子が `php` のファイルが要求されると、サーバーが PHP のプログラムを処理して、その出力がクライアント側に送られる。

PHP で処理すべき部分は `<?php` と `?>` の間に記述する。それ以外の部分はそのまま出力される。このブロックは複数あってもかまわない。

9.4 データの型

PHP は 8 種類の基本型をサポートする。

- 4 種類のスカラー型
  - 論理値 (boolean)
    `TRUE` と `FALSE` の値をとる。小文字で書いてもよい。
  - 整数 (integer)
    整数の割り算はない。つまり 1/2 は 0 ではなく 0.5 となる。なお、PHP 7 では整数の割り算の結果を与える `intdiv()` 関数が導入された。たとえば、`intdiv(1,2)` の結果は `0` となる。
  - 浮動小数点数 (float, double も同じ)
  - 文字列 (string)
    JavaScript と同様に文字列の型はあるが、文字の型はない。
- 2 種類の複合型
  - 配列 (array)
    `array()` を用いて作成する。引数にはカンマで区切られた `key=>value` の形で定義する。`key=>`の部分はなくてもよい。このときは `key` として `0,1,2,...` が順に与えられる。なお、PHP の配列はキーと値を関連付ける連想配列しかない。
  - オブジェクト (object)
- 2 種類の特別な型:

- リソース (resource)
  オープンされたファイル、データベースへの接続、イメージなど特殊なハンドル
- ヌル (NULL)
  ある変数が値を持たないことを示す。

9.5　変数

PHP の変数の特徴は次のとおりである。

- 変数は特に宣言しない。
- 変数名は $で始まる。
- 変数に値を代入すればその値はすべてコピーされる。
- コピーではなく参照にしたい場合には変数の前に &を付ける。

実行例 9.1 次の例は単純な値を代入してある変数の参照をチェックしている。

```
 1  <?php
 2  $a = 1;
 3  $b = $a;
 4  print "1:\$a=$a, \$b=$b\n";// 1:$a=1, $b=1
 5  $a = 2;
 6  print "2:\$a=$a, \$b=$b\n";// 2:$a=2, $b=1
 7  $b = &$a;
 8  print "3:\$a=$a, \$b=$b\n";// 3:$a=2, $b=2
 9  $a = 10;
10  print "4:\$a=$a, \$b=$b\n";// 4:$a=10, $b=10
11  ?>
```

- 3 行目で変数 $a の値を変数 $b の代入している。4 行目で見るように両者の値は同じである。
- 5 行目で $a の値を変更しても、変数 $b の値は変化しない (6 行目)。
- 7 行目では変数 $b に変数の参照を代入している。この場合には変数 $a の値を変更すると (9 行目) 変数 $b の値も変更される (10 行目)。

9.5.1　定義済み変数

PHP にはいくつかの定義済みの変数が存在する。そのうち、スーパーグローバルと呼ばれる変数はグローバルスコープを持つ定義済みの変数である (表 9.2)。

これらの変数のうち、$_ENV $argc と$argv 以外は CGI で使用する。

表 9.2: スーパーグローバル

変数名	変数の内容
$GLOBALS	グローバルスコープで使用可能なすべての変数への参照
$_SERVER	サーバー情報および実行時の環境情報
$_GET	HTTP 通信における GET によるパラメータ
$_POST	HTTP 通信における POST によるパラメータ
$_FILES	HTTP 通信でファイルアップロードに関する情報
$_COOKIE	HTTP 通信におけるクッキーの情報
$_SESSION	HTTP 通信におけるセッション中に保持される情報
$_REQUEST	HTTP 通信におけるパラメータ (GET や POST の区別をしない)
$_ENV	環境変数のリスト
$argc	コマンドプロンプトから実行したときに与えられた引数の数
$argv	コマンドプロンプトから実行したときに与えられた引数のリスト

$argv は PHP をコマンドプロンプトから実行したときの (空白で区切られた) パラメータを保持する文字列の配列である。$argv[0] には実行している PHP ファイル名が渡される。$argc は count($argv) と同じ値である。

9.6 文字列

9.6.1 文字列の定義方法

PHP では文字列を定義する方法が 3 通りある。

- シングルクオート (') ではさむ。
  中に書かれた文字がそのまま定義される。
- ダブルクオート (") ではさむ。
  改行などの制御文字が有効になる。また、変数はその値で置き換えられる。これを変数が展開されるという。
- ヒアドキュメント形式
  複数行にわたる文字列を定義できる。<<<の後に識別子を置く。文字列の最後は行の先頭に初めの識別子を置き、そのあとに文の終了を表す; を置く。平ドキュメントの終わりを示す識別子は行の先頭から書く必要がある。**識別子の前に空白などを入れてはいけない。**

実行例 9.2 次の例はいろいろな文字列の動作を確認するものである。

```
1 <?php
2 print 'string1 \':abcd\n';
```

```
 3  print "string2 \":abcd\n";    // string1 ':abcd\nstring2 ":abcd
 4
 5  $a = 1;
 6  print 'string3 \':$a bcd\n';
 7  print "string4 \":$a bcd\n"; //string3 ':$a bcd\nstring4 ":1 bcd
 8  print "string5 \":{$a}bcd\n";//string5 ":1bcd
 9
10  $b = array(1,2,3);
11  print "string6:$b\n";        //string6:Array
12  print "string7:$b[1]aa\n";  //string7:2aa
13  print "string7:$b[$a]aa\n"; //string7:2aa
14
15  print <<<_EOL_
16  string8:
17    aa
18    \$a = $a
19  _EOL_;          //string8:
20                  //  aa
21                  //  $a = 1
```

- シングルクオートの文字列では改行を意味する \n がそのまま出力されているのに対し、ダブルクオートの文字列では改行に変換されている。
- シングルクオートの文字列では変数名がそのまま出力されているのに対し、ダブルクオートの文字列では変数の値に変換されている。この場合、変数として解釈されるのは変数名の区切りとなる文字 (空白など) である。そのため 7 行目では変数の後に空白を入れている。
- 空白を入れたくない場合などは変数名全体を { と } で囲む (8 行目)。
- 変数が配列の場合には添え字を個々に指定しなくてはいけない。その指定には別の変数が使用できる。
- ヒアドキュメントでは途中の改行もそのまま出力される。また、変数は展開される。

9.6.2　文字列を取り扱う関数

PHP では文字列を取り扱う関数が用意されている。これらの関数は文字列のメソッドとなっていないことに注意する必要がある。Unicode 文字列を取り扱う必要がある場合には mb_ で始まる関数群を使用する必要がある。具体的な関数の例は PHP のサイトで確認すること。

9.7 式と文

文の最後には必ず; が必要である。その他はほとんど C 言語と同じ構文が使える。演算子「+」は JavaScript と異なり、通常の数の演算となる。文字列に対して「+」を用いると数に直されて計算される。文字列の連接には「.」を用いる。

実行例 9.3 次の例は文字列の演算子の確認である。

```
1  <?php
2  print  1  + "2" . "\n";    // 3
3  print "1" + "2" . "\n";    // 3
4  print "1" . "2" . "\n";    // 12
5  print 1 . 2 . "\n";        // 12
6
7  print "1a" + "2" . "\n";  // 3
8  print "a1" + "2" . "\n";  // 2
9  print "0x1a" + "2" . "\n";// 28
```

- +演算子は常に数としての加法として扱われ、. は文字列の連接として扱われる。
- 文字列が数として不正な文字を含むとその直前まで解釈した値を返す (7 行目と 8 行目)。
- 16 進リテラルも正しく解釈される (9 行目)。

比較演算子についても C 言語と同様のもののほかに、データ型も込めて等しいことをチェックする===演算子と!==演算子がある。

表 9.3[6]は変数の状態を調べる関数が引数の値によりどのようになるか、また、論理値が必要なところでどのようになるかを表している。

- gettype() 変数の型を調べる
- isempty() 変数が空であることを調べる
- is_null() 変数の値が NULL であるかを調べる
- isset() 変数の値がセットされていて、その値が NULL でないことを調べる
- if(変数) としたときに TRUE となるかどうか

いろいろな値を論理値として評価した結果が JavaScript と異なる場合があるので注意が必要である。

表 9.4は比較演算子==の結果をまとめたものである。この演算子ではいろいろなものを数に変換してから判定をする。通常のプログラムでは厳密な比較をするのがよい。

課題 9.1 PHP と JavaScript で if(変数) と==の結果が異なるものを指摘しなさい。

PHP では分岐、繰り返しなどの制御構造は C 言語と同じように if 文、for 文、while 文と switch 文などがある。

[6] この表と次の表は PHP ユーザー会の HP から転用した。

表 9.3: PHP 関数による $x の比較

式	gettype()	empty()	is_null()	isset()	boolean : if($x)
$x = "";	string	TRUE	FALSE	TRUE	FALSE
$x = null;	NULL	TRUE	TRUE	FALSE	FALSE
var $x;	NULL	TRUE	TRUE	FALSE	FALSE
$x が未定義	NULL	TRUE	TRUE	FALSE	FALSE
$x = array();	array	TRUE	FALSE	TRUE	FALSE
$x = false;	boolean	TRUE	FALSE	TRUE	FALSE
$x = true;	boolean	FALSE	FALSE	TRUE	TRUE
$x = 1;	integer	FALSE	FALSE	TRUE	TRUE
$x = 42;	integer	FALSE	FALSE	TRUE	TRUE
$x = 0;	integer	TRUE	FALSE	TRUE	FALSE
$x = -1;	integer	FALSE	FALSE	TRUE	TRUE
$x = "1";	string	FALSE	FALSE	TRUE	TRUE
$x = "0";	string	TRUE	FALSE	TRUE	FALSE
$x = "-1";	string	FALSE	FALSE	TRUE	TRUE
$x = "php";	string	FALSE	FALSE	TRUE	TRUE
$x = "true";	string	FALSE	FALSE	TRUE	TRUE
$x = "false";	string	FALSE	FALSE	TRUE	TRUE

9.8　配列

9.8.1　配列の基礎

PHP の配列は array() 関数で作成する。配列には C 言語などのような数字を添え字とするものとキーと値の組み合わせであるものがある。

実行例 9.4 次のリストは配列の取り扱いに関する簡単な例である。

```
1 $a = array(0,10,20);
2 $b = $a;
3 $a[0] = 5;
4 print_r($a);
5 print_r($b);
```

- 1 行目で 3 つの値を持つ配列を定義
- ２行目で別の変数に値を代入
- ３行目で初めの配列の先頭の値を変更

表 9.4: == による緩やかな比較

	TRUE	FALSE	1	0	-1	"1"	"0"	"-1"	NULL	array()	"php"	""
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

- 4 行目と 5 行目で配列の内容を出力するために print_r() 関数を用いている。

出力結果は次のとおりである。

```
Array
(
    [0] => 5
    [1] => 10
    [2] => 20
)
Array
(
    [0] => 0
    [1] => 10
    [2] => 20
)
```

$b=$a で代入された変数$b は$a とは別の領域が割り当てられていることがわかる。

課題 9.2 実行例 9.1で print_r() 関数の代わりに var_dump() 関数を用いて出力がどのように変わるか確認しなさい。

キーと値を組み合わせた配列については次の項で解説をする。

9.8.2　配列に関する制御構造

JavaScript のオブジェクトのプロパティを列挙するのに利用した `for(... in ...)` に相当する `foreach` がある。この構文は形をとる。

`foreach(`「配列名」 `as [`キー `=>] `値`)`

「キー」と「値」のところは単純な変数を置くことができる。「キー」の部分は省略可能である。

実行例 9.5 次の例は `foreach` の使用例である。

```
1 $a = array("red","yellow","blue");
2 foreach($a as $val) {
3   print "$val\n";
4 }
5 foreach($a as $key=>$val) {
6   print "$key:$val\n";
7 }
```

1 行目で単純な配列を定義して、2 行目から 4 行目と 5 行目から 7 行目でこの配列の要素をすべて表示させている。この部分の出力は次のとおりである。

```
red
yellow
blue
0:red
1:yellow
2:blue
```

単純な配列のときは「キー」の値は $0, 1, 2, \ldots$ を順にとる (出力 4 行目から 6 行目)。ここでは `$a[$key]=$val` の関係が成立していることに注意すること。

```
1 $a = array("red"=>"赤","yellow"=>"黄","blue"=>"青");
2 foreach($a as $val) {
3   print "$val\n";
4 }
5 foreach($a as $key=>$val) {
6   print "$key:$val\n";
7 }
```

1 行目では JavaScript におけるオブジェクトリテラルの形式に似た連想配列を定義して、2 行目から 4 行目と 5 行目から 7 行目で以前とと同じ形で配列の内容を表示している。この出力は次のようになる。

```
赤
黄
青
```

red:赤
yellow:黄
blue:青

実行例 9.6 次の例は PHP の配列に関する特別な状況を説明するためのものである。

```
 1 $b = array();
 2 $b[3] = "3rd";
 3 $b[0] = "0";
 4 print count($b)."\n";
 5 foreach($b as $key=>$val) {
 6   print "$key:$val\n";
 7 }
 8 for($i=0;$i<count($b);$i++) {
 9   if(isset($b[$i])) {
10     print "$i:$b[$i]\n";
11   }
12 }
```

この出力は次のようになる。

```
2       //5 行目の出力
3:3rd   //6 行目の出力 1 回目
0:0     //6 行目の出力 2 回目
0:0     //1 0 行目の$i=0 のときの値
```

- 1 行目で配列を初期化し、その後、3 番目と先頭の要素に値を代入している。
- 関数 count() は配列の大きさ (正確には配列にある要素の数) を求めるものである。2 つの値しか定義していないので 2 となる。
- foreach で配列の要素を渡るときは定義した順に値が設定される。count($b) の値が 2 であったことから$b[3] の出力は現れない。

9.8.3 配列に関する関数

PHP には配列の処理に関するいろいろな関数 (もどき) がある。

list() 配列の個々の要素を変数にまとめて代入する。これは関数ではなく、言語構造である。

```
list($first,$second, $third) = array(1,2,3,4,5);
```

結果は、変数$first、$second と$third にそれぞれ 1,2,3 が代入される。途中に変数がなければ、その分は飛ばされる。次の例では変数$first と$third だけ代入が行われる。

```
list($first, , $third) = array(1,2,3,4,5);
```

array_pop() と array_push()　関数 array_pop() は引数の配列から最後の要素を取り除き、その要素を戻り値とする。関数 array_push() は 2 番目以降で与えられた引数を順に初めの引数の配列の最後に付け加える。

```
$a = array(1,2,3,4);
print array_pop($a);  //4 が出力される
array_push($a,10,20); //$a は array(1,2,3,10,20) となる
```

配列の先頭の要素の取り出し (array_shift()) や、配列の先頭に要素を追加 (array_unshift()) する関数がある。

配列のキーと値の存在の判定　in_array($needle, $array) は与えられた配列 ($array) 内に指定した値 ($needle) が存在するかを調べる関数である。値の型まで比較するためには省略可能な 3 番目の引数を TRUE にすればよい。存在すれば TRUE が返る。

array_key_exists($needle, $array) は与えられた配列 ($array) 内に指定したキー ($needle) が存在するかを調べる関数である。存在すれば TRUE が返る。

並べ替え　PHP には配列を並べ替える関数がいくつか用意されている。sort() は各要素を昇順に並べかえる。降順に並べ替えるには rsort() 関数を用いる。このほか、連想配列のキーで並べ替える ksort() と krsort() 関数、ユーザー指定の比較関数を用いて並べ替える usort() 関数などがある。詳しくは PHP ユーザー会のマニュアルを参考にすること。

配列からランダムな値を取り出す　shuffle() は与えられた配列の要素をランダムに並べ替える関数である。配列をそのままにしておきたい場合には array_rand() 関数を用いる。この関数は取り出すキーの値を返す。

配列の切り出しと追加　配列の要素の一部を取り出して別の配列にしたり (array_slice())、配列の一部を別の要素に置き換えたり (array_splice()) できる。

array_slice($array, $start, $length) は与えられた配列$array の$start の位置から長さ$length の部分を配列として返す関数である。

- $length が与えられないときは、配列の最後まで切り取られる。
- 元の配列は変化しない。
- $start の値が負のときは、配列の最後から数えた位置となる。

```
$a = array(0,1,2,3,4,5);
$res = array_slice($a,3);// $res は array(3,4,5), $a は変化しない
$res = array_slice($a,-2);// $res は array(4,5)
$res = array_slice($a,-3,2);// $res は array(3,4)
```

array_splice($array, $start, $length, $replace) は配列$array の$start の位置から長さ$length の部分を切り取り戻り値とする。切り取られた部分には配列$replace の要素が入る。なお、引数$start については array_slice() と同じ扱いとなる。

```
$a = array(0,1,2,3,4,5);
$r = array_splice($a,3,1,array("a","b"));// $r は array(3)
        // $a は array(0,1,2,"a","b", 4, 5)
```

課題 9.3 array_splice() 関数を用いて、array_pop()、array_push()、array_shift()、array_unshift() 関数の機能を実現しなさい。

9.9 関数

9.9.1 PHP の関数の特徴

PHP の関数の特徴は次のとおりである。

- 関数はキーワード function に引き続いて () 内に、仮引数のリストを書く。そのあとに {} 内にプログラム本体を書く。
- 引数は値渡しである。参照渡しをするときは仮引数の前に&を付ける。
- 関数のオーバーロードはサポートされない。
- 関数の宣言を取り消せない。
- 仮引数の後に値を書くことができる。この値は引数がなかった場合のデフォルトの値となる。デフォルトの値を与えた仮引数の後にデフォルトの値がない仮引数を置くことはできない。
- 関数は使用される前に定義する必要はない。
- 関数内で関数を定義できる。関数内で定義された関数はグローバルスコープに存在する。ただし、外側の関数が実行されないと定義はされない。
- 関数内の変数はすべてローカルである。
- 関数の外で定義された変数は参照できない。関数外で宣言された変数を参照するためにはスーパーグローバル$GLOBAL を利用する。
- 関数の戻り値は return 文の後の式の値である。複数の値を関数の戻り値にしたいときは配列にして返せばよい。

なお、PHP 7 では関数の仮引数や戻り値に対して型を宣言できるようになった。使用できる型には array、bool、float、string などがある。詳しくは PHP のサイトで確認すること。

PHP には外部ファイルを読み込むための関数 include() と一度だけ指定されたファイルを読み込む require_once() がある。このファイルの中で定義された変数は、読み込むテキストを直接記述したときと同じである。

9.9.2　関数の定義の例

実行例 9.7 次の例はユーザー定義関数の使用例である。

```
 1 <?php
 2 function example($a, $as, &$b, $f=false) {
 3   print "\$a = $a\n";
 4   print_r($as);
 5   print "\$b = $b\n";
 6   if(!isset($x)) $x = "defined in function";
 7   print "\$x = $x\n";
 8   if($f) {
 9     print "\$GLOBALS['x'] = ". $GLOBALS['x']."\n";
10   }
11   $a = $a*2;
12   $as[0] += 10;
13   $b = $b*2;
14   return array($a,$as);
15 }
16
17 $a = 10;
18 $as = array(1,2);
19 $b = 15;
20 $x = "\$x is defined at top level";
21 example($a, $as, $b, true);
22 print "\$a = $a\n";
23 print_r($as);
24 print "\$b = $b\n";
25 list($resa) = example($a, $as, $b);
26 print "\$resa = $resa\n";
27 ?>
```

- 1 行目の関数では 3 番目の引数が参照渡し、4 番目の引数にデフォルト値が設定されている。
- 6 行目の関数 `isset()` は与えられた変数に値が設定されているかを調べる関数である。
- ここでは`$x` は仮引数ではないのでローカルな変数となり、`isset($x)` は `false` となる。`!isset($x)` は `true` となるので変数`$x` には`"defined in function"`が代入される。
- 9 行目ではデフォルトの引数のチェックのための部分である。
- 変数に値を代入して、呼び出し元の変数が変わるかのチェックをする (11 行目から 13 行目)。
- 14 行目は初めの二つの仮引数を配列にして戻り値としている。

この関数の動作をチェックするのが残りの部分である。ここでは、関数に渡す引数の値を設定している。20 行目で呼び出した関数内での出力結果は次のようになる。

```
$a = 10
Array
(
    [0] => 1
    [1] => 2
)
$b = 15
$x = defined in function
$GLOBALS['x'] = $x is defined at top level
```

- 仮引数の値は正しく渡されている。
- 6 行目は判定が `true` になるのでここで新たに値が設定され、それが 7 行目で出力される。
- デフォルトの仮引数に対して `true` が渡されているので、9 行目が実行される。スーパーグローバル`$GLOBAL` にはグローバルスコープ内の変数が格納されている。ここでは 19 行目に現れる変数の値が表示される。

21 行目から 23 行目の出力は次のようになる。

```
$a = 10
Array
(
    [0] => 1
    [1] => 2
)
$b = 30
```

- 初めの 2 つの引数は配列であっても書き直されていない。11 行目と 12 行目の設定は戻り値にしか反映されない。
- 参照渡しの変数`$b` は書き直されている。
- 24 行目の関数呼び出しはデフォルトの引数がないので、引数`$f` の値が `false` に設定され、9 行目は実行されない。
- `list()` は配列である右辺の値のうち先頭から順に指定された変数に代入するので関数内の 11 行目で計算された値を変数`$resa` に代入することになる。

この部分の結果は次のとおりである。

```
$a = 10
Array
(
    [0] => 1
    [1] => 2
)
$b = 30
$x = defined in function
$resa = 20
```

第10回　サーバーとのデータのやり取り

10.1　サーバーとのデータ交換の基本

Web ページにおいてサーバーにデータを送る方法には POST と PUT の 2 通りの方法がある。

実行例 10.1 次のリストは実行例 8.1のリストに form のデータを POST で送るようにしたものである。JavaScript の部分だけ異なるのでそこだけのリストになっている。

windows.onload =function() 内に次のコードを追加する。

```
let Form = document.getElementsByTagName("form")[0];
Form.setAttribute("method","POST");
Form.setAttribute("action","09sendData.php");
```

HTML の要素に対しては次のことを行う。

- <select>要素の属性に name="select" を追加する。
- id が"colorName"であるテキストボックスに name="colorName" を追加する。
- 「設定」ボタンの要素の後に次の要素を追加する。

```
<input type="submit" value="送信" id="Send"></input>
```

このページでは「送信」ボタンを押すと<form>の action 属性で指定されたプログラムが呼び出される。ここでは Web ページと同じ場所にある 09sendData.php が呼び出される。このファイルのリストは次の通りである。

```
 1 <?php
 2 print <<<_EOL_
 3 <!DOCTYPE html>
 4 <head>
 5 <meta charset="UTF-8"/>
 6 <title>サーバーに送られたデータ</title>
 7 </head>
 8 <body>
 9 <table>
10 _EOL_;
11 foreach($_POST as $key=>$value) {
```

```
12   print "<tr><td>$key</td><td>$value</td></tr>\n";
13 }
14 print <<<_EOL_
15 </table>
16 </body>
17 </html>
18 _EOL_;
19 ?>
```

- 2 行目から 10 行目のヒアドキュメント形式で HTML 文書の初めの部分を出力している。
- method="POST"で呼び出されたときには form 要素内の name 属性が指定されたものの値が name 属性の属性値をキーとしてスーパーグローバル$_POST の連想配列として値が得られる。これはフォームの中のデータを application/x-www-form-urlencoded あるいは multipart/form-data によりエンコードされたもの[1]を、フォームの個々の値に戻して格納する。生の$_POST データを得るには関数 file_get_contents でファイル名に php://input を指定する。
- 11 行目から 13 行目でそれらの値を table 要素内の要素として出力している。

出力結果は次のようになる。

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8"/>
<title>サーバーに送られたデータ</title>
</head>
<body>
<table><tr><td>select</td><td>yellow</td></tr>
<tr><td>color</td><td>green</td></tr>
<tr><td>colorName</td><td>gray</td></tr>
</table>
</body>
</html>
```

なお、method="PUT"で呼び出した場合にはスーパーグローバル$_GET を用いる。また、スーパーグローバル$_REQUEST は method="POST"でも method="PUT"で呼び出された場合の$_POST や$_GET の代わりに使用できる。

type="submit"の input 要素はボタンであり、押されると直ちに action 属性で指定された処理が呼び出される。通常は、サーバーにデータを送る前に最低限のエラーチェックを行い、エラーがない場合にだけサーバーと通信するのが良い。

[1] GET メソッドでデータを送るときには URI の後に?name=value+を付ける。この形式が x-www-form-urlencoded である。

課題 10.1 PUT と GET を用いた HTML ファイルで実行後の URL がどうなっているか確認しなさい。さらに、実行例 10.1の PHP のプログラムを直接 URL に記述して呼び出したらどうなるか。また、`Form.setAttribute("method","POST");` の部分を `Form.setAttribute("method","PUT");` に変更して PUT による通信の場合について調べよ。

10.2 スパーグローバルの補足

スパーグローバルのうち、これまでに説明していないものについて解説をする。

$_SERVER この変数はサーバーにアクセスしたときのクライアントの情報などを提供する。具体的な内容はクライアントごとに異なる。

課題 10.2 foreach 構文を用いて$_SERVER の内容を表示し、それらの値を確認せよ。

$_COOKIE COOKIE とは Web サーバー側からクライアント側に一時的にデータを保存させる仕組みである。サーバーと通信するたびに、これらのデータがクライアント側からサーバー側に送られる。これにより、すでに訪問したことがあるサイトに対して情報を開始時に補填する機能などを実現できる。

$_SESSION セッションとはある作業の一連の流れを指す。たとえば会員制のサイトではログイン後でなければ会員専用のページを見ることができない。情報のページに直接行くことができないような仕組みが必要である。

HTTP 通信はセッションレスな通信である（各ページが独立して存在し、ページ間のデータを直接渡せない)。セッションを確立するために、クライアント側から情報を送り、それに基づいてサーバー側が状況を判断するなどの操作を意識的にする必要がある。

初期のころはページ間のデータを受け渡すために表示しない<input>要素の中に受け渡すデータを入れていた。これはセキュリティ上問題が生ずるので、COOKIE による認証が行われるようになった。

PHP ではセッションを開始するための関数 session_start() とセッションを終了させるための関数 session_destroy() が用意されている。セッションを通じで保存させておきたい情報はこの連想配列に保存する。セッションの管理はサーバーが管理することとなる。なお、この機能は COOKIE の機能を利用して実現されている。

10.3 Web Storage

HTML5 から独立した規格である Web Storage には localStoarage と sessionStorage の 2 種類のオブジェクトがある。これらのオブジェクトは文字列をキーに、文字列の値を持つ Storage オブジェクトである。同一の出身 (プロトコルやポート番号も含む) のすべてのドキュメントが同じ localStorage と sessionStorage を共有する。sessionStorage のデータは意識的に消さない限り存在する。これに対し、sessionStorage はウインドウやブラウザが閉じられると消滅する。

これにより、セッション間の情報の移動や以前の訪れたことのあるページの情報の保存を可能にしている。

実行例 10.2 次の例は localStorage を用いて、ページのアクセス時間を保存し、他のページに移動してもそのデータが利用できることを示すものである。

ページのアクセス時間を記録するページのリストは次のとおりである。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>WebStorage --- localStorage</title>
<script type="text/javascript">
//<![CDATA[
let Storage = window.localStorage;
//let Storage = window.sessionStorage;
window.onload = function() {
  let AccessList, Message = document.getElementById("message");
  let D = new Date();
  if(Storage["access"]) {
    AccessList = JSON.parse(Storage["access"]);
  } else {
    AccessList = [];
    appendMessage(Message, "初めてのアクセスです");
  }
  AccessList.unshift(D.getTime());
  Storage["access"] = JSON.stringify(AccessList);
  appendMessage(Message, "今までのアクセス時刻です");
  AccessList.forEach(function(D, i, A) {
    appendMessage(Message, new Date(D));
  });
}
function appendMessage(P, Mess) {
  let div = document.createElement("div");
  P.appendChild(div);
  div.appendChild(document.createTextNode(Mess));
}
//]]
</script>
</head>
<body>
```

```
35   <form action="10next.html">
36     <input type="submit" value="次のページ"></input>
37   </form>
38   <div  id="message"/>
39 </body>
40 </html>
```

- 8 行目で記述を簡単にするためと後で localStorage を sessionStorage に簡単に修正できるように変数 Storage を導入している。
- 11 行目で結果を表示するための 要素の得ている。
- 12 行目ではアクセスされた時間を求めている。
- WebStorage に access の要素が存在するかを調べ、存在する場合には変数 accessList に配列のデータとして復元する (13 行目から 14 行目)。これは、今までのアクセス時間を新しい順に並べた配列である。
- 存在しない場合には、変数 accessList を初期化し (16 行目)、初回のアクセスである旨の表示を行う (17 行目)。テキストを表示すある関数は 26 行目から 30 行目に定義してある。
- 新しい順に並べてあるアクセス時間の先頭に今回のアクセス時間 (1970 年 1 月 1 日 0 時 (GMT) からのミリ秒単位の時間) を挿入し (19 行目の unshift() は配列の先頭に引数のデータを挿入する配列のメソッドである)、WebStorage に JSON 形式で保存する (20 行目))。
- 以下のデータがアクセス時間のリストであるメッセージを表示した (21 行目) 後、アクセス時間をすべて表示している (22 行目から 24 行目)。
  - 配列の forEach メソッドは、配列の個々の要素に対して引数で与えられた関数を実行する。このメソッドの戻り値はない。
  - 値が undefined である要素に対しては実行されない。
  - 引数で渡される関数の引数は、処理する配列の現在の値 (D)、配列の位置 (i) と処理する配列 (A) の 3 つである。したがって、A[i] と D は同じ値である。
  - 処理結果を配列に戻したければ A[i] に値を代入すればよい。
- 26 行目から 30 行目が Web ページ上にデータを表示する関数を定義している部分である。
  - 引数は、表示するための親要素 (P) と表示する文字列 (Mess) である。
  - 要素を作成し (27 行目)、与えられた親要素の子要素にする (28 行目)。
  - さらに作成した子要素に与えられた文字列を表示するテキスト要素を子要素に付け加える (29 行目)

前のページで「次のページ」ボタンを押したとき、移動先のページのリストは次のとおりである。

```
 1 <!DOCTYPE html>
 2 <head>
 3   <meta charset="UTF-8"/>
 4   <script type="text/javascript">
 5     //<![CDATA[
 6     let Storage = window.localStorage;
 7     //let Storage = window.sessionStorage;
 8     window.onload = function() {
 9       let AccessList, Message = document.getElementById("message");
10       appendMessage(Message, "新しいページです");
11       if(Storage["access"]) {
12         AccessList = JSON.parse(Storage["access"]);
13       } else {
14         AccessList = [];
15         appendMessage(Message, "初めてのアクセスです");
16       }
17       appendMessage(Message, "今までのアクセス時刻です");
18       AccessList.every(function(D, index, A) {
19         appendMessage(Message, new Date(D-0));
20         return index < 3;
21       });
22     }
23     function appendMessage(P, Mess) {
24       let div = document.createElement("div");
25       P.appendChild(div);
26       div.appendChild(document.createTextNode(Mess));
27     }
28     //]]
29   </script>
30 </head>
31 <body>
32   <div  id="message"/>
33 </body>
34 </html>
```

このリストは呼び出す前のページのものとほとんど変わらないが、アクセス時間の表示が今回のものを含めて 4 つだけ表示されるようになっている (15 行目から 18 行目)。

- 前のリストの 22 行目の `AccessList.forEach` が `AccessList.every` となっている[2]。

[2] `forEach` と同様に値が `undefined` の要素に対しては実行されない。

- この配列のメソッドは引き渡された関数をすべての要素を順に適用し、戻り値の論理積を返す。
- どこかのところで `false` となれば全体の論理積は `false` となるので実行はそこで打ち切られる。
- 論理和を求める `some` という配列のメソッドもある。

課題 10.3 実行例 10.2で次のことを行いなさい。

1. デベロッパーツールから「Application」タブ (Chrome の場合) を開き、Local Storage の内容を確認する。
2. このページのタブを閉じ、いったんブラウザを終了してから再度、このページを開いたときに、以前のデータがそのまま表示されることを確認する。
3. `localStorage` の代わりに `sessionStorage` に変更したとき (リストの 3 行目をコメントにし、4 行目のコメントを外す) に、1と 2がどのように変わるか調べる。
4. `AccessList.every` となっている部分を `AccessList.some` でも正しく実行できるようにする。

すでに、いくつかのサイトではこの機能を用いており、その開発者ツールで見ることが可能である。これらのデータの形式は文字列である。構造化されたデータは JSON 形式で保存するのがよいであろう。

課題 10.4 いくつかのサイトにおいて Storage が利用されているか、どのようなデータが保存されているか調べよ。

10.4 Ajax

Ajax とは Asynchronous Javascript+XML の略で、非同期 (Asynchronous) で Web ページとサーバーでデータの交換を行い、クライアント側で得られたデータをもとにその Web ページを書き直す手法である[3]。Google Maps がこの技術を利用したことで一気に認知度が高まった。検索サイトでは検索する用語の一部を入力していると検索用語の候補が出てくる。これも Ajax を使用している (と考えられる)。

Ajax の機能は `XMLHTTPRequest` というオブジェクトの機能を用いて実現されている。

実行例 10.3 次の例は実行例 8.2で日付が変わったときに、その日の記念日をメニューの下部に示すものである。記念日のデータは `http://ja.wikipedia.org/wiki/`日本の記念日一覧の表示画面からコピーして作成した。

```
1 <!DOCTYPE html>
2 <html>
3 <head>
```

[3] `http://adaptivepath.org/ideas/ajax-new-approach-web-applications/`

```
 4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
 5 <title>今日は何の日</title>
 6 <script type="text/ecmascript">
 7 //<![CDATA[
 8   window.onload = function(){
 9     function makeSelectNumber(from, to, prefix, suffix, id, parent){
10       let Select = makeElm("select", {"id":id}, parent);
11       for(let i=from; i<=to; i++) {
12         let option = makeElm("option",{ "value":i}, Select);
13         makeTextNode(prefix+i+suffix,option);
14       }
15       return Select;
16     };
17     function makeElm(name, attribs, parent) {
18       let elm = document.createElement(name);
19       for(let attrib in attribs) {
20         elm.setAttribute(attrib,attribs[attrib]);
21       };
22       if(parent) parent.appendChild(elm);
23       return elm;
24     }
25     function makeTextNode(text,parent) {
26         parent.appendChild(document.createTextNode(text));
27     };
28     let Form  = document.getElementById("menu");
29     let Year  = makeSelectNumber(2000,2020,"","年","year", Form);
30     let Month = makeSelectNumber(1,12,"","月","month", Form);
31     let Days = [];
32     for(let d=28; d<=31; d++) {
33       Days[d] = makeSelectNumber(1,d,"","日","day");
34     }
35     let today = new Date();
36     Year.value = today.getFullYear();
37     Month.value = today.getMonth()+1;
38     let d = new Date(Year.value, Month.value,0).getDate();
39     Form.appendChild(Days[d]);
40     Form.children[2].value = today.getDate();
41     let changePulldown = function(){
42       let d2 = Form.children[2].value
43       d = new Date(Year.value, Month.value, 0).getDate();
```

```
44      if( d != Form.children[2].children.length) {
45        Form.replaceChild(Days[d],Form.children[2]);
46        d2 = Math.min(Form.children[2].length, d2);
47        Form.children[2].value = d2;
48      }
49      let  xmlHttpObj = new XMLHttpRequest();
50      if(xmlHttpObj) {
51        xmlHttpObj.onreadystatechange = function(){
52          if(xmlHttpObj.readyState == 4 && xmlHttpObj.status == 200) {
53            document.getElementById("details").innerText = xmlHttpObj.responseText;
54          }
55        }
56        xmlHttpObj.open("GET",
57          `./aniversary.php?month=${Month.value}&day=${d2}`,true);
58        xmlHttpObj.send(null);
59      }
60    }
61    Form.addEventListener("change", changePulldown,false);
62    changePulldown();
63  }
64 //]]>
65 </script>
66 </head>
67 <body>
68   <form id="menu"></form>
69   <p id="details"></p>
70 </body>
71 </html>
```

- 実行例 8.2とは 41 行目以降が異なっている。
- イベントハンドラーを関数として定義している (41 行目から 60 行目)。
- 42 行目から 48 行目は以前と同じプルダウンメニューの処理である。
- 49 行目では裏でサーバーと通信をするための `XMLHttpRequest` オブジェクトを作成している。
- `XMLHttpRequest` が生成できたら (50 行目)、このオブジェクトの `onreadystatechange` イベントのイベントハンドラーを登録する (51 行目から 55 行目)。
  - `XMLHttpRequest` の `readyState` は通信の状態を表す。4 は通信終了を意味する。これらの値については表 10.1を参照のこと。

  - 通信が終了しても正しくデータが得られたかを調べる必要がある。200 は正しくデータが得られたことを意味する[4]。
  - 得られたデータは `responseText` で得られる。この場合、得られたデータは文字列となる。このほかに `responXML` で XML データが得られる。

- 56 行目から 58 目が通信の開始する。ここでは、`GET` で行うので、URL の後に必要なデータを付ける。
- `GET` では送るデータ本体がないので、通信終了のため `null` を送信する。`POST` のときはここでデータ本体を送る。
- プルダウンメニューが変化したときのイベントハンドラーを登録し (62 行目)、最後に現在の日付データをサーバーに要求する (63 行目)。
- 得られたデータは 69 行目の `p` 要素のテキスト (`innerText`) として代入する (53 行目)。

表 10.1の状態は `XMLHttpRequest` オブジェクトのプロパティである。たとえば、`XMLHttpRequest.DONE` で利用できる。

表 10.1: XMLHttpRequest の通信の状態[5]

値	状態	詳細
0	UNSENT	`open()` がまだ呼び出されていない。
1	OPENED	`send()` がまだ呼び出されていない。
2	HEADERS_RECEIVED	`send()` が呼び出され、ヘッダーとステータスが通った。
3	LOADING	ダウンロード中 ; `responseText` は断片的なデータを保持している。
4	DONE	一連の動作が完了した。

課題 10.5 `XMLHttpRequest` オブジェクトのプロパティの値が利用できる確認しなさい。

次のリストは Ajax で呼び出される PHP のプログラムである。

```
 1 <?php
 2 mb_internal_encoding("UTF8");
 3 //print mb_internal_encoding();
 4 $m = isset($_GET["month"])?$_GET["month"]: $argv[1];
 5 $d = isset($_GET["day"])?$_GET["day"]:$argv[2];
 6 $data = file("aniversary.txt",FILE_IGNORE_NEW_LINES);
 7 for($i=0;$i<count($data);$i++) {
 8   $data[$i] = mb_convert_encoding($data[$i],"UTF8");
 9   $mm = mb_split("\[",$data[$i]);
10   if(count($mm) >1) {
```

[4]Http 通信の終了コードについては `http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html` を参照のこと

```
11     if(mb_convert_encoding($m."月","UTF8") === $mm[0]) break;
12   }
13 }
14 for($i++;$i<count($data);$i++) {
15   $data[$i] = mb_convert_encoding($data[$i],"UTF8");
16   $dd = mb_split("\s-\s",$data[$i]);
17   if(($d."日") === $dd[0]) break;
18 }
19 //  print mb_convert_encoding($dd[1],"SJIS");
20 print $dd[1];
21 ?>
```

- 2行目で内部で処理をするエンコーディングを`UTF8`にしている。関数、`mb_internal_encoding`関数を引数なしで呼び出すと現在採用されているエンコーディングを得ることができる。
- 4行目と5行目では月 (`$m`) と日 (`$d`) の値をそれぞれの変数に設定している。
  - ここではコマンドプロンプトからもデバッグできるように、スーパーグローバル`$_GET`内に値があれば (`isset()`) が`true`になれば、その値を、そうでなければコマンドからの引数を設定している。
  - コマンドから実行したときの引数はスーパーグローバル`$argv`に格納される。一番目は呼び出したファイル名であり、その後に引数が順に入る[6]。
- 6行目の`file`関数は指定されたファイルを行末文字で区切って配列として返す関数である。この引数にはURLも指定できる。
  - この関数は2番目の引数をとることができる。次の定数を組み合わせて使う。

`FILE_USE_INCLUDE_PATH`	`include_path`のファイルを探す
`FILE_IGNORE_NEW_LINES`	配列の各要素の最後に改行文字を追加しない
`FILE_SKIP_EMPTY_LINES`	空行を読み飛ばす

  - `file_get_contents()`はファイルの内容を一つの文字列として読み込む。Webページの解析にはこちらの関数を使うとよい。
- 読み込むファイルの一部を次に記す。

```
1月 [編集]
1日 - 鉄腕アトムの日
2日 - 月ロケットの日
[中略]
31日 - 生命保険の日、愛妻家の日
2月 [編集]
```

[6] C言語の`main`関数は通常、`int main(int argc, char* argv[])`と宣言される。`argc`は`argv`の配列の大きさを表し、渡された引数のリストが`argv[]`に入っている。このとき、`argv[0]`は実行したときのファイル名が入る。

```
1 日 - テレビ放送記念日、ニオイの日
2 日 - 頭痛の日
[以下略]
```

  - 月の部分の後には [がある。
  - 日の情報は␣-␣で区切られている (␣は空白を表す)。
  - すべての日の情報が入っている。

- 7 行目から 13 行目までは指定された月の行を見つける。
  - 8 行目で念のためコードを UTF8 に変更している。
  - 関数 mb_split() 関数は第 1 引数に指定された文字列パターンで第 2 引数で指定された文字列を分割して配列として返す関数である。
  - 分割を指定する文字列には正規表現が使えるので、文字 [で分割するために、"\["としている (9 行目)。
  - 指定された文字列があれば配列の大きさが 1 より大きくなる。その行に対して求める月と一致しているか判定し、等しければループを抜ける (11 行目)。
- 14 行目から 18 行目までは指定された月での指定された日の情報を探している。日を決定する方法も月と同じである。文字列の分割は"\s-\s"となっている[7]。
- 20 行目で得られた情報をストリームに出力している。

なお、構造化されたデータとしては XML 形式でもよいが、より軽量な JSON で与えることも可能である。このときは、JSON.parse() で JavaScript のオブジェクトに直せばよい。

課題 10.6 上の HTML のリストの 85 行目の と の空白を取り除いたら正しく動かなくなることを確認せよ。

Ajax の通信で大きなデータを渡すためには POST で行う行う。この例においては 57 行目から 59 行目の部分を次のように変更すれば POST で通信できる。

```
57        xmlHttpObj.open("POST", "./aniversaryPost.php",true);
58        xmlHttpObj.setRequestHeader("Content-type",
59                                    "application/x-www-form-urlencoded");
60        xmlHttpObj.send(‘month=${Month.value}&day=${d2}‘);
```

- open メソッドの一番目の引数を"POST"に変更し、2 番目の引数を呼び出す URL にする。URL には"GET"のときのようにデータを記述しない。
- その後、setRequestHeader メソッドで Content-type を指定する。
- その後、send メソッドで GET のときと同じ形式でデータを送信する。
- 受け取る側の PHP のプログラムでは$_GET のところを$_POST に変更する。

[7] これは"\s"ではうまく行かなかったためである。

第11回　jQuery

11.1　jQueryとは

jQuery の開発元[1]には次のように書かれている。

> jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

jQuery は JavaScript のライブラリーであり、これまでに解説してきた JavaScript の使い方を簡単にすることを目的としている。2017 年 11 月 26 日現在、jQuery は ver. 3.2.1 が最新のものとなっている。

11.2　jQueryの利用方法

jQuery はjquery.comのページの右上にある「Download jQuery」をクリックすることでダウンロードのページに行くと次の 2 つの最新版のダウンロードが可能となる。

Download the compressed, production jQuery 3.2.1

Download the uncompressed, development jQuery 3.2.1

これらのファイルの違いはコメントなどがそのまま入っているもの (uncompressed) と、コメントを取り除き、変数名などを短いものに置き換えて、ファイルサイズを小さくしたもの (compressed) の 2 種類がある。通常は compressed　バージョンを用いる。ダウンロードしたファイルを取り込む`<script>`要素を記述すれば利用できる。

jQuery のファイルをローカルにダウンロードしないで CDN(Contents Deliverly Network) を利用する方法がある。CDN とはライブラリーをネットワークからダウンロードできるようにしているサイトのことである。jQuery ではcode.jquery.com を開いて必要とされるバージョンのところをクリックすることで CDN を安全に利用する`<script>`要素の記述がクリップボードにコピーされる[2]。jQuery の compressed の最新版は minified をクリックすることで得られる。次のようなテキストがコピーされる。

[1] `https://jQuery.com` 参照日:2017/11/26

[2] コピーがうまくいかなかったら手作業で行うこと。

```
<script
  src="https://code.jquery.com/jquery-3.2.1.min.js"
  integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4="
  crossorigin="anonymous"></script>
```

課題 11.1 CDN を利用する利点について述べなさい。また、CDN を利用して配布されているライブラリーにどのようなものがあるか調べなさい。

11.3 jQuery() 関数と jQuery オブジェクト

jQuery ライブラリーは `jQuery()` というグローバル関数を一つだけ定義する。この関数の短縮形として、`$`というグローバル関数も定義する。jQuery() 関数はコンストラクタではない。jQuery 関数は引数の与え方により動作が異なるが、戻り値として jQuery オブジェクトとと呼ばれる DOM の要素を拡張した集まりを返す。このオブジェクトには多数のメソッドが定義されていて、これらの要素群を操作できる。

jQuery() 関数の呼び出し方は引数の型により返される要素群が異なる。

- 引数が (拡張された)CSS セレクタ形式の場合(**機能 1**)

  CSS セレクタにマッチした要素群を返す。省略可能な 2 番目の引数として要素や jQuery オブジェクトを指定した場合には、その要素の子要素からマッチしたものを返す。この形はすでに解説した DOM のメソッド `querySelectorAll()` と似ている。

- 引数が要素や Document オブジェクトなどの場合(**機能 2**)

  与えられた要素を jQuery オブジェクトに変換する。

- 引数として HTML テキストを渡す場合(**機能 3**)

  テキストで表される要素を作成し、この要素を表す jQuery オブジェクトを返す。`createElement()` メソッドに相当する。

  省略可能な 2 番目の引数は属性を定義するものであり、オブジェクトリテラルの形式で与える。

- 引数として関数を渡す場合(**機能 4**)

  引数の関数はドキュメントがロードされ、DOM が操作で可能になった時に実行される。

11.4 jQuery オブジェクトのメソッド

jQuery オブジェクトに対する多くの処理は HTML の属性や CSS スタイルの値を設定したり、読み出したりすることである。

- これらのメソッドに対してセッターとゲッターを同じメソッドを使う。メソッドに値を与えるとセッターになり、ないとゲッターになる。

- セッターとして使った場合は戻り値が jQuery オブジェクトとなるので、メソッドチェインが使える。
- ゲッターとして使った場合は要素群の最初の要素だけ問い合わせる。

HTML 属性の取得と設定 `attr()` メソッドは HTML 属性用の jQuery のゲッター/セッターである。

- 属性名だけを引数に与えるとゲッターとなる。
- 属性名と値の 2 つを与えるとセッターになる。
- 引数にオブジェクトリテラルを与えると複数の属性を一時に設定できる。
- 属性を取り除く `removeAttr()` もある。

CSS 属性の取得と設定 `css()` メソッドは CSS のスタイルを設定する。

- CSS スタイル名は元来の CSS スタイル名（ハイフン付）でも JavaScript のキャメル形式でも問い合わせ、設定が可能である。
- 戻り値は単位を含めて文字列で返される。

HTML フォーム要素の値の取得と設定 `val()` は HTML フォーム要素の `value` 属性の値の設定や取得ができる。これにより、`<select>`要素の選択された値を得ることなどができる。

要素のコンテンツの取得と設定 `text()` と `html()` メソッドはそれぞれ要素のコンテンツを通常のテキストまたは HTML 形式で返す。引数がある場合には、既存のコンテンツを置き換える。

11.5 ドキュメントの構造の変更

表 11.1は挿入や置換を行う基本的なメソッドをまとめたものである。これらのメソッドには対になるメソッドがある。

表 11.1: ドキュメントの構造の変更するメソッド

`$(T).method(C)`	`$(C).method(T)`	機能
`append`	`appendTo`	要素 `T` の最後の子要素として `C` を付け加える
`prepend`	`prependTo`	要素 `T` の初めの子要素として `C` を付け加える
`before`	`insertBefore`	要素 `T` の直前の要素として `C` を付け加える
`after`	`insertAfter`	要素 `T` の直後の要素として `C` を付け加える
`replaceWith`	`replaceAll`	要素 `T` と `C` を置き換える

このほかに、要素をコピーする `clone()`、要素の子要素をすべて消す `empty()` と選択された要素 (とその子要素すべて) を削除する `remove()` もある。

11.6　イベントハンドラーの取り扱い

jQuery のイベントハンドラーの登録はイベントの種類ごとにメソッドが定義されていてる。指定された jQuery オブジェクトが複数の場合にはそれぞれに対してイベントハンドラーが登録される。次のようなイベントハンドラー 登録メソッドがある。

```
blur()      error()    keypress()   mouseup()     mouseover()  select()
change()    focus()    keyup()      mouseenter()  mouseup()    submit()
click()     focusin()  load()       mouseleave()  resize()     unload()
dbleclick() keydown()  mousedown()  mousemove()   scroll()
```

このほかに、特殊なメソッドとして hover() がある。これは mouseenter イベントと mouseleave イベントに対するハンドラー を同時に登録できる。また、toggle() はクリックイベントに複数のイベントハンドラー を登録し、イベントが発生するごとに順番に呼び出す。

イベントハンドラー の登録解除には unbind() メソッドがある。このメソッドの呼び出しはいろいろな方法があるが、removeEvenListener() と同様な形式として 1 番目の引数にイベントタイプ (文字列で与える)、2 番目の引数に登録した関数を与えるものがある。この場合に、登録したイベントハンドラー には名前が必要となる。

イベントに関してはこのほかにも便利な事項が多くある。

11.7　Ajax の処理

jQuery では Ajax の処理に関するいろいろな方法を提供している。ここではもっとも簡単な処理を提供する jQuery.ajax() 関数を紹介する。

この関数は引数にオブジェクトリテラルをとる。このオブジェクトリテラルの属性名として代表的なものを表 11.2に挙げる。

表 11.2: jQuery.ajax() 関数で利用できる属性 (一部))

属性名	説明
type	通信の種類。通常は"POST"または"GET"を指定
url	サーバーのアドレス
data	"GET"のときは URL の後に続けるデータ。
dataType	戻り値のデータの型を指定する。"text"、"html"、"script"、"json"、"xml"などがある
success	通信が正常終了したときに呼び出されるコールバック関数
error	通信が成功しなかったときに呼び出されるコールバック関数
timeout	タイムアウト時間をミリ秒単位で指定

11.8 jQuery のサンプル

実行例 11.1 次の例は実行例 10.3を jQuery を用いて書き直したものである。

```
 1 <!DOCTYPE html>
 2 <html>
 3 <head>
 4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
 5 <title>今日は何の日 (jQuery 版)</title>
 6
 7 <script type="text/ecmascript" src="jquery-3.2.1.min.js"></script>
 8 <script type="text/ecmascript">
 9 //<![CDATA[
10 $(window).ready(function(){
11     function makeSelectNumber(from, to, prefix, suffix, id, parent){
12       let Select = $("<select/>", {"id":id});
13       if(parent) parent.append(Select);//$(parent).append(Select);
14       for(let i=from; i<=to; i++) {
15         option = $("<option/>",{"value":i,"text":prefix+i+suffix});
16         Select.append(option);
17 //          $("<option/>",{"value":i,"text":prefix+i+suffix}).appendTo(Select);
18       }
19       return Select;
20     };
21     let today = new Date();
22     let y = today.getFullYear();
23     let m = today.getMonth();
24     let Form  = $("#menu");
25     let Year  = makeSelectNumber(2000,2020,"","年","year", Form);
26     let Month = makeSelectNumber(1,12,"","月","month", Form);
27     let Days = [];
28     for(let i=28; i<=31; i++) {
29       Days[i] = makeSelectNumber(1,i,"","日","day");
30     }
31     Year.val(y);     //$("#year").val(y);
32     Month.val(m+1);  //$("#month").val(m+1);
33     let d = new Date(y, m+1,0).getDate();
34     Form.append(Days[d]);
35     $("#day").val(today.getDate());
36     let changePulldown = function(){
37       let d2 = $("#day").val();
```

```
38      d = new Date(Year.val(), Month.val(), 0).getDate();
39      if( d != $("#day option").length) {
40        $("#day").replaceWith(Days[d]);
41        $("#day").val(Math.min($("#day option").length, d2));
42      }
43      jQuery.ajax({
44        type:"GET",
45        url:      "./aniversary.php",
46        data:     "month=" + Month.val()+ "&day="+d2,
47        dataType: "text",
48        success : function(Data){
49          $("#details").text(Data);
50        },
51        error:function(){alert("error");}
52      });
53    };
54    Form.change(changePulldown);
55    changePulldown();
56 });
57 //]]>
58 </script>
59 </head>
60 <body>
61   <form id="menu"></form>
62   <p id="details"></p>
63 </body>
64 </html>
```

- 7 行目でローカルに保存した jQuery のライブラリーを読み込んでいる。
- 10 行目の$(window).redy() はドキュメントが解釈されたときに引数の関数が呼び出されるイベントである。ここでの要素の参照は機能 2を用いている。
- 11 行目から 21 行目は連続した番号を値に持つプルダウンメニューを作成する関数である。仕様は以前と同じである。

```
11    function makeSelectNumber(from, to, prefix, suffix, id, parent){
12      let Select = $("<select/>", {"id":id});
13      if(parent) parent.append(Select);//$(parent).append(Select);
14      for(let i=from; i<=to; i++) {
15        option = $("<option/>",{"value":i,"text":prefix+i+suffix});
16        Select.append(option);
```

```
17 //          $("<option/>",{"value":i,"text":prefix+i+suffix}).appendTo(Select);
18        }
19        return Select;
20      };
```

- 12 行目では<select>要素を作成し、同時に属性も設定している (機能 3)。
- 14 行目から 18 行目で<option>要素を定義し、<select>要素の子要素にしている。なお、15 行目と 16 行目は appendTo() メソッドを用いると 17 行目のコメントアウトしてあるように記述することができる。

- 21 行目から 34 行目も依然とほとんど同じである。24 行目では機能 3を用いて要素を参照している。
- 31 行目、32 行目と 35 行目はプルダウンメニューの初期値を本日に設定している。
- 36 行目から 53 行目はプルダウンメニュが変化したときに呼び出される関数を定義している。
  - jQuery のメソッドを用いていることを除けば 37 行目から 42 行目までは以前と同じである。39 行目のセレクタは id が day(#day) の下にある<option>要素を選択し、その数を length で調べている。
  - 43 行目から 52 行目は Ajax の処理を定義している。

```
43        jQuery.ajax({
44          type:"GET",
45          url:      "./aniversary.php",
46          data:     "month=" + Month.val()+ "&day="+d2,
47          dataType: "text",
48          success : function(Data){
49            $("#details").text(Data);
50          },
51          error:function(){alert("error");}
52        });
53      };
```

    * 44 行目では通信の方法を GET に指定している。
    * 45 行目では通信の相手のファイルを指定している。呼び出されるファイルは 106 ページのものがそのまま使える。
    * 46 行目では送信のデータを定義している。通信方法が GET の場合でも jQuery ではこのように分けて書いてよい。
    * 47 行目ではサーバーから返されるデータの形式を指定している。
    * 49 行目から 50 行目ではサーバーとの通信が成功した場合の処理を指定している。引数がサーバーから送られたデータとなる。json を指定するとこの値はオブジェクトになっている。

＊ 51 行目では通信にエラーが発生したときの処理を指定している。

XMLHTTPRequest の処理部分が大幅に減っていて見やすいコードになっていることがわかる。

課題 11.2 実行例 11.1に関して次の問いに答えよ。

1. 送信メソッドを `POST` に変更して実行しなさい。
2. メニューを作成終了後に変数 `Form` をコンソールに出力させてその構造を調べなさい。変数 `Form` の値と `document.getElementById("menu")` で得られるものが同じかどうかに注意すること。
3. 実行例 7.1を jQuery を用いて書き直しなさい。

第12回　XMLファイルの処理

12.1　XMLファイルとは

12.1.1　W3Cにおける XMLの解説

World Wide Web Consortium (W3C) によれば XML(eXtensible Markup Language) は次のように説明されている[1]。

> The Extensible Markup Language (XML) is a simple text-based format for representing structured information: documents, data, configuration, books, transactions, invoices, and much more. It was derived from an older standard format called SGML (ISO 8879), in order to be more suitable for Web use.

XML は構造化されたデータを表現する単純なテキストベースのフォーマットである。構造化された文書とは文書、データ、構成、本、、送付状やそのほかもろもろのものである。

さらに続けて次のように記されている。

> If you are already familiar with HTML, you can see that XML is very similar. However, the syntax rules of XML are strict: XML tools will not process files that contain errors, but instead will give you error messages so that you fix them. This means that almost all XML documents can be processed reliably by computer software.

ここにも書いてあるように XML の形式は HTML の形式に似ているが、XML の文法上に規則は厳格である。

さらに続けて HTML との違いが述べられている。

- すべての要素は閉じられているか、空の要素としてマークされる。
- 空の要素は通常のように閉じられている (`<happiness></happiness>`) か、特別な短い形式 (`<happiness />`) である。
- HTML では属性値はある条件下のもと (空白や特殊文字を含む場合) 以外では囲む必要はないが、その規則は覚えておくのには難しい。XML においては属性値は常に囲む必要がある (`<happiness type="joy"`)。
- HTML では組み込まれている要素名 (とその属性) の集まりがあるが、 XML ではそのようなものは存在しない (例外は `xml` で始まるものがある)。

[1] `https://www.w3.org/standards/xml/core`

- HTML ではいくつかの組み込まれた文字名 (エンティティ) があるが、XML には次の 5 つのものしかない。

  <(<), >(>), &(&), "(&), &apos(')

  XML では独自にエンティティを定義できる。

12.1.2　XML ファイルの例

ここでは GPS のログデータを保存する形式の一つとして普及している GPX 形式を取り上げる。`http://www.topografix.com/gpx.asp` で規格を見ることができる。

GPX ファイルの構造

GPX ファイルの構造は次のようになっている。
ルート要素は `gpx` で、その下の子要素としては次のようなものがある。

- `wpt`(ウェイポイント)
  ある地点の情報
- `rte`(ルート)
  子要素として地点を表す `rtept` を持つ
- `trk`(トラック)
  順序付けられた地点のリスト。子要素として `trkseg` を持つ
- `trkseg` は順序付けられた地点 (`trkpt`) のリストを持つ

これらの要素はすべてある必要はない。
`wpt` と `trkpt` のおもな構成要素は次の通りである。

名称	型	説明
`lat`	属性	地点の緯度
`lon`	属性	地点の経度
`ele`	要素	標高
`time`	要素	地点での時刻。世界標準時

GPX ファイルの例

次の GPX ファイルの例は Garmin E-treck Vista における記録の例である。

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <gpx version="1.1"
3 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4 xmlns="http://www.topografix.com/GPX/1/1"
5 xsi:schemaLocation="http://www.topografix.com/GPX/1/1
6 http://www.topografix.com/GPX/1/1/gpx.xsd>
```

```
 7 <trk>
 8  <name>20160324</name>
 9  <number>1</number>
10 <trkseg>
11 <trkpt lat="35.485802" lon="139.340909">
12  <ele>70.794189</ele>
13  <time>2016-03-24T08:33:23Z</time>
14 </trkpt>
15 <trkpt lat="35.485799" lon="139.340913">
16  <ele>70.794189</ele>
17  <time>2016-03-24T08:33:28Z</time>
18 </trkpt>
19 <trkpt lat="35.485793" lon="139.340899">
20  <ele>70.794189</ele>
21  <time>2016-03-24T08:33:33Z</time>
22 </trkpt>
23 ... 略 ...
24 </trkseg>
25 </trk>
26 </gpx>
```

- 2 行目から 7 行目が gpx のルート要素
- 8 行目に trk 要素
- 9 行目と 10 行目にはこのルートの名前、通し番号などが存在
- さらに trkseg が一つだけ存在
- trkseg 内には trkpt が存在

  初めの位置の情報は次の通り

  - 位置は北緯 35.4858026°(lat)、東経 139.340909°(lon)
  - 標高 70.794189m(ele)
  - 時刻 $2016-03-24T08:33:28$(2016 年 3 月 24 日 8:33:28)

地球の形

地球の形は測量法や測量法施行令で定められている。測量法施行令第三条では地球の長半径と扁平率が定められている。

一　長半径　六百三十七万八千百三十七メートル
二　扁平率　二百九十八・二五七二二一〇一分の一

これから地球上の 3 次元の位置を求めるには次のようにする。

元来は緯度と経度に対して扁平率を考慮して空間の位置を求める必要がある。しかし、GPX ファイルのデータでは 2 点間の距離が小さいのと、扁平率が小さく、高さも地球の半径 $R = 6378137$m

に対して小さいので無視して計算することにする。経度 λ、緯度 ϕ の地点の空間位置は次の通りである。

$$\begin{aligned} x &= R\cos\phi\cos\lambda \\ y &= R\cos\phi\sin\lambda \\ z &= R\sin\phi \end{aligned}$$

12.2　Google Maps における Polyline の表示

Polyline は指定した地点を折れ線でつなぎ、Polygon は指定した地点を頂点とする多角形を表示する。これらのオブジェクトは new google.maps.Polyline(<option>) または new google.maps.Polygn(<option>) で作成する。代表的なオプションは次の通りである。

プロパティ	説明
map	表示する地図
path	LatLng を要素とする (MVC) 配列
strokeColor	(縁取りの) 線の色
strokeOpacity	(縁取りの) 線の色の不透明度
strokeWeight	(縁取りの) 線の幅
fillColor	塗りつぶしの色 (Polygon のみ)
fillOpacity	塗りつぶしの色の不透明度 (Polygon のみ)

12.3　ブラウザでの処理

ブラウザ上ではセキュリティのためローカルのフォルダを参照できない。例外がファイルのアップロードである。ユーザの責任において指定したファイルをサーバーにアップできるようになる。

最近の JavaScript では FileReader オブジェクトを用いることで、サーバーに転送する代わりにそのファイルをブラウザ内で処理できる。

与えられた GPX ファイル内の道のりを Google Maps 上に表示する例を考える。

次のリストは HTML の部分である。

```
 1 <!DOCTYPE html>
 2 <html>
 3 <head>
 4 <title>Check Trace</title>
 5 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
 6 <link rel="stylesheet" type="text/css" href="map.css"/>
 7 <script src="http://maps.google.com/maps/api/js?"
 8         type="text/ecmascript" charset="UTF-8"></script>
 9 <script src="map-new2.js" type="text/ecmascript"></script>
10 </head>
11 <body>
12 <div id="map_canvas" style="width:800px; height:800px"></div>
13 <div id="form">
```

```
14   <div>
15     <form id="params">
16       <div><input type="file" id="file" value="ログデータファイルの選択"/></div>
17     </form>
18   </div>
19   <div id="info"> </div>
20 </div>
21 </body>
22 </html>
```

- 6 行目はこのページの外部スタイルシートを読み込むことを指定している。
- 7 行目から 8 行目で Google Maps API のファイルを読み込む。ここでは API キーが表示されていないので、コンソール画面に警告が現れる[2]。
- 9 行目は GPX ファイルのデータを処理する JavaScript の読み込みを指定している (ここでのファイル名は `map-rev2.js`)。
- 12 行目にある Google Maps の表示域である。
- 15 行目から 18 行目にはファイルのアップロードの要素が定義されている。
- 19 行目は表示された GPX ファイルに関する情報を示す位置を定めている。

次のリストはこの HTML で読み込まれるスタイルシート (`map.css`) である。

```
1 #map_canvas{
2     display: inline-block;
3 }
4 #form{
5     display: inline-block;
6     vertical-align:top;
7 }
8 table tr td {
9     text-align: right;
10 }
11 table > tr:nth-child(1) td:nth-child(2) {
12     text-align:center;
13 }
```

主に、地図とフォームを横並びにし (2 行目と 5 行目の `display:inline-block`)、表の数字を右寄せに (9 行目)、文字のところを中央ぞろえにしている (12 行目)。

次のリストは外部の JavaScript ファイルである。読み込み終了後に実行される関数だけを定義している。

```
1 window.onload = function (){
2   let PARAMS = {w:800, h:800};
3   let param = location.href.split("?");
4   if(param.length>1) {
5     param[1].split("&").forEach(function(p) {
6       v = p.split("=");
```

[2]実行が中断されるかもしれない。

```
7       if(v.length>1 && PARAMS[v[0]] !==undefined) PARAMS[v[0]] = v[1];
8     });
9   }
```

- 2 行目で地図を設定するデフォルトのパラメータを設定している。
- 3 行目で URL の後ろにパラメータ (?) があるかをチェックするためにこの文字で分けている。
- あった場合 (4 行目) は、パラメータの区切り文字&でさらに分解する (5 行目)。各文字列を=で名前と値に分割し (6 行目の forEach)、パラメタ (変数 PARAM) に代入している (7 行目)。

```
10  let Clrs = [[255,0,0],[0,255,0],[0,0,255],[127,127,0],[127,0,127],[0,127,127]];
11  let Colors = Clrs.map(function(C){ return `rgb(${C.join(",")})`;});
12  let C2 = Clrs.map(function(C) {
13    return "rgb("+ C.map(function(C1){
14        return Math.floor((C1+255*2)/3,1)}).join(",")+")"});
```

ここでは、GPX のログ内に複数の<trkseg>があった場合に、それらを色分けするための色を指定している。道のりの色は後で見るように下が透けるように指定して (86 行目) いるので、地図上と情報用の背景色が同じように見えるように色を調整している。

```
15  let GMap = google.maps;
16  let canvas = document.getElementById("map_canvas");
17  canvas.setAttribute("style","width:"+PARAMS.w+"px; height:"+PARAMS.h+"px;");
18  let map = new GMap.Map(canvas,
19    {//center:new GMap.LatLng(35.486210,139.341443),
20     mapTypeId: GMap.MapTypeId.ROADMAP,
21     scaleControl:true,
22     scaleControlOptions:GMap.ControlPosition.BOTTOM_LEFT,
23     zoom:10
24    });
```

ここでは初期の地図の表示を行っている。地図の中央 (center) を指定していないのでこの時点では地図は表示されない。

```
25  let Routes;
26  let table = document.getElementById("info");
27  document.getElementById("file").onchange =function(E) {
28    let R = 6378137;
29    let reader = new FileReader();
30    reader.onload = function(){
31      let log = new window.DOMParser().parseFromString(reader.result,"text/xml");
32      let trks = log.getElementsByTagName("trk");
33      Routes = Array.prototype.map.call(trks,function(trk){
34        let trksegs = trk.getElementsByTagName("trkpt");
35          return {route:Array.prototype.map.call(trksegs,function(P) {
36          let lat = P.getAttribute("lat");
37          let lon = P.getAttribute("lon");
38          let latRad = lat * Math.PI/180;
39          let lonRad = lon * Math.PI/180;
40          let latCos = Math.cos(latRad);
41          return [
42            lat, lon,
```

```
43            new Date(P.getElementsByTagName("time")[0].firstChild.nodeValue).getTime(),
44            R*latCos*Math.cos(lonRad),
45            R*latCos*Math.sin(lonRad),
46            R*Math.sin(latRad)];
47          })};
48      });
49      ShowPath(8, 0.5, 0);
50    }
51    reader.onerror= function(){alert("Error")};
52    reader.readAsText(E.target.files[0]);
53  };
```

ここでは、アップロードするファイルが変化したときのイベント処理を行っている。

- 29 行目で FileReader オブジェクトを作成している。
- 30 行目から 48 行目で、このオブジェクトが利用可能になった時のイベント処理関数を登録している。実際の起動は 52 行目で発生する。
- 33 行目で読み込まれたテキスト (reader.result) を"text/xml"で処理をする。処理をするオブジェクトが window.DOMParser() の parseFromString() メソッドである。
- この処理結果が DOM の構造になるので、今までの DOM の処理が適用できることになる。
- 32 行目で trk 要素に分解している。
- 各 trk 要素に対して処理を行うために、Array オブジェクトのメソッド forEach を用いる。
- 32 行目で得られた HTML コレクションには、このメソッドが直接適用できないあため、call メソッドを利用することで解決している。
- 34 行目で各 trk 要素内の trkpt 要素を求め、そこから各 trkpt 要素の緯度 (36 行目)、経度 (35 行目)、時間 (43 行目) を求めて、空間の位置を計算し (38 行目から 40 行目と 44 行目から 46 行目)、結果を配列で返している (map メソッドを利用)。空間の位置を計算する目的は道のりの長さを求めるためであるが、今回のプログラムでは計算していない。
- 49 行目でログを表示する関数を呼び出している。
- 51 行目は読み込みが失敗したときの関数を登録している。
- 52 行目では指定されたファイルのデータをテキストとして読み込んでいる。

次の部分は、ログの情報を表示する部分である。

```
54  function makeTR(data, tbl, color){
55    let tr = document.createElement("tr");
56    tbl.appendChild(tr);
57    if(color>=0) tr.style.background = C2[color%C2.length];
58    data.forEach(function(val) {
59      let td = document.createElement("td");
60      tr.appendChild(td);
61      td.appendChild(document.createTextNode(val));
62    });
63  }
```

- この関数の引数は、表に表示するデータ、表示するオブジェクト、(11 行目で定義した) 背景色の指定の 3 つである。
- 55 行目で作成した tr 要素の中に、与えられたデータを表示 (58 行目から 62 行目) する。

次のリストは道のりを表示する部分である。

```
64  function ShowPath(w, o, s) {
65    let tbl = document.createElement("table");
66    table.appendChild(tbl);
67    let latMin=180, latMax=-180, lonMin=180, lonMax=-180;
68    makeTR(["番号","日付","開始時間","終了時間","地点数"], tbl, -1);
69      Routes.forEach(function(Ps, i) {
70      let points = Ps.route.map(function(P){
71          latMax = Math.max(latMax, P[0]);
72          latMin = Math.min(latMin, P[0]);
73          lonMax = Math.max(lonMax, P[1]);
74          lonMin = Math.min(lonMin, P[1]);
75          return new GMap.LatLng(Ps[0],Ps[1]);
76      });
77      let d1 = new Date(Ps.route[0][2]);
78      let d2 = new Date(Ps.route[Ps.route.length-1][2]);
79      makeTR([i+1,
80        d1.toLocaleDateString(), d1.toLocaleTimeString(), d2.toLocaleTimeString(),
81        points.length], tbl, i);
82      new GMap.Polyline({
83        path: Ps.route.map(function(P){return new GMap.LatLng(P[0],P[1]);}),
84        strokeColor: Colors[(i+s)%Colors.length],
85        strokeWeight:w,
86        strokeOpacity:o,
87        map: map});
88      }
89    );
90    let latLngB = new GMap.LatLngBounds(new GMap.LatLng(latMax, lonMax));
91    latLngB.extend(new GMap.LatLng(latMin, lonMin));
92    map.fitBounds(latLngB);
93  }
94 }
```

- この関数の引数は道のりの幅、不透明度である。
- 69 行目から 76 行目で道のりを構成する地点を Google Maps の地点オブジェクトに変え (83 行目)、かつ、道のりの緯度、経度の最大値と最小値を求めている。
- 79 行目から 83 行目で道のりの情報を表示する。
- 82 行目から 89 行目で与えられたデータをもとに、道のりを表示している。道のりのデータはオブジェクトリテラルの形で与えられている。
- 90 行目から 92 行目では表示された道のりを構成する地点をすべて含む範囲 (LatLngBounds) を表示する最大のズームレベルを設定 (fitBounds メソッド) している。

12.4 PHP による処理

XML ファイルの処理は PHP でも可能である。次の例は GPX アイルから道のりに必要な JSON オブジェクトを作成する例である。

```
1 <?php
2 function getPos($lat, $lon) {
3     $R = 6378137;
4     $latRad = $lat * M_PI/180;
5     $lonRad = $lon * M_PI/180;
6     $latCos = cos($latRad);
7     return array(
8         $R*$latCos*cos($lonRad),$R*$latCos*sin($lonRad),$R*sin($latRad));
9 }
```

与えられた緯度経度から空間の位置を求める関数である。戻り値は空間の位置を配列として返す。

次のリストは与えられたファイルから GPX ファイルを処理する関数である。ファイル名は初めの引数で与えられる。

```
10 function SetDatafromFile($fn,$mode) {
11   $data = new DOMDocument();
12   $data->load($fn);
13   $trks = $data->getElementsByTagName("trk");
14   $len = $trks->length;
15   $trackdata = array();
16   $lengthdata = array();
17   $cnt=0;
```

- XML ファイルを読み込むために DOMDocument のインスタンスを作成 (11 行目) している。
- load メソッドにファイル名を指定することで、DOM オブジェクトに変換される (12 行目)。
- PHP ではピリオド . は文字列の連接演算子なのでメソッドを使用するためには->を用いる。
- getElementsByTagName で trk を取得 (13 行目) する。
- 14 行目でその数を変数$len に格納している。
- トラックごとの情報をしまうための変数を初期化 (15 行目と 16 行目) している。

```
18   for($i=0;$i<$len;$i++){
19     $trk = $trks->item($i);
20     $trksegs = $trk->getElementsByTagName("trkpt");
21     $len2 = $trksegs->length;
22     if($len2 <10) continue;
23     $trkseg = $trksegs->item(0);
24     $latmin = $latMax = $lat = $trkseg->getAttribute("lat");
25     $lonmin = $lonMax = $lon = $trkseg->getAttribute("lon");
26     $newtrk = array("[$lat,$lon]");
27     $ppos = getPos($lat, $lon);
28     $length = 0;
```

各 trk に対して距離などを求める。

- trk の一つを変数$trk に格納している。呼び出しに注意 (19 行目) すること。
- $trk に含まれる trkpt のリストを得る (20 行目)。
- 極端に短いログは省く (22 行目)。
- ログ内の地点の範囲を示す変数を初期化 (24、25 行目) している。
- 地点の情報をしまう配列を初期化 (26 行目) している。
- 初めの位置の空間座標を変数$ppos に格納 (27 行目) している。
- 積算距離の変数を初期化 (28 行目) する。

```
29     for($j = 1; $j < $len2; $j++) {
30       $trkseg = $trksegs->item($j);
31       $lat = $trkseg->getAttribute("lat");
32       $lon = $trkseg->getAttribute("lon");
33       $latMax = max($lat-0,$latMax);
34       $latmin = min($lat-0,$latmin);
35       $lonMax = max($lon-0,$lonMax);
36       $lonmin = min($lon-0, $lonmin);
37       array_push($newtrk,"[$lat,$lon]");
38       $cpos = getPos($lat, $lon);
39       $xd = $ppos[0]-$cpos[0];
40       $yd = $ppos[1]-$cpos[1];
41       $zd = $ppos[2]-$cpos[2];
42       $length += sqrt($xd*$xd+$yd*$yd+$zd*$zd);
43       $ppos = $cpos;
44     }
```

- 各 trkpt から緯度と経度を取り出し (31、32 行目)、今までの範囲外ならば範囲を更新 (33 行目から 36 行目) している。
- 緯度と経度を配列に追加 (37 行目) し、空間の位置を計算 (38 行目) する。
- ひとつ前の点との差を求め (39 行目から 41 行目)、距離を計算 (42 行目) する。
- ひとつ前の点を更新 (43 行目) する。

```
45     if($lonMax - $lonmin >0.001 ||$latMax - $latmin > 0.001) {
46       $cnt++;
47       array_push($trackdata,
48         '{"range":['."$latMax,$latmin,$lonMax,$lonmin".'],"route":[' .
49       implode(",",$newtrk).'],"length":'.($length/1000).'}');
50     }
51   }
52   return $trackdata;
53 }
54 ?>
```

- 移動がほとんどないトラックは登録しない (45 行目)。
- 登録するトラックを JSON 形式で作成 (48 行目から 49 行目) する。
- 関数 implode は配列の要素を与えられた文字列を挟んで出力する。

第13回　クラスの例

この回はクラスの例を示す。

13.1　例の概要

図 13.1はここで示す例の表示画面である。

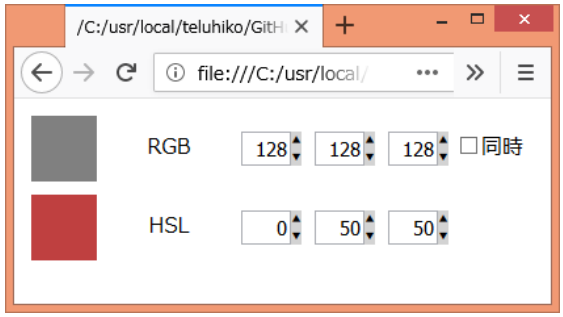

図 13.1: 色の指定を見る

この例では上下 2 行のテキストボックスで指定された色を左側の正方形の部分で示す。上は RGB 形式で、下は HSL[1]形式で色を指定することができる。色を変える操作は次のとおりである。

- それぞれのテキストボックスの値は▲をクリックすると増加し、▼をクリックすると 1 ずつ減少する。シフトキーを押しながらクリックすると 5 ずつ変化する。
- RGB 形式の値は 0 ～ 255 の間で変化する。上限または下限の範囲を超える場合は上限または下限の値にのままである。一番右のチェックボックスをチェックすると 3 つの値が同時に増減する。
- HSL 形式の値は一番左 (H–色相) が 0 ～ 359 の間で循環して変化し、残りの 2 つは 0 ～ 100 の間で変化する。

[1] W3C の CSS Color Module Level 3`https://www.w3.org/TR/2011/REC-css3-color-20110607/`を参照のこと

13.2　ソースコードの解説

13.2.1　HTML ファイル

次のリストは HTML ファイルのものである。

```
 1 <!DOCTYPE html>
 2 <html>
 3   <meta charset="UTF-8"/>
 4   <head>
 5     <script type="text/ecmascript" src="13Ex.js"></script>
 6     <script type="text/ecmascript" src="13UI.js"></script>
 7     <style type="text/css">
 8       .color{
 9       width:50px;
10       height:50px;
11       display:inline-block;
12       vertical-align:middle;
13       margin:5px;
14       }
15       </style>
16   </head>
17   <body>
18     <div id="RGB"><div class="color"></div></div>
19     <div id="HSL"><div class="color"></div></div>
20   </body>
21 </html>
```

- 5 行目ではこのページに関する処理を定義する JavaScript ファイル (`13Ex.js`) を読み込む。
- 6 行目ではユーザーインターフェイスを定義している JavaScript ファイル (`13UI.js`) を読み込む。
- 7 行目から 15 行目は色を表示する部分の CSS を定義している。
  - 9 行目と 10 行目では表示部分の大きさ
  - 11 行目では配置方法 (横に並べる)
  - 12 行目では上下の位置 (ここでは中央に指定)
  - 13 行目では色の表示域の周りの空白
- 18 行目と 19 行目では色の表示位置と値を設定するための`<div>` 要素を定義している。

13.2.2　ユーザーインターフェイス

次のリストはユーザーインターフェイスのクラスを定義するものである。

次のリストの部分は指定されたオブジェクトに指定できるオプションの値を変更するための関数を定義している。

```
1 function setOptions(props, Opt) {
2   for( let key in Opt) {
3     if(props.hasOwnProperty(key)) props[key] = Opt[key];
4   }
5 }
```

- 2 番目の引数 (Opt) は変更する値が入っているオブジェクトである。
- 1 番目の引数はキーがオブジェクト内の指定できるオプション (デフォルト値が設定されている) からなるオブジェクトである。
- 3 行目で指定できるキーであれば値を設定している。

次のリストの部分は HTML 要素に対する style を設定するための関数である。

```
6 function setStyle(props, Opt) {
7   setOptions(props, Opt);
8   return {
9     style: JSON.stringify(props).replace(/[{"]/g,"").replace(/[,}]/g,";")};
10 }
```

- 7 行目で、指定されたオプションを設定している。
- 9 行目でオプションのオブジェことを style 属性の形式になるように変更している。
  1. オブジェクトを JSON 形式の文字列に変換
  2. キーなどを囲む { と"を取り除く
  3. , と } を; に変換

11 行目から 43 行目のリストの部分は基本となるオブジェクト DOMObject を定義している。

```
11 let DOMObject = (()=>{
12   let NS ={
13     HTML:"http://www.w3.org/1999/xhtml",
14     SVG: "http://www.w3.org/2000/svg"
15   }
16   return class{
17     static getElmId(id) {
18       return {elm:document.getElementById(id)};
19     }
20     static getElmsTagName(elm) {
21       let elms = document.getElementsByTagName(elm);
22       return Array.prototype.map.call(
23           elms,(E)=>{r = new DOMObject(); r.elm=E;return r});
24     }
25     constructor(elm, attribs, parentNode, events, nameSpace="HTML"){
26       if(elm){
27         this.elm = document.createElementNS(NS[nameSpace], elm);
28         this.setAttribs(attribs);//console.log(parentNode);
29         if(parentNode&& parentNode.elm) {
30           parentNode.elm.appendChild(this.elm);
31         }
```

```
32        for(let evt in events) {
33          this.elm.addEventListener(evt, events[evt], true);
34        }
35      }
36    }
37    setAttribs(Opt) {
38      let elm = this.elm;
39      for(let attrib in Opt) {
40        elm.setAttribute(attrib,Opt[attrib]);
41      }
42    }
43  };})();
```

- オブジェクト内で有効な定数を定義するためにクラス式を返す関数を定義しその場で実行している (43 行目の `()`)。
- 12 行目から 15 行目で作成する要素の名前空間のリストをオブジェクトリテラルの形で定義している。ここでは HTML 要素と SVG 要素の名前空間がある。
- 17 行目から 19 行目で `document.getElementById()` の相当するこのオブジェクト用の関数を定義している。
- 20 行目から 24 行目で `document.getElementsByTagName()` の相当するこのオブジェクト用の関数を定義している。
  - 21 行目で指定された要素のリストを得ている。
  - 22 行目から 23 行目でそのリストを `DOMObject` のリストに変更している。
  - `map` は配列オブジェクト `Array` のメソッドで、各要素に対して引数で与えられた関数を実行し、その結果からなる配列を返す。
  - 21 行目で得られたリストは配列ではないのでこのメソッドを使用できない。
  - `call` は指定した関数が参照する `this` をその 1 番目の引数にする。
  - 23 行目の `(E)` 以降の書き方は新しい無名関数の記述方法である。`function(E){...}` と書くのと同じである。
  - 新しい書き方と従来の書き方の一番の違いは関数内での `this` の取り扱いである。`class` 内のコードは `strict` モードで実行される。このとき、`function` で定義された関数内では `this` は `undefined` になる。一方、簡略化された記述では `this` の値がオブジェクト自身を指す。この違いが問題となる例はこのリストの 170 行目などにある。
- 25 行目から 43 行目はこのオブジェクトの `constructor` を定義している。
  - 引数は順に作成する要素名、その属性のリスト、親要素、イベント処理のリスト、名前空間 (デフォルトは HTML) である。
  - 27 行目で名前空間を指定して要素を作成している。
  - 28 行目は作成した要素に、与えられた属性を設定する関数を呼び出している。

- 29 行目から 31 行目では親要素がある場合にはその子要素になるように指定している。
- 32 行目から 34 行目ではイベント処理を設定している。
- 37 行目から 42 行目では与えられたリストの属性を設定する関数を定義している。

次のリストの部分は与えられた文字列を表示するためのクラス divWithText を定義している。

```
44 class divWithText extends DOMObject{
45   constructor(text, parentNode, opt) {
46     super("div", opt, parentNode);
47     this.elm.innerText = text;
48   }
49 }
```

- このクラスは DOMObject を継承している。
- コンストラクタの引数は順に、表示するテキスト、親要素、属性である。
- 46 行目で 要素を作成し、その要素の innerText プロパティを設定することで表示を可能としている

50 行目からのリストの部分はテキストボックスに付属した値を上下するボタンを持つ spinbox のクラスを定義している。HTML にも spinbox があるが少し機能を変えている。
次のリストはこのクラスのコンストラクタの部分である。

```
50 class SpinBox {
51   constructor(Opt, P, Events, callback) {//console.log("called");
52     this.callback = callback;
53     this.values= {
54       max:Number.MAX_VALUE, min:-Number.MAX_VALUE,
55       skip: 1, bigSkip:5, value:0,
56       type:"limited"// "cyclic"
57     },
58     this.boundary = {
59         display:"block", margin: "0px", border: "0px", padding:"0px"};
60     let container = new DOMObject("div", setStyle(this.boundary, Opt), P);
61     setOptions(this.values, Opt);
62     let div0 = new DOMObject("div",
63       {style:"display:inline-block; vertical-align:middle;"},
64        container);
65     this.textBox = new DOMObject(
66       "input", {type:"text", size:"1em", value:this.values.value, readonly:"readonly",
67            style:"font-size:15px; text-align:right; disable" }, div0);
68     let div = new DOMObject("div",
69          {style:"display:inline-block;vertical-align:middle"}, container);
70     let buttonStyle = {
71       style: "font-size:8px;margin:0px;cursor:default;"+
72              "background:lightgray;border-color:black;"
73     }
74     buttonStyle.type = "up";
75     this.buttonUP = new DOMObject(
76         "div", buttonStyle, div, {click:(E)=>{this.up = E.shiftKey;}});
77     this.buttonUP.elm.innerText ="▲";
```

```
78     buttonStyle.type = "down";
79     this.buttonDOWN = new DOMObject(
80         "div", buttonStyle, div, {click:(E)=>{this.down = E.shiftKey;}});
81     this.buttonDOWN.elm.innerText ="▼";
82   }
```

- コンストラクタの引数は順に、オプション、親要素、イベント処理関数群、値が変化したときの処理関数である。
- 52 行目は値が変化したときに呼び出される関数をオブジェクトに登録している。
- 53 行目から 57 行目ではこのオブジェクトのデフォルトのパラメータを定義している。
  - max は設定できる値の最大値 (デフォルト値は JavaScript で扱うことができる最大値)
  - mix は設定できる値の最小値 (デフォルト値は JavaScript で扱うことができる最小値)
  - skip は値の変化量 (デフォルト値は 1)
  - bigSkip はシフトキーを押しながらクリックしたときの値の変化量 (デフォルト値は 5)
  - value は初期値 (デフォルト値は 0)
  - type は両端の値から外れたときの処理方法 (デフォルトは両端の値に固定–"limited"。値が循環的に変わる"cyclic"がある)
- 58 行目から 59 行目でオブジェクトの周りの余白の設定をしている。
- 60 行目で入れ物の 要素を作成している。
- 61 行目では 53 行目からのデフォルトのパラメータを与えられた値に変更している。
- 62 行目から 64 行目では 65 行目から 67 行目で定義されるテキストボックスを入れるための 要素を作成している。
- 66 行目から 67 行目で定義されるテキストボックスは値を直接変更できない設定をしている ("readonly"の指定)
- 68 行目から 69 行目ではテキストボックスの隣に置く上下のボタンを入れるための 要素を作成している。
- 70 行目から 73 行目では値の上下させるボタンの表示形式を定義している。
- 74 行目でさらにボタンのタイプ (値を増加) を設定し、75 行目から 76 行目でボタンとして機能するようなオブジェクトを追加している。この上でクリックイベントが生じたときにイベント発生時ののシフトキーの状態を sipnbox の up プロパティの与えている (代入で行っているが、up は 85 行目から 94 行目で定義されているセッターである)。
- 77 行目から 82 行目は値を減少させるボタンを設定している。

残りの部分はこのクラスのセッターまたはゲッターを定義している部分である。

```
 83   get value()    {return this.values.value;}
 84   set value(val) {this.textBox.elm.value = this.values.value = val;  }
 85   set up(shift){
 86       let skip = shift?this.values.bigSkip:this.values.skip;
 87       this.value += skip;
 88       if(this.values.type=="limited"){
 89           this.value = Math.min(this.value, this.values.max);
 90       } else {
 91           if(this.value >=this.values.max) this.value -= this.values.max;
 92       }
 93       if(this.callback) this.callback();
 94   }
 95   set down(shift){
 96       let skip = shift?this.values.bigSkip:this.values.skip;
 97       this.value -= skip;
 98       if(this.values.type=="limited"){
 99           this.value = Math.max(this.value, this.values.min);
100       } else {
101           if(this.value < this.values.min) this.value += this.values.max;
102       }
103     if(this.callback) this.callback()
104   }
105 }
```

- 83 行目と 84 行目はそれぞれテキストボックスの値に関するゲッターとセッターである。
- 85 行目から 94 行目は設定値を増大させるメソッドであり、95 行目から 104 行目は設定値を減少させるメソッドである。
- 86 行目ではシフトキーが押されているかを判定して、変化量を決めている。
- 87 行目で仮の値を求め、それが上限値を超えている処理を 88 行目から 92 行目で行っている。上限値で固定 (`type` が`"limited"`) のときは上限値と仮の値の小さい方を設定している (99 行目)。値が循環する (`type` が`"cyclic"`) ときは上限値が超えた場合、上限値の値を引いている (91 行目)。
- 値の設定後の処理関数が定義されているときはその関数を呼び出す (93 行目)。
- 値が減少する場合も同様である。

リストの最後の部分は色の 3 成分を設定するクラスの定義である。

```
106 class SetColor {
107   constructor(title, type, P, callback, opt) {
108 //    let that = this;
109     this.callback = callback;
110     if(title) {
111       new divWithText(title, P,
112         {style:"display:inline-block;width:100px;text-align:center"});
113     }
114     if(type.match(/^rgb$/i)) {
115       this.type = "rgb";
116       this.elm = new DOMObject("div",opt, P,
```

```
117           {click:(E)=>{
118           if(this.checkBox.elm.checked){
119             this.RGB.forEach((C)=>
120               {C[E.target.getAttribute("type")] = E.shiftKey;});
121             E.stopPropagation();
122             if(this.callback) this.callback(E);
123           }
124         }
125       }).elm;
126       this.RGB = Array(3).fill(0).map(()=>{
127         return new SpinBox({
128           type:"limited",
129           display:"inline-block", margin:"5px",
130           value:128, max:255, min:0
131           },
132           this, {}, this.callback);
133       });
134       this.checkBox = new DOMObject("input",
135           {type:"checkbox", style:"display:inline-block"},this);
136       new divWithText("同時", this, {style:"display:inline-block;"});
137     } else {
138       this.type = "hsl";
139       this.elm = new DOMObject("div",opt, P).elm;
140       this.HSL = [[360,0],[100,50],[100,50]].map((V)=>{
141         return new SpinBox({
142           type:"limited",
143           display:"inline-block", margin:"5px",
144           value:V[1], max:V[0], min:0
145           },
146           this, {}, this.callback);
147       });
148       this.HSL[0].values.type="cyclic";
149     }
150   }
151   get rgb()    {return this.RGB.map((V)=>{return V.value});}
152   set rgb(vals){vals.forEach((V, i)=>{this.RGB[i].value = V;});}
153   get hsl()    {return this.HSL.map((V)=>{return V.value});}
154   set hsl(vals){vals.forEach((V, i)=>{this.HSL[i].value = V;});}
155   toString(){
156     return (this.type == "rgb") ?
157       `rgb(${this.rgb.join(",")})`:
158       `hsl(${this.hsl[0]},${this.hsl[1]}%,${this.hsl[2]}%)`;
159   }
160 }
```

- コンストラクタに引数は順に、左端に示すテキスト、RGB 形式か HSL 形式のタイプ、親要素、値が変化したときの処理関数、オブジェクトのオプションである。
- 108 行目のコメント行は、メソッド内で呼び出される無名関数内でオブジェクト自身を示す `this` が参照できないときの対処法である。`this` を変数 (`that`) に格納し、それを参照する。
- 110 行目から 113 行目は左端にテキストが表示されるようにしている。
- 114 行目から 137 行目ではタイプが`"rgb"`のときのオブジェクトを作成している。

- 115 行目でタイプを"rgb"に設定している。
- 116 行目から 141 行目でオブジェクト全体を囲む要素を作成して、そこに click イベントの処理関数を定義している。
- このオブジェクトの右端のチェックボックスにチェックがある場合は RGB の値を一斉に増減させる。そのときは (118 行目) この関数で処理をする (119 行目から 123 行目)。
- 119 行目から 120 行目で RGB の値を変化させている (this.RGB.forEach)。
- それぞれの sipnbox にもイベントの処理関数がついているので、2 回処理させないために、121 行目でイベントの伝搬を止めている (E.stopPropagation())。
- その後、登録された処理関数を呼び出している。
- 126 行目から 133 行目では spinbox3 つ作成のために、大きさ 3 の配列を作成し (new Array(3))、それぞれに 0 を設定 (fill(0)) する。この配列に forEach を用いて spinbox を作成する。

  Array のメソッド forEach は undefined の要素に対して実行されないのでこのような処理が必要となる。
- これらの spinbox はタイプが"limited"、最小値が 0、最大値が 255、初期値を 128 である。
- 134 行目から 136 行目では RGB 値を同時に変化させるかどうかのチェックボックスを作成している。

- 138 行目から 148 行目はタイプが"hsl"のときのオブジェクトを作成している。
  - 138 行目でタイプを"hsl"に設定している。
  - 139 行目ではオブジェクト全体を囲む要素を定義している。
  - 140 行目から 147 行目では 3 つの spinbox を定義している。
  - 色相 (H) と残りの 2 つでは値の範囲と取り扱いが異なるので、範囲と初期値を与える配列を利用して 3 つの spinbox を作成している。
  - 148 行目でははじめの spinbox のタイプを"cyclic"に修正している。
- 151 行目では rgb の値をコピーして配列で返すゲッターを、152 行目では rgb の値のセッターをを定義している。
- 153 行目では hsl の値をコピーして配列で返すゲッターを、154 行目では hsl の値のセッターをを定義している。
- 155 行目から 159 行目では CSS3 で定義されている RGB や HSL 形式に変換するメソッドを定義している。

13.2.3　ユーザーインターフェイスを引用する JavaScript ファイル

次のリストはユーザーインターフェイスを利用して、色の変化を見えるようにするものである。

```
 1 window.onload = function(){
 2   let areaColor = ["RGB", "HSL"].map((type)=>{
 3     let area = DOMObject.getElmId(type);
 4       return [area, new SetColor(type, type, area, function(){setColors();},
 5                     {style:"display:inline-block;vertical-align:middle;"})];
 6     });
 7   setColors();
 8   function setColors(){
 9     areaColor.forEach((C)=>
10       {C[0].elm.children[0].style.background = `${C[1]}`;});
11   }
12 }
```

- 2 行目から 6 行目までで 2 つの色が設定できるオブジェクトを作成している。一つ目は RGB 形式で 2 つ目が HSL 形式となる。区別するパラメータは配列で与えている。
- 7 行目ではその値を左側の背景色に反映するための関数を呼び出している。
- 8 行目から 11 行目では `spinbox` で設定された色を背景色に設定する関数である。
- 10 行目で `spinbox` の値を文字列に変換することで行っている。

第14回　システム開発のためのヒント

14.1　コードの構造

良いコードとは

良いコードとはどのようなものであろうか。それは個人によっても違うであろう。しかしながら、多くの公開されているコードを眺めているといくつかの共通点が見つかる。

- プログラムの構造がわかりやすくなるように字下げ (インデント) を付ける。
- 変数名や関数名は実態を表すようにする。
  いくつかの単語をつなげて変数名や関数名にすることが多くなる。これらが読みやすいように途中に出てくる単語の先頭は大文字にすることが最近の傾向である (キャメル形式と呼ばれる)。
- 不必要に長いプログラム単位を作らない。
  ある程度まとまった作業をする部分は関数にすると見通しの良いプログラムとなる。
- プログラム自体が説明書になっている。
  コメントを多用して必要な説明をすべて記述する。
- プログラムとデータの分離
  ある処理をするときの分岐の条件をプログラム内に直接記述すると、分岐の条件が変わったときにはプログラムをコンパイルしなおす必要が生ずる。これを避けるためには外部からデータを読み込んで、それに基づいた処理を行うようにすることができるか考えてみる。

プログラムとデータの分離

実行例 14.1 期末試験の結果、90 点以上ならば S、80 点以上ならば A、70 点以上ならば B、60 点以上ならば C、それ以外は E の成績をつけることにした。期末試験の点を与えて、評価の S などを返す関数 seiseki を作成しなさい。

一番思いつくコードは次のようなものであろう (JavaScript で記述)。

```
function seiseki(val) {
  if(val >= 90) return "S";
  if(val >= 80) return "A";
  if(val >= 70) return "B";
```

```
  if(val >= 60) return "C";
  return "E";
}
```

これに対し、数値と戻り値をそれぞれ配列の値として用意するとつぎのようなコードとなるであろう。

```
function seiseki(val) {
  let Score = [90,80,70,60,0];
  let Results= ["S","A","B","C","E"];
  for(let i=0;i<Grade.length;$i++) {
    if(val>=Grade[i]) return Hyouka[i];
  }
  return "Error";
}
```

二つの配列をまとめて一つの配列にするほうがコードの管理上わかりやすくなるであろう。

```
let Results =[["S",90],["A",80],["B",70],["C",60],["E",0]];
```

または、次のような連想配列にしてもよい。

```
let Results ={S:90,A:80,B:70,C:60,E:0};
function seiseki(val) {
  for(v in Results) {
    if(val >=Results[v]) return v;
  }
}
```

JavaScript では `for(..  in ..)` で連想配列の要素をすべて渡ることができ、かつわたる順序が定義された順なのでこのコードが可能となる。

このように配列をうまく利用するとロジックの部分とパラメータの部分の分離が可能となる。

なお、JavaScript では JavaScript 内から外部ファイルを自動で読み込むことができないので上のような変数 `Results` を宣言する必要がある。この変数部分を別のプログラムから作成すればもう少し管理が楽になるであろう。

14.2　コードの管理の問題点

複数でシステム開発をするときの問題点

システムを開発していると次のような問題が発生する。

- 複数の人間で開発している場合、開発者の間で最新のコードの共有
- ファイルを間違って削除したり、修正がうまくいかなかったときに過去のコードに戻す

- 安定しているコードに新規の機能を付け加えてテストをする場合、安定したコードに影響を与えないようにする

このようなコードの変更履歴の管理するソフトウェアをバージョン管理ソフトという。バージョン管理ソフトウェアの対象となるファイルはテキストベースのものを主としている。

14.3 バージョン管理

14.3.1 バージョン管理の概念

バージョン管理ソフトは各時点におけるファイルの状態を管理するデータベースである。このデータベースは一般にリポジトリと呼ばれる。

開発者は通常次の手順でシステムの開発を行う。

- リポジトリからファイルの最新版を入手
- ローカルな環境でファイルを変更。
- ファイルをリポジトリに登録

バージョン管理ソフトでは同じファイルを複数の開発者が変更した場合、競合が発生していないかをチェックする機能が備わっている。この機能がない場合には同じファイルの変更は同時に一人しか変更ができないようにファイルをロックする。バージョン管理システムでは一連の開発の流れがある。この開発の流れをブランチと呼ぶ。あるブランチに対して新規の機能を付け加えたいときなどには元のブランチには手を付けずに別のブランチを作成して、そこで開発をする。機能が安定すれば元のブランチに統合 (merge) する。

14.3.2 バージョン管理ソフト

CVS(Concurrent Versions System) やこの改良版である Subversion はバージョン管理するためにサーバーが必要となる。開発者はこのサーバーにアクセスしてファイルの更新などを行う。

これに対し、git は各開発者のローカルな環境にサーバー上のリポジトリの複製を持つ。これにより、ネットワーク環境がない状態でもバージョン管理が行える。git ではネットワーク上に GitHub と呼ばれるサーバーを用意しており、ここにリポジトリを作成することで開発者間のデータの共有が可能となっている。データを公開すれば無料で利用できるほか、有料のサービスやサーバー自体を個別に持つサービスも提供している。

現在では有名なオープンソースのプロジェクトが ‘GitHub を利用して開発を行っている。

14.3.3 git の使い方

git の詳しい使い方については次のサイトを参照すること。

`https://git-scm.com/book/ja/v2/`

ここでは簡単な使い方を解説する。

git の特徴

git のバージョン管理システムとして次のような点が挙げられている。

- 分散型のバージョンシステムなので、ローカルでブランチの作成が可能
  集中型のバージョン管理システムではブランチの作成が一部の人しかできない。
- 今までのファイルの変更履歴などが見える。
- GitHub 上ではファイルの更新などの通知を受けることができる。
- 開発者に対して変更などの要求が可能 (pull request)

git のインストール

git はローカルにリポジトリを持つのでそれを処理する環境をインストールする必要がある。ここでは git for windows を紹介する。このソフトは次のとこからダウンロードできる。

`https://git-for-windows.github.io/`

デフォルトの設定で十分である。

「Git Bash」、「GitCMD」と「Git GUI」の 3 つがインストールされる。Unix 風のコマンドプロンプトが起動する Git Bash がおすすめである。

初期設定

ローカルでいくつかの準備が必要となる。

ユーザーネームの登録　GitHub で使用するリポジトリのユーザ名を登録は次のコマンドで行う。

```
git --config --global user.name "Foo"
```

ここでは `Foo` がユーザ名となる。

メールアドレスの登録　GitHub からの通知を受けるメールアドレスは次のコマンドで行う。

```
git --config --global user.email "Foo@example.com"
```

SSH Key の登録　GitHub との接続には SSH による通信で行う。この通信はは公開鍵暗号方式で行うので、キーを生成する必要がある。キーの生成は次のコマンドで行う。

```
ssh-keygen -t rsa -C "Foo@example.com"
```

`-C` のオプションはコメントである。

この後でキーを保存するフォルダが聞かれるが、デフォルトでかまわない。その後、passphrase の入力が求められるので、何かの文字列を入力する。マニュアルによれば passphrase はパスワードのようなもので、10 から 20 文字の長さで大文字、小文字、数字と記号が混在することが推奨されている。再度、同じものの入力が求められ、一致すれば通信のためのキーが生成される。

指定したフォルダの `.ssh` の下に `id_rsa`(秘密鍵) と `id_rsa.pub`(公開鍵) の 2 つのファイルが作成される。

GitHub のアカウントの取得

ローカルだけで開発をするのであればアカウントは必要ないが、データを交換するのであればアカウントを取ることを勧める。有料で使用するのでなければこのためのアカウントは新たにとることを勧める。

作成した公開鍵の内容をアカウントで設定する必要がある。

アカウントを取ったら GitHub 上でテストのプロジェクトを中身が空で作成する。

リポジトリの作成と運用

リポジトリの初期のブランチ名は `master` となっている。

コマンド	パラメータ	説明
`git init`		プロジェクトの初期化
`git add`	ファイルリスト	プロジェクトへのファイルの登録
`git commit`	`-m` コメント	リポジトリへの変更の登録
`git remote add`	短縮名 リポジトリ	リポジトリの短縮名の登録
`git push`	短縮名 ブランチ名	ブランチへの変更の登録
`git clone`	リポジトリ名	リモートのリポジトリのコピーを作成
`git pull`	リポジトリ名 ブランチ名	リモートのリポジトリのコピーを作成

通常は次の手順で新しいプロジェクトを構成する。

1. `git init` でリポジトリの初期化を行う。新しいリポジトリーは前もって直接 github に作成しておく。さらに次のコマンドでプロジェクト名の短縮名を登録するのが良い。

   `git remote add origin git.github.com:<userId>/<リポジトリー名>/<プロジェクト名>.git`

   その後、`git pull <短縮名> <ブランチ名>` を用いてコピーを作る。

2. または、次のコマンドで GitHub からリポジトリのコピーを取得する。

   `git clone git.github.com:<userId>/<リポジトリー名>/<プロジェクト名>.git`

3. ファイルを編集する。

4. 編集したファイルを `git add` でステージする。

5. `git commit` でその時点での変更を登録する。`-m` オプションで簡単なコメントをつけることを忘れないこと。このオプションをつけないと `vim` というテキストエディタが立ち上がり、コメントの編集を行うこととなる。

6. `git push` リモート名 ブランチ名 で変更をリモートのリポジトリに反映される。これが拒否された場合には誰かが `push` しているのでその変更を `pull` してから調整して再度 `push` する。

ファイルの状態については次のことに注意する必要がある。

- リポジトリ内のファイルは追跡されている (tracked) ものと追跡されていない (untracked) に分けられる。

- ファイルの状態として変更されていない (unmodified)、変更されている (modified) とステージされている (staged) の 3 つの状態がある。
- modified から staged にするコマンドが `git add` である。このコマンドを新規にリポジトリに追加するという意味にとらえないことが必要である。

第15回　JavaScriptライブラリーの配布

15.1　jQueryのコードを読む

次のリストはjQuery-3.2.1.jsの冒頭の部分である[1]。

```
 1 /*!
 2  * jQuery JavaScript Library v3.2.1
 3  * https://jquery.com/
 4  *
 5  * Includes Sizzle.js
 6  * https://sizzlejs.com/
 7  *
 8  * Copyright JS Foundation and other contributors
 9  * Released under the MIT license
10  * https://jquery.org/license
11  *
12  * Date: 2017-03-20T18:59Z
13  */
14 ( function( global, factory ) {
15
16         "use strict";
17
18         if ( typeof module === "object" && typeof module.exports === "object" ) {
19
20                 // For CommonJS and CommonJS-like environments where a proper ‘window‘
21                 // is present, execute the factory and get jQuery.
22                 // For environments that do not have a ‘window‘ with a ‘document‘
23                 // (such as Node.js), expose a factory as module.exports.
24                 // This accentuates the need for the creation of a real ‘window‘.
25                 // e.g. var jQuery = require("jquery")(window);
26                 // See ticket #14549 for more info.
27                 module.exports = global.document ?
28                         factory( global, true ) :
29                         function( w ) {
30                                 if ( !w.document ) {
31                                         throw new Error( "jQuery requires a window with a document" );
32                                 }
33                                 return factory( w );
34                         };
35         } else {
36                 factory( global );
37         }
38
39 // Pass this if window is not defined yet
```

[1]元来のソースどのインデントはタブでつけているがここでは空白2文字に変更してある。また、一部の空行も省略した。

```
40 } )( typeof window !== "undefined" ? window : this, function( window, noGlobal ) {
41
42 // Edge <= 12 - 13+, Firefox <=18 - 45+, IE 10 - 11, Safari 5.1 - 9+, iOS 6 - 9.1
43 // throw exceptions when non-strict code (e.g., ASP.NET 4.5) accesses strict mode
44 // arguments.callee.caller (trac-13335). But as of jQuery 3.0 (2016), strict mode should be common
45 // enough that all such attempts are guarded in a try block.
46 "use strict";
```

このリストでは詳細なコメントが付いていて比較的読みやすい。コードから次のことがわかる。

- このバージョンでは strict モードで動作する (15 行目と 41 行目)
- クロスブラウザ対策が行われている (37 行目から 40 行目のコメント)。
- 36 行目で定義されている関数の仮引数に window が使われている。

次のリストはこの部分の最小化をした部分のリストである。元来は改行が入っていないが、対応をわかりやすくするために改行を入れてある。

```
 1 /*! jQuery v3.2.1 | (c) JS Foundation and other contributors | jquery.org/license */
 2 !function(a,b){"use strict";
 3   "object"==typeof module&&"object"==typeof module.exports?
 4     module.exports=a.document?
 5       b(a,!0):function(a){
 6         if(!a.document)throw new Error("jQuery requires a window with a document");
 7         return b(a)}:
 8  b(a)}
 9 ("undefined"!=typeof window?window:this,function(a,b){
10     "use strict";
```

短縮化では次のことを行っていることがわかる。

- 各関数内で変数名は 1 文字から始めている。
- 関数の仮引数も a から付け直している。
- JavaScript の固有の関数は当然のことながら変換されていない。
- このライブラリーは一つの関数を定義して、その場で実行している。14 行目の function() の前に (がついている。
- 短縮化されたコードではこの部分が!function() となっている。関数オブジェクトを演算の対象とすることでその場で実行する。
- そのほかにもキーワード true の代わりに!0 としている。
- if(){}else{} 構文は?に置き換えている。

15.2 JavaScript ファイルの短縮化

15.2.1 JavaScript ファイルの短縮化について

JavaScript ファイルの短縮化を行う方法はいくつかあるが、ここでは、Google が提供する Closure Compiler を紹介する[2]。

このサービスは次のように説明されている。

> The Closure Compiler is a tool for making JavaScript download and run faster. Instead of compiling from a source language to machine code, it compiles from JavaScript to better JavaScript. It parses your JavaScript, analyzes it, removes dead code and rewrites and minimizes what's left. It also checks syntax, variable references, and types, and warns about common JavaScript pitfalls.

これによると、元来の JavaScript のコードをより良い JavaScript のコードに変換し、簡単な警告を表示するようである。

使い方については次のように書かれている。

> You can use the Closure Compiler as:
>
> - An open source Java application that you can run from the command line.
> - A simple web application.
> - A RESTful API.
>
> To get started with the compiler, see "How do I start" below.

ここでは 2 番目にある Web アプリケーションで行う。

このサイトは `http://closure-compiler.appspot.com/home` である (図 15.1)。左側のテキストボックスにコードを張り付けて、「Compile」のボタンを押せばよい。

15.2.2 短縮化の例

ここでは実行例 8.1にある `event.js` で短縮化の効果を見ることにする。

図 15.2はその結果である。

画面の右のほうで、元のファイルの大きさが 1.53KB であったのに対し、短縮化の結果が 1.06KB となったことがわかる。

短縮化の効果を見るために、対応するコードのところに改行を入れたものが次のリストである。

```
1  window.onload=function(){var b=document.getElementById("Squares"),
2    e=document.getElementById("select"),
3    g=document.getElementById("colorName"),
4    f=document.getElementById("radio"),
5    h=document.getElementById("Set"),
```

[2] `https://developers.google.com/closure/compiler/` 2016 年 11 月 23 日参照

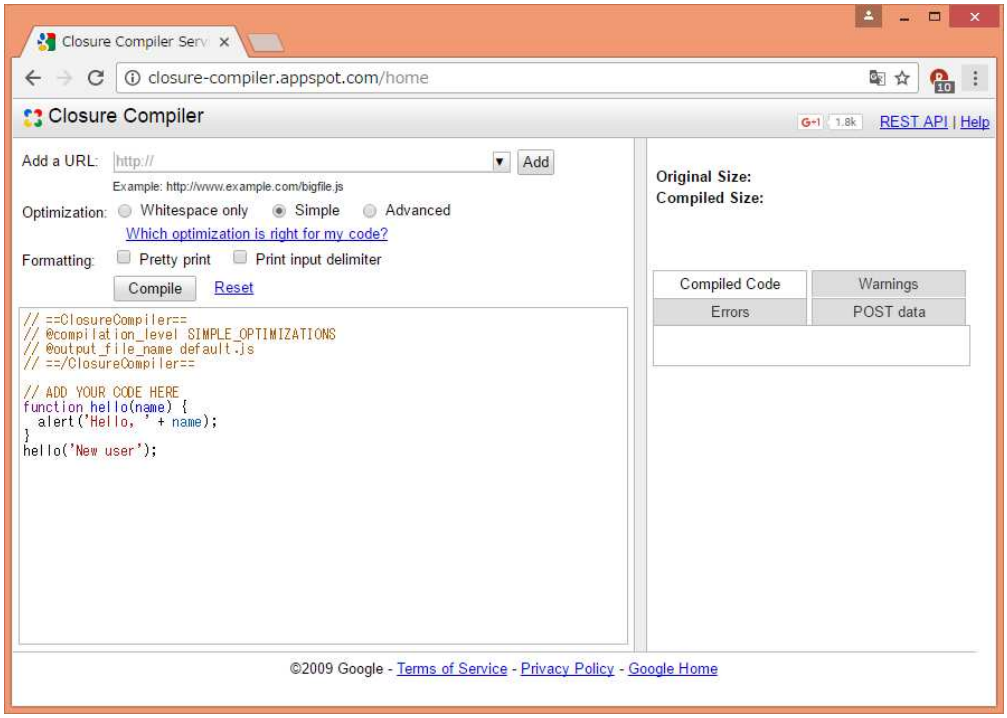

図 15.1: Closure Compiler のホームページ

```
6     c=document.getElementsByClassName("click"),
7     d=b.children[1];
8     b.children[0].style.background="red";
9     b.children[1].style.background="yellow";
10    b.children[2].style.background="blue";
11    e.style.fontSize="30px";
12    b.addEventListener("click",function(a){
13      c[0].value=a.clientX;
14      c[1].value=a.clientY;
15      c[2].value=a.pageX;
16      c[3].value=a.pageY;
17      c[4].value=window.pageXOffset;
18      c[5].value=window.pageYOffset;
19      var b=a.target.getBoundingClientRect();
20      c[6].value=a.pageX-b.left;
21      c[7].value=a.pageY-b.top;
22      g.value=a.target.style.background;d=a.target},!1);
23      e.addEventListener("change",function(){d.style.background=e.value},!1);
```

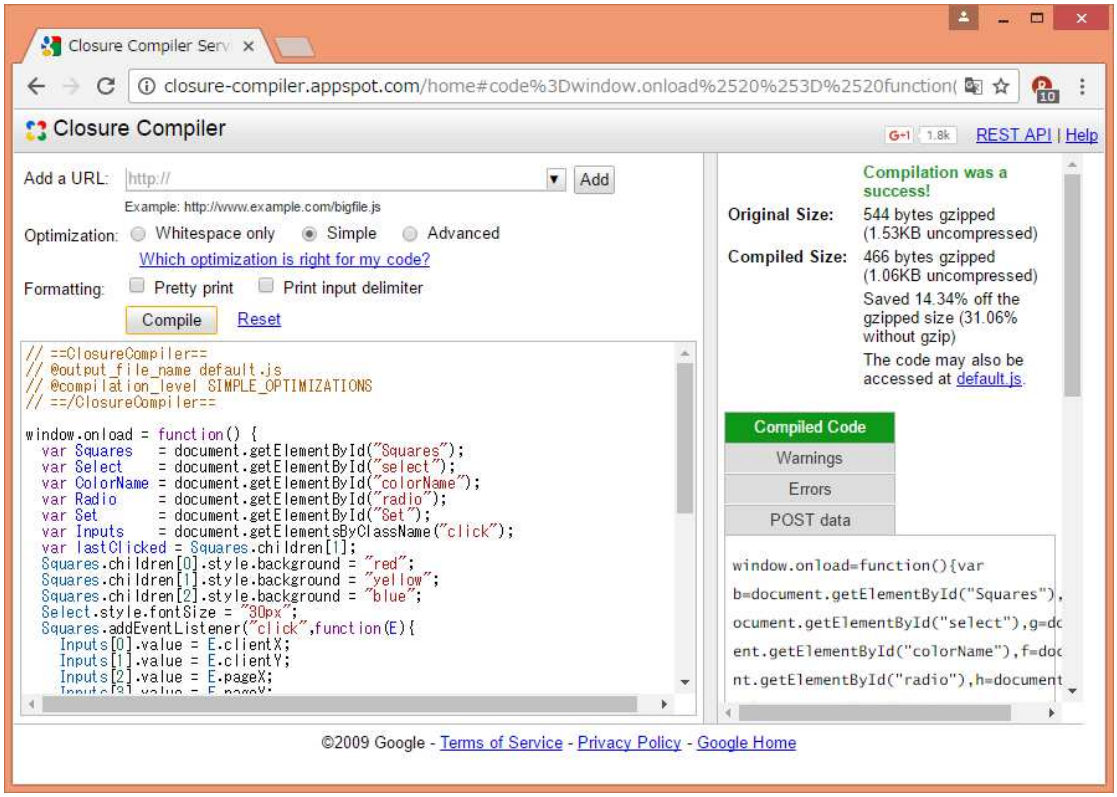

図 15.2: Closure Compiler の結果

```
24      f.addEventListener("click",function(a){
25        alert(a.target.tagName);
26        "DIV"===a.target.tagName&&(a.target.firstChild.checked=!0);
27        console.log("----"+f.value);
28        d.style.background=f.querySelector("input:checked").value},!1);
29      h.addEventListener("click",function(){d.style.background=g.value},!1)};
```

ここで行われている短縮化は次のとおりである。

- 空白の除去
- 変数宣言をまとめる。
- 変数名の単純化
- いくつかの定数を短いものに変える。

  `false` は`!1` に変えている (5 文字から 2 文字)。
- `if` 文の簡略化

  短縮化後の 29 行目は `if` 文であったものが論理式の`&&`で置き換えられている。

この一方で document.getElementById などはシステムで定義されているのでそのまま短縮化、共通化できない。この部分も短くするためにはソースコードに工夫が必要である。

しかし、document.getElementById をラップする関数を定義してプログラムコードを短くしてもうまくいかない。

```
function getElm(N){
  return document.getElementById(N);
}
var Squares   = getElm("Squares");
var Select    = getElm("select");
      ...
```

図 15.3がコンパイルの結果である。

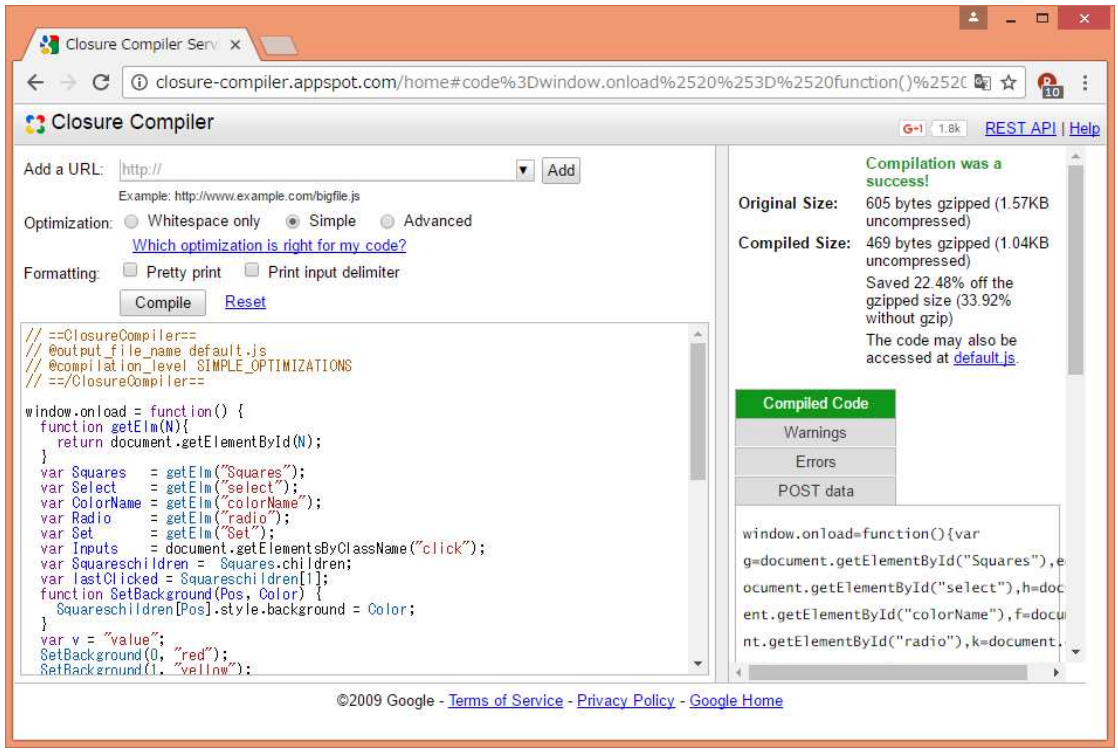

図 15.3: Closure Compiler の結果 (2)

コンパイルの結果を見ると、ラップした関数は消えていて、もとの document.getElementById に戻っている。

これを避けるためには関数の引数に document と document.getElementById を渡すことで解決できる。このとき、引数で渡された document.getElementById の実行時の this が document にならないので、call を用いて this を document にする必要がある。また、関数は 1 度実行する必要がある。

次のリストはそのように書き直したものである。

```
var Squares, Select, ColorName, Radio, Set;
(function(document,getElementById){
    Squares   = getElementById.call(document,"Squares");
    Select    = getElementById.call(document,"select");
    ColorName = getElementById.call(document,"colorName");
    Radio     = getElementById.call(document,"radio");
    Set       = getElementById.call(document,"Set");
      })(document,document.getElementById);
```

図 15.4がその結果である。コンパイル後の結果が 1KB とわずかに小さくなっていることがわかる。

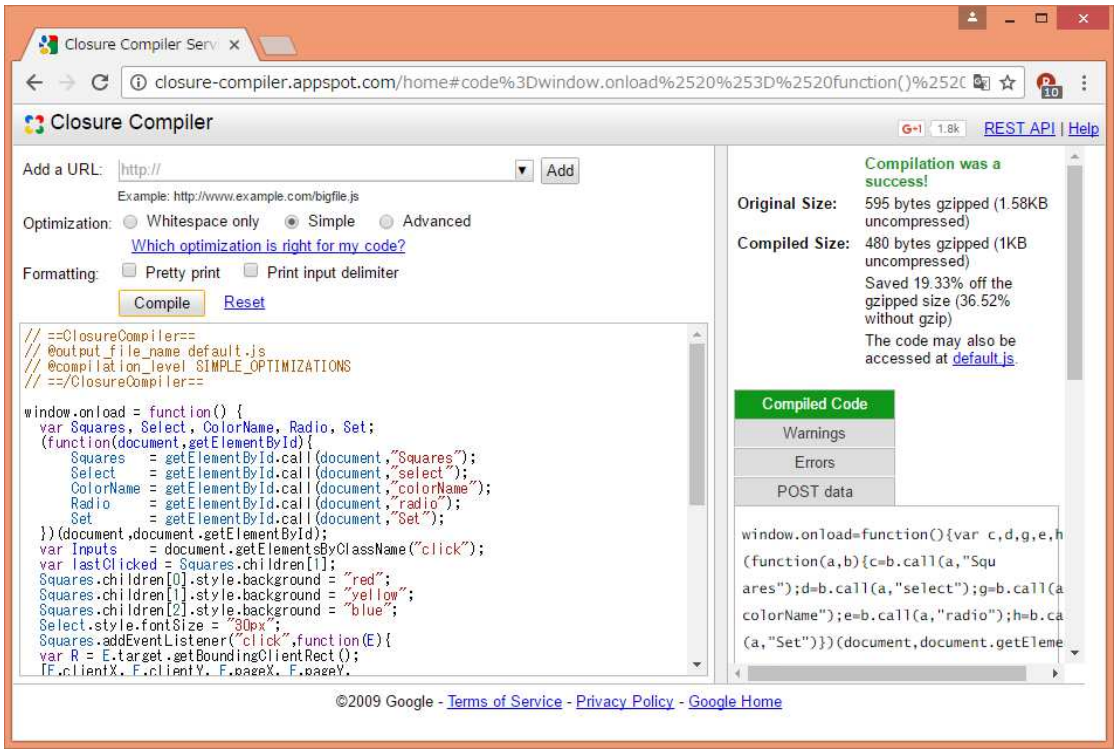

図 15.4: Closure Compiler の結果 (3)

この部分の短縮化のコードは次のようになっている。

```
var c,d,g,e,h;
  (function(a,b){c=b.call(a,"Squares");d=b.call(a,"select");
  g=b.call(a,"colorName");e=b.call(a,"radio");
  h=b.call(a,"Set")})(document,document.getElementById)
```

このようにシステムで定義されたメソッドやプロパティを関数の引数として渡すと短縮化の効率が上がる。また、これによりソースコードの理解が難しくなる。

15.3　Web サイトの効率化

15.3.1　Contents Deliverly Network(CDN)

jQuery の短縮化はファイルサイズを減らすことで Web サイトの負荷も減らしている。さらに、このようなファイルはいろいろなサイトで使用されているのでクライアント側でもキャッシュしておけばダウンロードの回数を減らせば、回線の負荷が減る。このためには、ライブラリーを置いておくサイトを用意し、そこのファイルを参照すればよい。このような目的のサイトを Contents Deliverly Network(CDN) と呼ぶ。jQuery の場合は `https://code.jquery.com/jquery-3.1.1.min.js` が短縮化されたファイルの CDN の一つである。

15.3.2　CSS Sprite

Yahoo Japan のトップページには小さな画像がたくさんある。ブラウザは表示しようとするページに画像があれば、そのページのサイトに画像を要求しに行く。したがって、表示する画像が多いとその回数分だけ、サーバーと通信が行われる。このことはサーバーに負荷がかかる。これを避けるためにブラウザは最近ダウンロードしたファイルを保存して、同じものが要求されたときには、サーバーに要求しないで、保存してあるファイルを使用する (キャッシュの利用)。

この機能を利用してサーバーは、小さな画像が複数含まれる単一の大きな画像を用意し、その一部だけをページに表示するページを用意する。この方法で画像を表示するためには CSS の background 機能を利用する。これを CSS Sprite と呼ぶ。

Yahoo Japan のソースコードで `background-image` をキーワードに検索すれば CSS Sprite で使用されている画像が見つかるであろう。

なお、Yahoo Japan などのアクセスが多いサイトではサーバーとの通信の回数を減らす目的で CSS ファイルや JavaScript のファイルは外部ファイルにしていないので合わせて確認してほしい。