ソフトウェア開発

– 授業配布版 –

平野 照比古

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。
- 演習は原則行わない。出された課題は自宅で行うこと。また、レポートの提出を必ずすること。
- パソコンを授業に持参する必要がある場合はその旨、前回の授業で指示する。
- 11 年度以前の学生に対しては「アルゴリズムとデータ構造」の読み替え科目となっている。
- 復習用の課題を必ず行うこと。次回の授業の開始時に確認の小テストを行うことがある。
- 最終的な成績は試験を行う。
- 配布資料等は `http://www.hilano.org/hilano-lab` で公開する予定

1.2　参考図書

1. D. Crockford, JavaScript: The Good Parts 「良いパーツ」によるベストプラ クティス, 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方法
第 5 回	オブジェクト属性と継承 プロトタイプによる継承など
第 6 回	正規表現と文字列の処理
第 7 回	正規表現の利用例
第 8 回	DOM の利用 HTML 文書の例、CSS と DOM の基礎
第 9 回	イベント処理 イベントモデルとイベント処理の例
第 10 回	PHP 超入門 PHP に関する簡単なプログラムの例
第 11 回	サーバーとのデータの交換 (1) PHP 入門の続きとサーバーとのデータ交換の基礎
第 12 回	サーバーとのデータの交換 (2) サーバーとのデータの交換と Ajax の基礎
第 13 回	jQuery DOM の処理を簡単にするライブラリーの紹介
第 14 回	jQuery のコード jQuery のコードの短縮化についての解説
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。このテキストでは JavaScript の計算機言語としての特徴について述べ、その後、Web ブラウザ内での処理について解説を行う。しかしながら、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。Opera の開発者用ツール、FireFox の Web 開発、Chrome のデベロッパーツール、Internet Explorer

[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のでここでは取り上げない。

の開発者ツールなどがそうである。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[2]。このときに表示されるタブには次のようなものがある[3]。

- **Elements** HTML の構造 (正式には 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だけではなく、JavaScript のプログラムが実行できる。
- **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。

1.5 strict モードについて

ECMAScript の最新版では `strict` モードと呼ばれる厳密な解釈をするモードが導入されている。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。

1.6 第 1 回目復習課題

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を確認しなさい。また、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。

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] Opera は「F12」に、Internet Explorer は「Control + Shift+I」に対応していないようである。
[3] この例は Chrome のものの一部である。

```
 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   var 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 であることを宣言している。
- 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
- 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
- 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。

[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が取り扱うデータ

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。

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`	浮動小数点数だけ
`String`	文字列型、1 文字だけのデータ型はない。ダブルクオート (") かシングルクオート (') で囲む。
`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

変数や値の型を知りたいときは `typeof` 演算子を使う。

2.1.2　Number 型

JavaScript で扱う数は 64 ビット浮動小数点形式である。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。`0` で始まりそのあとに `x` または `X` が来ない場合には 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。

特別な Number Number 型には次のような特別な Number が定義されている。

- `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。計算ができなかった場合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。

2.1.3 String 型

文字列に関する演算子としては、2 つの文字列をつなげる連接演算子がある。JavaScript では +を用いる。詳しくは 2.4を参照のこと。

文字列に関する情報 (プロパティ) や操作 (メソッド) として次のものがある。

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)`	`needle` が与えられた文字列内にあればそれが初めて現れる位置を返す。`start` の引数がある場合には、指定された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)`	`needle` が与えられた文字列内にあればそれが一番最後に現れる位置を返す。`start` の引数がある場合には、指定された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` で与えられた文字列 (または正規表現) で与えられた文字列を分けて配列で返す。セパレーター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)`	与えられた文字列の `start` から `end` の前の位置までの部分文字列を返す。
`slice(start,end)`	与えられた文字列の `start` から `end` の前の位置までの部分文字列を返す。値が負の場合には文字列の最後から数えた位置を表す。

この表に現れる正規表現については後日解説をする。

実行例 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".slice(-3);`
10. `"0123456789".substring(3,5);`
11. `"0123456789".slice(3,5);`
12. `"0123456789".slice(3,-3);`

2.1.4 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。

2.1.5 `undefined`

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。

2.1.6 `null`

`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。

2.2 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 `var` で行う。
- 宣言時に初期化ができる。
- 非 `strict` モードのときは変数は宣言をしなくても使用できる。初期化していない変数を演算の対象として使用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。

```
var a = [];
var b = [1,2,3];
```

a は空の配列で初期化されている。b は　b[0]=1,b[1]=2,b[2]=3 となる配列で初期化されている。次のことに注意する必要がある。

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

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

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。配列をスタックとして利用できる。
push(i1,i2,...)	引数で渡された要素を配列の最後に付け加える。配列をスタックやキューとして利用できる。
shift()	配列の最初の要素を削除し、その値を返す。配列をキューとして利用できる。
slice(start,end)	start から end の前の位置にある要素を取り出した配列を返す。元の配列は変化しない。
splice(start,No,i1,i2,...)	start の位置から No 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。元の配列を変更する。

実行例 2.2 次の実行結果を確かめなさい

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

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 のときだけ、数の和をとる。どちらかが数でもう一方が文字列の場合は数を文字列に直して、文字列の連接を行う。

```
1+2  => 3
1+"2" => 12
```

- 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  => "23"
"2"-0 +3 => 5
"2"*3    => 6
"2"*"3"  => 6
"f" *2   => NaN
"f" *2   => NaN
"0xf"*2  => 30
"0o10"*2 => 16
"010"*2  => 20
```

- 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
```

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元の値が Boolean 型に変換されてから実行される。次の値は false に変換される。

- 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。

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

では変数 a の値は 1 のままである。

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。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。

```
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。

また、NaN == NaN の結果は false である。そのために値が NaN であるかどうかを調べる関数 (isNaN()) がある。

2.5 組み込みオブジェクト

2.5.1 組み込み関数

JavaScript に初めから組み込まれている関数としては次のものがある。

- `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 `true`、そうでなければ `false` 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 URL エンコードする。Google の検索で日本語を含むものを対象とすると、アドレスバーに%で始まる文字列が見えるが、これが URL 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。

このほかにも関数があるが、省略する。

2.5.2 Date オブジェクト

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。
引数は、年、月、日、時、分、秒、ミリ秒の順で設定される。

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

省略された引数は `0` が設定されたものとみなされる。月の値は `0` が 1 月を表す。上記の例では月の値 `10` となっているので 11 月に設定されている。
引数がなく使用すると、現在の日時を設定したものと解釈される。

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

なお、`Date` オブジェクトの `getTime()` メソッドで得られるタイムスタンプの値は世界標準時 (UTC) において 1970 年 1 月 1 日 0 時が基準となっていて単位はミリ秒である。

表 2.4: 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
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

課題 2.1 次の日時を求める式を答えよ。

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

2.5.3 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 $\tan x$

2.6　第 2 回目復習課題

次の式の評価結果を求めなさい。

式	結果	理由
`5%3`	2	5 を 3 で割った余り
`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+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"`	2 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" == []`	`false`	文字列`"false"`は空文字ではないので `true` に変換される。
`[] == []`	`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`	両方とも `false` に変換されてから比較される。
`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 における関数はいろいろな目的のために利用される。今回の授業では次のことについて説明する。

- 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
- JavaScript 内での変数のスコープ
- 関数も単なるオブジェクト

さらに次の事柄についても説明する。

- 無名関数
- コールバック関数
- 自己実行可能関数
- 関数内で定義された関数
- 関数を返す関数
- クロージャ

関数はオブジェクトを作成するときのも用いられるが、その解説は次回で行う。

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 `sum()` という関数を定義している例である。

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

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。この場合は sum が関数の名前になる。
- 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。戻り値がない場合には戻り値として undefined が返される。

実行例　次の部分はこの関数の実行例である。

```
>sum(1,2)
3
>sum(1)
NaN
>sum(1,2,3)
3
```

- 引数に 1 と 2 を与えれば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 (この場合には b) には undefined が渡されるためである。1+undefined の結果は NaN になる。
- 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、次の例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。

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

課題 3.1 ここで定義した関数 sum(a,b,c) について上の実行例と同じことをしたら結果がどうなるか確認しなさい。

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。

```
1  function func1(a){
2    a = a*2;
3    return 0;
4  }
5  function func2(a){
6    a[0] *=2;
7    return 0;
8  }
```

- func1() では仮引数 a の値を 2 倍している。これを次のように実行すると、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を関数の引数で渡すと値そのものが渡される (値渡し)。

  ```
  >a = 4;
  4
  >func1(a);
  0
  >a;
  4
  ```

- func2() の仮引数は配列が想定してある。この先頭の値だけ 2 倍される関数である。これに配列を渡すと、戻ってきたとき配列の先頭の値が変化している。つまり、プライミティブ型以外では仮引数の渡し方が参照渡しであることがわかる。

  ```
  >a = [1,2,3];
  [1, 2, 3]
  >func2(a);
  0
  >a;
  [2, 2, 3]
  ```

3.2.3　aruguments について

JavaScript では引数リストで引数の値などが渡されるほかに arguments という配列のようなオブジェクトでもアクセスできる。このオブジェクトは非 strict モードと strict モードでは扱いが異なる部分がある。

- 引き渡された変数の数は length で知ることができる。

```
function sumN(){
  var i, s = 0;
  for(i = 0; i <arguments.length;i++) {
    s += arguments[i];
  }
  return s;
}
```

実行例は次のとおりである。

```
>sumN(1,2,3,4);
10
>sumN(1,2,3,4,5);
15
```

- 引数があっても無視できる

```
function sumN2(a,b,c){
  var i, s = 0;
  for(i = 0; i <arguments.length;i++) {
    s += arguments[i];
  }
  return s;
}
```

この例では引数が 3 個より少なくても正しく動く。実行例は次のとおりである。

```
>sumN2(1,2,3,4,5);
15
```

- 非 strict モードでは仮引数と arguments は対応していて、片方を変更しても他の方も変更される。実行例は次のとおりである。

```
function sum2(a, b){
  var c;
  a *= 3;
```

```
  console.log(arguments[0]);
  return a + b;
}

>sum2(1,2,3,4,5);
3
5
```

一方、strict モードでは argument は仮引数の静的なコピーが保持されるので、a *=3 により arguments[0] の値は変更されない。

```
>sum2(1,2,3,4,5);
1
5
```

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から参照できるかという概念である。JavaScript では次の特徴がある。

1. 非 strict モード変数は宣言しなくても使用できる。
2. 関数内で var により明示的に定義された変数はその関数内で有効となる。
3. 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。
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グローバルとなる。

実行例 3.2 変数のスコープを次の例で確かめる。

```
 1  var S = "global";
 2  function func1(){
 3    console.log(S);
 4    return 0;
 5  }
 6  function func2(){
 7    console.log(S);
 8    var S = "local";
 9    console.log(S);
10    return 0;
11  }
12  function func3(){
13    var S = "local";
14    func1();
```

```
15    return 0;
16  }
17  function func4(){
18    var 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 が宣言されていて"global" という文字列の値に初期化されている。
- 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 は宣言されていないので 2 行目で定義したグローバル変数が参照される。

  この関数内で変数

  ```
  >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 行目でローカル変数として宣言されている。
  - したがって、7 行目の変数 S はローカル変数となる (3参照)。
  - この段階ではローカル変数 S には値が代入されていないのでその値は undefined となる。
  - 9 行目の出力は 8 行目で定義された値となる。

    ```
    >func2();
    undefined
    local
    0
    ```

- 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() の実行の際は、もともと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。
  - var 宣言がないので変数 func5 はグローバル変数となる (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の変数 S はこの関数が func4() の中で定義されているので、18 行目の変数 S が参照される。この状況についてはクロージャの項でより詳しく説明する。

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

課題 3.2 実行例 3.2のプログラムにおいて、次のように書き直したらどうなるか答えなさい。なお、示している修正は問題ごとに元のファイルに対して行うものとする。

1. 8 行目のキーワード var を省略したのち、func2() を実行し、そのあとで変数 S の値を出力する。
2. 13 行目のキーワード var を省略したのち、func3() を実行する。
3. 19 行目の先頭にキーワード var を付け、func4() を実行したのち、func5() を実行する。

3.2.5　スコープチェイン

関数の中で関数を定義すると、その内側の関数内で var で宣言された変数のほかに、一つ上の関数で利用できる (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。

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

- 関数 func1() ではグローバル変数である G1 と G2、仮引数の a とローカル変数 b と c が利用できる。
- 関数 func2() ではグローバル変数である G1、func1() の仮引数の 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。

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。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。

3.3　JavaScript における関数の特徴

JavaScript 関数ではほかの言語では見られない関数の取り扱い方法がある。

3.3.1　関数もデータ

関数もデータ型のひとつなので、関数の定義を変数に代入することができる。課題 3.2の 19 行目以降で関数オブジェクトを変数に代入している。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ことも可能である。

3.3.2　無名関数とコールバック関数

課題 3.2の 19 行目以降の関数オブジェクトは関数の引数として直接渡すこともできる。このとき、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、いろいろなイベント (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。このように関数に引数として渡される関数のことをコールバック関数という。

実行例 3.3 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  var T = new Date();
2  window.setTimeout(
3    function(){
4      var NT = new Date();
5      if(NT-T<10000) {
6        console.log(Math.floor((NT-T)/1000));
7        window.setTimeout(arguments.callee,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。単位はミリ秒である。
- 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。
- この関数内で一定の条件のときはこの関数を呼び出す。この関数に名前はない (3 行目)。
- 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)。非 strict モードでは arguments を仮引数としてもつ関数を arguments.callee で呼び出すことができる。つまり自分自身を呼び出せることとなる。strict モードでは arguments.callee は使用できないので 3 行目の関数に名前を付けてその関数名を arguments.callee の代わりに用いる必要がある。

なお、関数の宣言を function foo(){...} でする代わりに、var foo = function()...{} と無名関数を利用して定義してもよい。

課題 3.3 次のプログラムは何を計算するか答えよ。

```
var f = function(n) {
  if( n<=1) return 1;
  return n*arguments.callee(n-1);
}
```

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。たとえば次のコードを考える。これは課題 1.2の関数の中身である。

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

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しまっている。これを避けるためには定義した関数をその場で実行できる機能があればよい。これを実行するためには次のように記述する。

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

この様に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を汚さない (余計な変数などを残さない) 手段として用いられる。

3.3.4　関数を返す関数

関数もデータなので、ある関数の戻り値として関数自体を返すことも可能である。使用例は次節を参照のこと。

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いでいることから、親の関数のローカル変数の参照が可能となる。

このように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ではより実用的なものと理解できるであろう。

実行例 3.4 次の関数を考える。

```
1  function f1() {
2    var 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。

```
>ff= f1();
() {
 return n++;
 }
```

- 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();
0
>ff();
1
>ff();
2
```

何回か実行すると戻り値が順に増加していることがわかる。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();
() {
    return n++;
  }
>ff();
2
>ff2();
0
```

次の章では関数を用いてオブジェクトを作成することができることを紹介する。このとき、同じ関数から作成されるオブジェクトに共通の変数を使うことはこの方法ではできない。

3.4.2　クロージャの例 – 無名関数をその場で実行する

例 3.4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 3.5 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。

```
var foo = (function () {
  var 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ない。
- 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();
0
>foo();
1
>foo();
2
```

実行例 3.6 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[1]

```
1  function f2() {
2    var a = [];
3    var 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[2]。

```
>funcs=f2()
[function a.(anonymous function)(), function a.(anonymous function)(),
                function a.(anonymous function)()]
>funcs[1]()
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 3.7 この不具合を直すためには、その値を関数の引数に (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    var a = [];
3    var i;
4    for(i=0; i<3; i++) {
5      a[i] = (function(x){
6        return function() {
7          return x;
8        }
9      })(i);
```

[1] 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。

[2] ここでの出力例は Opera によるものである。ブラウザによっては出力が異なる。

```
10    };
11    return a;
12  }
```

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]();
0
>funcs2[1]();
1
```

- 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。
- この関数の戻り値は、無名関数で与えられた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。

この書き方が有効になるのは関数をオブジェクトのコンストラクタとして使用し、インスタンスに通し番号を付けるときなどに必要となる。

課題 3.4 次の関数のたいして `var a=f4();` とした後、`a[1](0);` などの値がどうなるか調べなさい。また、そのような結果になる理由を考えなさい。

```
function f4() {
  var a = [], b;
  var i;
  for(i=0; i<3; i++) {
    b=[i];
    a[i] = function() {
      return b;
    };
  }
  return a;
}
```

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。これにたいし、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。

```
var 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` の値がまたオブジェクトとなっている。
- 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。
  - 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]
hometown 北海道
mother aaa
undefined
```

最後の undefined は for ループの戻り値である。

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。

4.2 コンストラクタ関数

オブジェクトを定義する方法としてはコンストラクタ関数を使う方法がある。

実行例 4.2 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 (インスタンス) を構成している。

```
function Person(){
  this.name = "foo";
  this.birthday = {
    year : 2001,
    month : 4,
    day : 1
  };
  this["hometown"] = "神奈川";
}
```

- 通常、コンストラクタ関数は大文字で始まる名前を付ける。
- そのオブジェクト内にメンバーを定義するために、this をつけて定義する。ここでは、前の例と同じメンバー名で同じ値を設定している。
- この関数には return がないことに注意すること。
- この関数を用いてオブジェクトを作成するためには、new をつけて関数を呼び出す。

  ```
  >var person = new Person();
  undefined
  ```

- 元来、戻り値がないので undefined が表示されているが、オブジェクトは作成されている。
- 前と同じ文を実行すれば同じ結果が得られる。

ここの例はコンストラクタ関数に引数がないが、引数を持つコンストラクタ関数も定義が可能である。これにより同じメンバーを持つオブジェクトをいくつか作る必要がある場合にプログラムが簡単になる。

実行例 4.3 次の例はコンストラクタ関数を new を用いないで実行した場合である。

```
>p = Person();
undefined
>p.name
VM88:2 Uncaught TypeError: Cannot read property 'name' of undefined(…)
>p;
undefined
>name;
"foo"
>window.name;
"foo"
>name === window.name
true
```

- この関数は戻り値がないので、undefined が変数 p に代入される。
- したがって、p.name も当然定義されていない。
- このとき、キーワード this が指すのはグローバルオブジェクトとなる。
- 現在の実行環境はブラウザ上なので、このときのグローバルオブジェクトは window である。
  - このとき、グローバル変数はすべてグローバルオブジェクトのメンバーとしてアクセス可能である。この例では this.name に値を代入した時点で変数 name が定義されている。
  - 最後の例からも、name と window.name が同じものであることがわかる。

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。

4.2.1 constructor プロパティ

オブジェクトが作成されると、constructor プロパティとよばれる特殊なプロパティも設定される。

- このプロパティはオブジェクトを作成したときに使われたコンストラクタ関数を返す。実行例 4.2で確認すると次のようになる。

```
>var p = new Person();
undefined
>p.name;
"foo"
>p.constructor;
function Person(){
```

```
  this.name = "foo";
  this.birthday = {
    year : 2001,
    month : 4,
    day : 1
  };
  this["hometown"] = "神奈川";
}
```

Opera では定義全体が表示される。

- このプロパティに含まれるものは関数なので、コンストラクタの名前を知らなくても、元と同じオブジェクトのコピーが作成できる。

```
>np = new p.constructor();
Person {name: "foo", birthday: Object, hometown: "神奈川"}
>np.constructor;
function Person(){
  this.name = "foo";
  this.birthday = {
    year : 2001,
    month : 4,
    day : 1
  };
  this["hometown"] = "神奈川";
}
```

- オブジェクトリテラルを使ってオブジェクトを作ると、組み込み関数の `Object()` コンストラクタ関数がセットされる。

```
>o = {}
Object {}
>o.constructor;
function Object() { [native code] }
```

- なお、実行例 4.1でみたように、このプロパティは `for-in` ループ内では表示されない。

4.2.2 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できる。

実行例 4.4 次の結果は Opera で 実行例 4.1をもとに、4.2.1を実行した後にさらに実行したものである。

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

Person オブジェクトが Object を継承しているために o instanceof Object が true となっている。継承については後の授業で解説する。

4.2.3　同じコンストラクタ関数からの生成されたオブジェクトに連番号を付ける

オブジェクト指向言語では同じコンストラクタ関数 (クラス) から生成されたオブジェクトに重複のない番号を付けることがある。JavaScript でそのようなことを実現するためには工夫が必要である。

次のように単純にメンバーを追加しても、コンストラクタ関数が呼ばれるごとに、変数 ID が初期化されてしまい、目的を果たすことができない。

```
function Person(){
  var ID = 0;
  this.ID = ID++;
  this.name = "foo";
}
```

これから 2 つオブジェクトを作成しても、ID がともに 0 となっていることがわかる。

```
>p1 = new Person();
Person {ID: 0, name: "foo"}
>p2 = new Person();
Person {ID: 0, name: "foo"}
```

クロージャを用いて関数ににすると一見、うまくいくように見える。

```
var Person2 = (function (){
  var ID = 0;
  return function(){
    this.ID = ID++;
    this.name = "foo";
  }
})();
```

これを実行すると次のようになる。

```
>p1 = new Person2();
Object {ID: 0, name: "foo"}
>p2 = new Person2();
Object {ID: 1, name: "foo"}
>p1.ID;
0
>p2.ID;
1
```

しかし、このプロパティは外部から変更が可能となってしまっている。

```
>p1.ID=10;
10
>p1.ID
10
```

これを避けるためにはクロージャ内の変数を参照するようなメソッドに変更すればよい。

```
var Person3 = (function (){
  var ID = 0;
  return function(){
    this.getID = function() {
      return ID++;
    }
    this.name = "foo";
  }
})();
```

このコードでは `getID()` メソッドを実行するごとに、クロージャ内の変数 `ID` が増加してしまい、失敗である。

```
>p1 = new Person3();
Object {name: "foo"}
>p1.getID();
0
>p2 = new Person3();
Object {name: "foo"}
>p2.getID();
1
>p1.getID();
2
>p1.getID();
3
```

呼ばれた時点での ID の値を保存するためには、いったん、スコープチェインを切る必要がある。

```
var Person4 = (function (){
  var ID = 0;
  return function(){
    this.getID = (function(x) {
      return function(){ return x;}
    })(ID++);
    this.name = "foo";
  }
})();
```

これで目的は達成されている。

```
>p1 = new Person4();
Object {name: "foo"}
>p1.getID();
0
>p2 = new Person4();
Object {name: "foo"}
>p2.getID();
1
>p1.getID();
0
```

しかし、これでも getID() メソッドは書き直すことが可能である。

課題 4.2 p1.getID() が設定しなおせることを確認しなさい。

4.3　ECMAScript5 のオブジェクト属性

4.3.1　オブジェクト指向言語におけるプロパティとメソッドの属性

オブジェクト指向言語ではオブジェクトのプロパティやメソッドは次のように分類されきる。

- **インスタンスフィールド**
  インスタンスごとに異なる値を保持できるプロパティ
- **インスタンスメソッド**
  クラスのすべてのインスタンスで共有されるメソッド
- **クラスフィールド**
  クラスに関連付けられたプロパティ

- **クラスメソッド**
  クラスに関連付けられたメソッド

JavaScript では関数もデータなのでフィールドとメソッドに厳密な区別はないのでフィールドとメソッドは同一視できる。また、`prototype` を用いるればクラスフィールドなども作成できる。

通常、オブジェクト指向の言語ではフィールドをかってに操作されないようにするために、フィールドを直接操作できなくして、値を設定や取得するメソッドを用意する。そのために、フィールドにアクセスするため記述が面倒になるという欠点もある。

一方、プロパティの代入の形をとっても実際はゲッターやセッター関数を呼ぶ形になっている言語も存在する。JavaScript の最新版 1.8.1 以降ではそれが可能となっている。

4.3.2 プロパティ属性

JavaScript は Ecma International が定義している ECMAScript の仕様に基づいている。2014 年現在、最新バージョンの ECMAScript7 の仕様は ECMA-262[1] で公開されている。

このバージョンではオブジェクトのプロパティやメソッドにプロパティ属性という機能が追加された。オブジェクトのプロパティの属性には表 4.1のようなものがある。

表 4.1: プロパティの属性のリスト

属性名	値の型	説明	デフォルト値
`value`	任意のデータ	プロパティの値	`undefined`
`writable`	`Boolean`	`false` のときは `value` の変更ができない	`true`
`enumerable`	`Boolean`	`true` のときは `for-in` ループでプロパティが現れる。	`false`
`configurable`	`Boolean`	`false` のときはプロパティを消去したり、`value` 以外の値の変化ができない	`false`

これにより作成したオブジェクトのプロパティのアクセス方法に制限をかけることが可能となる。

また、メソッドに関しては表 4.2のものがある。

これらの属性のうち、`get` や `set` を使うとオブジェクトのプロパティの呼び出しや変更に関して、いわゆるゲッター関数やセッター関数を意識しないで呼び出すことが可能となる。

これらの属性は `Object.defineProperty()` や `Object.defineProperties()` 関数を用いて設定する。

[1] `http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf`

表 4.2: メソッドの属性のリスト

属性名	値の型	説明	デフォルト値
get	オブジェクトまたは未定義	関数オブジェクトでなければならない。プロパティの値が読みだされるときに呼び出される	undefined
set	オブジェクトまたは未定義	関数オブジェクトでなければならない。プロパティの値を設定するときに呼び出される	undefined
enumerable	Boolean	true のときは for-in ループでメソッドが現れる。	false
configurable	Boolean	false のときはプロパティを消去したり、value 以外の値の変化ができない	false

実行例 4.5 次の例は今までの例にオブジェクト属性を付けてその効果を確認するためのものである。

```
1  var Person = (function (){
2    var ID = 0;
3    return function(name, year, month, day, hometown){
4      this.name = name;
5      var getID = (function(x) {
6        return function(){ return x;}
7      })(ID++);
8      this.birthday = {};
9      this.birthday.year = year;
10     this.birthday.month = month;
11     this.birthday.day = day;
12     this.name = name;
13     Object.defineProperty(this, "ID",
14       {get: getID,
15        enumerable:true,
16        configurable:false
17       });
18     Object.defineProperties(this.birthday,
19       {year  : {enumerable : true},
20        month : {enumerable : true},
21        day   : {enumerable : false, writable : false}
22     });
23   }
24 })();
```

- 3 行目から 23 行目までが、Person() で定義されるコンストラクタ関数である。
- このコンストラクタ関数は今まではデフォルトで与えられていたオブジェクトのメンバーを、コンストラクタ関数の引数で定義できるように変更している。
- 5 行目から 7 行目で呼び出されたときの ID の値を保存し、参照するためのローカルな関数 getID() を定義している。
- 8 行目では this.birthday をオブジェクトとして初期化し、9 行目から 10 行目でそのオブジェクトに値を設定している。
- 13 行目から 22 行目で、このオブジェクトの ID を設定している。
  - get で参照される関数を 5 行目で定義された getID 設定している (14 行目)。
  - 15 行目で for in ループで表示されるように、15 行目ではプロパティに変更ができないように defineProperty メソッドで設定している。

    defineProperty メソッドの 3 番目の引数は設定しようとするプロパティを列挙したオブジェクトになっている。
- 18 行目から 22 行目ではプロパティbirthday の各項目に必要な設定をしている。
- ID のプロパティでは set が設定されていないので、呼び出しの処理は行われない。

```
>p = new Person("foo",2001,4,1,"Japan");
Object {name: "foo", birthday: Object}
>p.ID;
0
>p.ID = 5;
5
>p.ID;
0
```

p.ID に値を代入してもエラーは起きていないので値が設定できたように見える。しかし、値を参照してみると変化がないことがわかる。

- また、p.ID は消去できない。

```
>        delete p.ID
false
```

- p のプロパティを列挙してみると、3 つが表示されることがわかる。

```
>for(c in p) console.log(c+":"+p[c]);
 name:foo
 birthday:[object Object]
 ID:0
undefined
```

- しかし、p.birthday のプロパティでは day が表示されない。

```
>for(c in p.birthday) console.log(c+":"+p.birthday[c]);
 year:2001
 month:4
undefined
```

これは 21 行目で enumerable を false に設定しているためである。

- p.birthday.year の値は変更できる。

```
>p.birthday.year = 2010
2010
>p.birthday.year;
2010
```

- p.birthday.day の値は writable が false に設定されているために変更できない。

```
>p.birthday.day = 20;
20
>p.birthday.day;
1
```

- p.birthday.day は configurable が設定されていないので消去できる。

```
>delete p.birthday.day;
true
>p.birthday.day;
undefined
```

- p.ID は configurable が false に設定されているので消去できない。

```
>delete p.ID;
false
>p.ID;
0
```

delete の結果が false となっている。

課題 4.3 実行例 4.5について次の問いに答えよ。

1. ID プロパティの get のローカルで定義された関数名を与えているが、ここは無名関数でも動くことを確認しなさい。
2. 残りの属性にも適当な属性を付けて意図したとおりに動作することを確認しなさい。
3. this.birthday の属性の一つ (以上) に configurable を false に設定したとき、birthday 属性が delete できるかどうか確認しなさい。

課題 4.4 4.2.3項の Person2 の ID プロパティにプロパティ属性を付けて、外部から変更できないようにしなさい。(ヒント:ID の値を参照するために別の変数が必要になるかもしれない。)

4.4 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法と全く同じである。

- 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 4.6 次の例は 2 つの同じ形式からなる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。

```
var 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"}]"
>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方法は `JSON.parse()` を用いる。
- 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 4.5 実行例 4.6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。

第5回　オブジェクトのプロトタイプと継承

5.1　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。

5.1.1　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 ECMAScript 5 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 5.1 実行例 4.1の例で確認する。

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 Person();
Person {name: "foo", birthday: Object, hometown: "神奈川"}
>p.toString();
"[object Object]"
>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。Array() オブジェクトには toString() が定義 (オーバーライド) されているので次のような形にしないとこのような結果が得られない。

```
>Object.prototype.toString.call([]);
"[object Array]"
```

これは、Object の prototype 属性に定義されている toString を引数のメソッドとして利用することを意味する。prototype の意味は後で説明をする。同じように他のオブジェクトで実行すると次のようになる (Opera で実行)。

```
>Object.prototype.toString.call(null);
"[object Null]"
>Object.prototype.toString.call(undefined);
```

```
"[object Undefined]"
>Object.prototype.toString.call(NaN);
"[object Number]"
>Object.prototype.toString.call(window);
"[object global]"
>window+""
"[object Window]"
```

なお、FireFox や Internet Explorer では最後の例は次のようになる。

```
>Object.prototype.toString.call(window);
"[object Window]"
>window+""
"[object Window]"
```

5.1.2　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。ECMAScript 5 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 5.2 実行例 4.1の例で確認する。

```
>p = new Person();
Person {name: "foo", birthday: Object, hometown: "神奈川"}
>p.mother = "aaa"
"aaa"
>Object.preventExtensions(p);
Person {name: "foo", birthday: Object, hometown: "神奈川", mother: "aaa"}
>p.grandmother = "AAA";
"AAA"
>p.grandmother;
undefined
>p.mother = "bbb";
"bbb"
>p.mother;
"bbb"
>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)。
- 設定後は、新しい属性が定義できていない (p.grandmother)。
- 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。また、すでにその状態になっているかどうかは、Object.isSealed() を用いる。書き込み可の属性の値は変えることができる。

実行例 5.3 実行例 4.1の例で確認する。

```
>p = new Person();
Person {name: "foo", birthday: Object, hometown: "神奈川"}
>Object.isSealed(p);
false
>Object.seal(p);
Person {name: "foo", birthday: Object, hometown: "神奈川"}
>Object.isSealed(p);
true
>p.mother = "aaa";
"aaa"
>p.mother;
undefined
>p.hometown
"神奈川"
>p.hometown = "Japan"
"Japan"
>p.hometown
"Japan"
```

- Object.seal() を実行した後、存在しない属性 p.mother は定義されていないことがわかる。
- 既存の属性の値は変更可能である。

最も強く、オブジェクトを拘束するためには、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。

実行例 5.4 実行例 5.3に引き続いて Object.freeze() を実行した結果である

```
>Object.isFrozen(p);
false
>Object.freeze(p);
Person {name: "foo", birthday: Object, hometown: "Japan"}
```

```
>p.hometown = "Tokyo"
"Tokyo"
>p.hometown;
"Japan"
```

p.hometown 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。

なお、Object.seal() や Object.freeze() の影響は、渡されたオブジェクト自身の属性にしか影響を及ぼさない。継承元のオブジェクトには影響を及ぼさない。

5.1.3 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。

ECMAScript 5 で定義されている Object.create() メソッドは引数で与えられたオブジェクトの prototype を prototype に持つオブジェクトを生成する。このメソッドに対して 2 番目の引数を与えて、新たに生成されたオブジェクトのプロパティを指定できる。

5.1.4 prototype の使用例

実行例 5.5 次の例は 4.1の属性を簡略し、いくつかのメソッドを追加したものである。

```
 1 function Person(){
 2   this.name = "foo";
 3   this.year = 2001;
 4   this.month = 4;
 5   this.day = 1;
 6   this.toString = function(){
 7     return "私の名前は"+this.name+"です";
 8   }
 9   this.showBirthday = function() {
10     return this.name+"の誕生日は"+
11       this.year+"年"+this.month+"月"+this.day+"日です";
12   }
13 }
```

- 6 行目から 8 行目ではオブジェクトを文字列に変換する必要ができたときに呼び出される toString() メソッドが定義されている。

- 9 行目から 12 行目では誕生日をわかりやすい形で表示する showBirthday() メソッドを定義している。

実行例は次のようになる。

```
>p=new Person();
Person {name: "foo", year: 2001, month: 4, day: 1}
>p +"";
"私の名前は foo です"
>p.showBirthday();
"foo の誕生日は 2001 年 4 月 1 日です"
```

このコードをみるとコンストラクタ関数が呼ばれるごとに、2 つのメソッドのコードが繰り返し使用されていることがわかる。

このように同じコンストラクタ関数から生成さるオブジェクトに対して共通するプロパティやメソッドはコンストラクタ関数の prototype に移動した方が効率が良い。

実行例 5.6 次のコードは、実行例 5.5のメソッドを prototype に移動し、さらにプロパティとして書き直したものである。

```
 1  function Person2(name, y, m, d){
 2    this.name = name;
 3    this.year = y,
 4    this.month = m,
 5    this.day = d
 6  }
 7  Person2.prototype = {
 8    toString : function(){
 9        return "私の名前は"+this.name+"です";
10      },
11    get age(){
12        var Now = new Date();
13        var Age = Now.getFullYear() - this.year;
14        if((Now.getMonth()+1) < this.month) {
15          Age--;
16        } else {
17          if((Now.getMonth()+1) == this.month &&
18             Now.getDate() < this.day) Age--;
19        }
20        return Age;
21      },
22    get birthday() {
23        return this.year+"年"+this.month+"月"+this.day+"日";
```

```
24      }
25  }
```

- prototype はオブジェクトなのでオブジェクトリテラルの形式で定義する。
- 8 行目から 10 行目では toString() メソッドを定義している。
- 11 行目から 21 行目では、age プロパティを定義している。function キーワードの代わりに get を用いることで、ゲッターとして定義していることになる。

  この関数は実行時におけるオブジェクトの満年齢を得ることができる。

  - 12 行目で実行時の日時を求めている。
  - 13 行目で実行時の年から、誕生日の年の差を求めている。
  - 14 行目では実行時の月と誕生日の月を比較し、誕生日の月を過ぎていなければ、年齢を 1 だけ減らす。

    Date.getMonth() メソッドは 0(1 月) から 11(12 月) の値を返すことに注意する。
  - 17 行目では実行時の月と誕生日の月が同じ時に、両者の日を比較して、誕生日前か判定している。
- 22 行目から 24 行目では birthday プロパティをゲッターとして定義している。

実行結果は次のとおりである。

```
>p2 = new Person2("me",1995,4,1);
Person2 {name: "me", year: 1995, month: 4, day: 1}
>p2.birthday;
"1995 年 4 月 1 日"
>p2.age;
20
>p.age=30;
30
>p2.age;
20
>Person2.prototype.constructor
Object() { [native code] }
```

- p.age にはセッターが定義されていないので見かけ上の代入を行っても無効であることがわかる。
- Person2.prototype.constructor の結果が Person2 になっていないのは、7 行目で Person2.prototype に代入しているいオブジェクトに constructor プロパティがないからである。通常はこれには問題があるので代入するオブジェクト内に次のような記述を追加するのがよい。

  ```
  constructor : Person2,
  ```

別の方法としては、Person2.prototype.constructor = Person2; の行を追加する方法もある。

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 5.1 実行例 5.6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。

5.2 継承

オブジェクトの継承とは既に存在するオブジェクト (クラス) に対して機能の追加や修正を行って新しいオブジェクト (クラス) を構成することである。JavaScript ではクラスをサポートしていないので厳密な意味での継承はできないが、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 5.7 次のリストは実行例 5.6の Person2 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  function Student(n, id, y, m, d){
2    this.name   = n;
3    this.year  = y;
4    this.month = m;
5    this.day   = d;
6    this.id    = id;
7  }
8  Student.prototype = new Person2();
9  Student.prototype.constructor = Student;
```

- name などのプロパティはオブジェクトごとに違う値をとるので Person2.rototype 内には置くことができない。したがって、それぞれを this のプロパティに格納する (2行目から 6 行目)。
- Person2 の prototype を利用するために、Student.prototype にオブジェクトを新規に作成して代入する (8 行目)。これにより、この後で Person2 のプロパティが変更されても、Student オブジェクトには影響がでない。
- Student.prototype.constructor を Student に戻しておく (9 行目)

実行結果は次のとおりである。

```
>s = new Student("me",1323300,1995,4,1)
Student {name: "me", year: 1995, month: 4, day: 1, id: 1323300}
>s.age;
20
```

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

```
>s.name
"me"
>s+"";
"私の名前は me です"
>s.constructor;
Student(n, id, y, m, d){
  this.name   = n;
  this.year  = y;
  this.month = m;
  this.day   = d;
  this.id    = id;
}
```

Person2 で定義されたメソッドが利用できていることがわかる。

課題 5.2 実行例 5.7の Student に所属学部を表示するメソッドを追加しなさい。所属学部のプロパティは追加しないで学籍番号から求めること。

5.3　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。

5.3.1　エラー処理の例

エラー処理の例をいくつか挙げる。

実行例 5.8 次のリストは、実行例 5.6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 Person2 の age の内容が以前のものと同じなので ... で省略している。

```
1  function Person(name, y, m, d){
2    if(name === "") throw new Error("名前がありません");
3    this.name = name;
4    this.year = y;
5    if(m<1 || m>12) throw new Error("月が不正です");
6    var date = new Date(y,m,0);
7    if(d<1 || d>date.getDate()) throw new Error("日が不正です");
8    this.month = m,
9    this.day = d
10 }
11 Person.prototype = {
12   toString : function(){
13       return "私の名前は"+this.name+"です";
14     },
15   get age(){
16     ... // 内容は省略
17     },
18   get birthday() {
19       return this.year+"年"+this.month+"月"+this.day+"日";
20     }
21 }
```

- 2 行目で、name が空文字であればエラーを発生させている。
- 5 行目では月の値の範囲をチェックしている。
- 6 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[1]。
- 7 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。

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}
```

- 1996 年はうるう年なので 2 月 29 日が存在する。したがって、エラーは起こらず正しく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}
```

[1] 課題 2.1 の 3 番目の問題の解答である。

- 1995 年はうるう年ではないので 2 月 29 日がない。したがって、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 5.3 実行例 5.8においてエラーチェックが完全ではないことを指摘し、それを改良しなさい。

このリストの欠点はこのオブジェクトを継承するオブジェクトに対して、継承先のオブジェクトに対してエラーチェックの部分を再び書く必要がある。これを改良したのが次のリストである。

実行例 5.9 次のリストはエラーチェックの部分を関数化して、継承先でも同じようなチェックができるようにしたものである。

```
 1  function Person2(name, y, m, d){
 2    this.checkDate(y, m, d);
 3    this.checkName(name);
 4    this.name = name;
 5    this.year = y;
 6    this.month = m,
 7    this.day = d
 8  }
 9  Person2.prototype = {
10    toString : function(){
11        return "私の名前は"+this.name+"です";
12      },
13    checkDate : function(y, m, d) {
14        var date = new Date(y,m,0);
15        if(m<1 || m>12) throw new Error("月が不正です");
16        if(d<1 || d>date.getDate()) throw new Error("日が不正です");
17      },
18    checkName : function(name) {
19        if(name === "") throw new Error("名前がありません");
20      },
21   get age(){
22      ... // 内容は省略
23       },
```

```
24    get birthday() {
25        return this.year+"年"+this.month+"月"+this.day+"日";
26      }
27  }
```

このリストではエラーチェックをする関数を作成して、それを Person2.prototype のなかに置いている。

- 2 行目と 3 行目でエラーチェックをする関数を呼び出している。この関数はエラーが起きたときはエラーを投げるので、コンストラクタ関数の制御から離れることとなる。
- それぞれのエラーチェックをする関数は 13 行目から 17 行目と 18 行目から 20 行目に記述されている。
- 実行例 5.7における Person2 継承した Student オブジェクトは次のようになる。

```
 1  function Student(n, id, y, m, d){
 2    this.checkDate(y, m, d);
 3    this.checkName(name);
 4    this.name   = n;
 5    this.year  = y;
 6    this.month = m;
 7    this.day   = d;
 8    this.id    = id;
 9  }
10  Student.prototype = new Person2();
11  Student.prototype.constructor = Student;
```

これを実行すると Person2 と同じように動作することがわかる。

```
>s=new Student("foo",1223300,1995,12,1);
Student {name: "foo", year: 1995, month: 12, day: 1, id: 1223300}
>s=new Student("foo",1223300,1995,4,0);
 Uncaught Error: 日が不正です (…)
>s=new Student("foo",1223300,1995,13,1);
 Uncaught Error: 月が不正です (…)
```

5.3.2 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。

実行例 5.10 次のリストは 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。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    var y, m, d;
3    for(;;) {
4      try {
5        y = Number(prompt("生まれた年を西暦で入力してください"));
6        m = Number(prompt("生まれた月を入力してください"));
7        d = Number(prompt("生まれた日を入力してください"));
8        return new Person2("foo", y, m, d);
9      } catch(e) {
10       console.log(e.name+":"+e.message);
11     }
12   }
13 }
```

- テストを繰り返す関数 test() が定義されている。
- 3 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 8 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。
  - ここでは年、月、日の入力を prompt 用いてダイアログボックスを表示させ、そこに入力させている。
  - 戻り値は文字列なので、Number で数に直している。
- 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 (10 行目)。

try{...}catch{} 構文については次のようなこともある。

- finally{} を付けることもできます。try{...}catch{} 内の部分は try や catch の部分が実行された後必ず呼び出される[2]。
- 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 にエラーがつかまる。

[2] この例では正常時には return が実行されるので呼び出されません。

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。

```
var pattern = new RegExp("^s");
var 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。

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。

```
var ex = /\d\d\d\d/;
undefined
```

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);
0
>"20144567".match(ex);
["2014"]
>"20144567".replace(ex,"AA");
"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。

```
>var exg = /\d\d\d\d/g;
undefined
>"20144567".search(exg);
0
>"20144567".match(exg);
["2014", "4567"]
>"20144567".replace(exg,"AA");
"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");
"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+/);
["aaaaa"]
>"aaaaab".match(/a+?/);
["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/);
["aaaaab"]
```

この例では、ab にマッチしていないことに注意すること。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/);
-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。したがって、(.)\2 は同じ文字が 2 つ続いていることを意味する。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/);
2
>"abccbcc".match(/((.)\2).*\1/);
["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っている。

```
>"abccbcckkccaaMMaa".match(/((.)\2).*\1/);
["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。この文字列には aa ではさまれた部分文字列も存在するが、g フラグが付いていないのではじめにマッチしたものだけが戻ってこない。
- 配列の残りの成分には\1 と\2 が入っている。

```
>"abccbcckkccaaMMaa".match(/((.)\2).*\1/g);
["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);
["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$/);
0
>'\'abcd"df"\''.match(/^(['"]).*\1$/);
["'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*/);
[" 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+/);
["", "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+/);
["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。

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);

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   var latlng = new google.maps.LatLng(35.486210,139.341443);
11   var myOptions = {
12     zoom: 10,
13     center: latlng,
14     mapTypeId: google.maps.MapTypeId.ROADMAP
15   };
16   var mapCanvas = document.getElementById("map_canvas")
17   var 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。利用当初の状況から API キーがなくても動作するが、API キーを取得することを勧める。取得の方法は付録に示す。

```
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>要素内に現れる。
- 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、`<div>`要素が 3 つ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 `nth-child` の `()` 内に次の式を入れた時どうなるか報告しなさい。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。

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
<title>nth-child のチェック</title>
<style type="text/css" >
li:nth-child(n){
  background:yellow;
}
</style>
</head>
<body>
  <ol>
    <li>1 番目</li>
    <li>2 番目</li>
    <li>3 番目</li>
    <li>4 番目</li>
    <li>5 番目</li>
    <li>6 番目</li>
  </ol>
</body>
</html>
```

1. `n`(ここでのリストの設定)
2. `2n`
3. `n+3`
4. `-n+2`

課題 7.2 前問のリストに対し、背景色が次のようになるように CSS を設定しなさい。

1. 偶数番目が黄色、基数番目がオレンジ色
2. 1 番目、4 番目、. . . のように 3 で割ったとき、1 余る位置が明るいグレー

3. 4 番目以下がピンク
4. 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。

- Opera と Google Chrome では開発者ツールの Elements タブで確認できる。ここで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 DOM ツリーが確認できる。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。
- IE では開発者ツールを開き、左にある「HTML」タブで同様のことができる。

7.4　DOM のメソッドとプロパティ

DOM では DOM ツリーを操作するためにメソッドやプロパティが規定されている。これらの手段を用いて DOM をサポートする文書にアクセスができる。

メソッドとはそのオブジェクトに対する操作である。次のような手段を提供している。

- 条件に合う要素または要素のリストを得る。
- 要素の属性を参照、変更ができる。
- 要素を新規に作成する。
- ある要素に子要素を追加したり、取り除いたりする。

プロパティはそのオブジェクトが持つ性質であり、それらの値を参照できる。ほとんどのプロパティは書き直しできない。

なお、ここでのメソッドやプロパティは DOM 文書で使用可能なものである。したがって、HTML 文書も `DOM` をサポートするブラウザであれば同様の方法で部分的に書き直すことが可能である。

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。なお、結果が要素のリストであるものについては配列と同様に `[ ]` 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。HTML 文書では `http://www.w3.org/1999/xhtml` を指定する[5]。

表 7.2: DOM のメソッド

メソッド名	対象要素	説　　明
`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。

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

```
<!DOCTYPE html>
<head>
<meta charset="UTF-8"/>
<title>プルダウンメニュー</title>
</head>
<body>
  <form id="menu">
    <select>
      <option value="1">1 月</option>
      <option value="2">2 月</option>
      <option value="3">3 月</option>
      <option value="4">4 月</option>
      <option value="5">5 月</option>
      <option value="6">6 月</option>
      <option value="7">7 月</option>
      <option value="8">8 月</option>
      <option value="9">9 月</option>
      <option value="10">10 月</option>
      <option value="11">11 月</option>
      <option value="12">12 月</option>
    </select>
  </form>
</body>
</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。
- プルダウンメニュー の要素名は<select>で、選択する内容は<option>要素である。
- <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。

```
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     var i, Option;
10     var Select = document.createElement("select");
11     document.getElementById("menu").appendChild(Select);
12     for(i=1;i<=12;i++) {
13       Option = document.createElement("option");
14       Option.setAttribute("value",i);
15       Select.appendChild(Option);
16       Option.appendChild(document.createTextNode(i+"月"));
17     }
18   }
19 //]]>
20 </script>
21 </head>
22 <body>
23   <form id="menu"></form>
24 </body>
25 </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 行目から始まる 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 7.3 実行例 7.2に対して次のことを行いなさい。

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。

表 7.3: DOM 要素に対するプロパティ

プロパティ名	説　　明
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 (DOM4 で定義)
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 (DOM4 で定義)
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 (DOM4 で定義)
nextElementSibling	指定された子要素の次の通常の要素 (DOM4 で定義)
previousElementSibling	現在の子要素の前にある通常の要素 (DOM4 で定義)

なお、DOM4[6] とは 2015 年 11 月現在 W3C が定める次の DOM の規格の Proposed Recommendation(勧告案) である。DOM の規格は今までに Level 1 から Level 3 までが Recommendation(勧告) となっている。

- 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 7.4 69ページから始まるリストと 70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 を使うとよい。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。

[6] http://www.w3.org/TR/domcore/

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   var Squares   = document.getElementById("Squares");
 3   var Select    = document.getElementById("select");
 4   var ColorName = document.getElementById("colorName");
 5   var Radio     = document.getElementById("radio");
 6   var Set       = document.getElementById("Set");
 7   var Inputs    = document.getElementsByClassName("click");
 8   var 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     var 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      var i, option;
11      var Select = makeElm("select", {"id":id}, parent);
12      for(i=from; i<=to; i++) {
13        option = makeElm("option",{ "value":i}, Select);
14        makeTextNode(prefix+i+suffix,option);
15      }
16      return Select;
17    };
```

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 17 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。
- 11 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。
- 12 行目から 15 行目で option 要素を順に作成している。
- 14 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() (28 行目から 30 行目) を呼び出している。

```
18    function makeElm(name, attribs, parent) {
19      var elm = document.createElement(name);
20      var attrib;
21      for(attrib in attribs) {
22        elm.setAttribute(attrib,attribs[attrib]);
23      };
24      if(parent) parent.appendChild(elm);
25      return elm;
26    }
```

18 行目から 26 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 (19 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 (21 行目から 23 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 (24 行目)。
- 作成した要素を戻り値にする (25 行目)。

```
27    function makeTextNode(text,parent) {
28        parent.appendChild(document.createTextNode(text));
29    };
```

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。

```
30    var Form  = document.getElementById("menu");
31    var Year  = makeSelectNumber(2000,2020,"","年","year", Form);
32    var Month = makeSelectNumber(1,12,"","月","month", Form);
33    var Days = [];
34    for(d=28; d<=31; d++) {
35      Days[d] = makeSelectNumber(1,d,"","日","day");
36    }
```

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。

- 30 行目では form 要素を得ている。
- 31 行目と 32 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 (34 行目から 36 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。

```
37    var d, today = new Date();
38    Year.value = today.getFullYear();
39    Month.value = today.getMonth()+1;
40    d = new Date(Year.value, Month.value,0).getDate();
41    Form.appendChild(Days[d]);
42    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 (37 行目)。[2]
- 38 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。
- 39 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。
- 41 行目と 42 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。

[2] この時間は 1970 年 1 月 1 日午前 0 時 (GMT) この時刻を UNIX エポックという。 からの経過時間であり、単位はミリ秒である。

- 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[3](40 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。

- 41 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、42 行目で、日の選択する値を設定している。

```
43     Form.addEventListener("change", function(){
44       var d2 = Form.children[2].value
45       d = new Date(Year.value, Month.value, 0).getDate();
46       if( d != Form.children[2].children.length) {//alert(d+" "+Form.children[2].children.leng
47         Form.replaceChild(Days[d],Form.children[2]);
48         Form.children[2].value = Math.min(Form.children[2].length, d2);
49       }
50     },false);
51   }
52 //]]>
53 </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。
- 44 行目で、現在表示されている日を保存している。
- 45 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 (46 行目) 子ノードを入れ替える (47 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 (48 行目)。

```
54 </head>
55 <body>
56   <form id="menu"></form>
57 </body>
58 </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。

[3] 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　PHPとは

日本 PHP ユーザー会のホームページ[1] 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。

ここにあるように、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。

なお、このテキストにおける PHP の使用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。

9.2　Web サーバーの基礎知識

9.2.1　サーバーの重要な設定項目

Web サーバーは　hypertext transfer protocol(HTTP) 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。このサービスを提供するプログラムとしては Apache が有名である。Apache によるサービスの設定ファイルは `httpd.conf` である。こ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 におけるサービスを識別するためのポート番号は大別して次の 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] http://www.php.gr.jp
[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。

HTTP の System ポート番号は 80 番となっている[3]。

- 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。

9.2.2　CGI

Common Gateway Interface(CGI) とは Web コンテンツをダイナミックに生成する標準の方法である。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。

9.2.3　Web サーバーのインストール

Web サーバーをインストールし、HTTP のサーバーを動かすためには Apache や PHP をインストールした後、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。なお、先頭の X はクロスプラットフォームを意味する。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 ISO-8859-1 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。

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。

[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`

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。
  - 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。
  - 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 では変数名は `$`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)。

次の例は配列に対して同様のことをしている。

```
 1  <?php
 2  $a = array(1,2);
 3  $b = $a;
 4
 5  print_r($a);
 6  print_r($b);
 7
 8  $a[0] = 20;
 9  print "1:\$b[0]=$b[0]\n";
10
11  $b = &$a;
12  $a[1] = 30;
13  print "2:\$b=";
14  print_r($b);
15  ?>
```

実行結果は次の通りである。

```
Array
(
    [0] => 1
    [1] => 2
)
Array
(
    [0] => 1
    [1] => 2
)
1:$b[0]=1
2:$b=Array
```

```
(
    [0] => 20
    [1] => 30
)
```

- 2 行目で配列の成分が 1 と 2 である配列を作成している。さらに、3 行目でこの配列を別の変数に代入している。
- 9 行目では配列の一部分の値を変えているが、3 行目で代入された変数の値には変化がないことがわかる (10 行目)。

  これから単純な配列の代入は配列全体がコピーされていることがわかる。
- 関数 print_r() は配列などの構造を持つ変数の内容をわかりやすく表示する関数である。より詳しい情報を知りたい場合には関数 var_dump() がある。

課題 9.1 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。

9.5.1 可変変数

ある変数の値が文字列のとき、その前に$をつけると、その文字列が変数名として使える。

実行例 9.2 次のリストは可変変数の使用例である。

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

- 変数$a と $b にそれぞれ 1 と 2 を代入している (1 行目と 2 行目)。
- 変数$c に文字列"a"を代入し、$$c を出力させると$a の値が表示される (5 行目と 6 行目)。
- 同様に、変数$c に文字列"b"を代入して、$$c を出力させると$b の値が表示される (8 行目と 9 行目)。

9.5.2　変数のスコープ

PHP では変数は特に宣言しなくても使用できる。また、指定しない限りローカルなスコープしか持たない。関数外で定義された変数も、関数内から参照できない。参照するためにはスーパーグローバル$GLOBAL を利用する。関数内での変数は外部からの参照できない。

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。

9.5.3　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 9.1)。

表 9.1: 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

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

$argv は PHP をコマンドプロンプトから実行したときの (空白で区切られた) パラメータを保持する文字列の配列である。$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

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 9.3 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位置を個々に指定しなくてはいけない。その指定にはべつ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。

9.7　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 9.4 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 9.2[4] 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 9.3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。

[4] この表と次の表は PHP ユーザー会の HP から転用した。

表 9.2: 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

課題 9.2 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。

9.8 配列

9.8.1 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  <?php
2  $a = array("red","yellow","blue");
3  foreach($a as $val) {
```

表 9.3: == 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    print "$val\n";
5  }
6  foreach($a as $key=>$val) {
7    print "$key:$val\n";
8   }
9   ?>
```

2 行目で単純な配列を定義して、3 行目から 5 行目と 6 行目から 8 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  <?php
2  $a = array("red"=>"赤","yellow"=>"黄","blue"=>"青");
3  foreach($a as $val) {
4    print "$val\n";
```

```
5  }
6  foreach($a as $key=>$val) {
7    print "$key:$val\n";
8    }
9  ?>
```

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  <?php
2  $b = array();
3  $b[3] = "3rd";
4  $b[0] = "0";
5  print count($b)."\n";
6  foreach($b as $key=>$val) {
7    print "$key:$val\n";
8  }
9  for($i=0;$i<count($b);$i++) {
10   if(isset($b[$i])) {
11     print "$i:$b[$i]\n";
12   }
13 }
14 ?>
```

この出力は次のようになる。

```
2
3:3rd
0:0
0:0
```

- 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。

9.8.2　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()　list() は配列の個々の要素をいくつかの変数にまとめて代入できる。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);
```

この結果は、変数$first、$second と$third にそれぞれ 1,2,3 が代入される。途中に変数を書かなければ、その分は飛ばされる。

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

この場合は変数$first と$third だけ代入が行われる。

array_pop() と array_push()　array_pop() はもとの配列から最後の要素を取り除いて配列の大きさを一つ少なくし、取り除いた要素を戻り値とする。array_push() は与えられた引数を順に配列の最後に付け加えて配列の大きさを増加させる。

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

配列の先頭に対して要素を取り出したり (array_shift())、追加する (array_unshift()) することもできる。

配列のキーと値に関する操作　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。

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。

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()* 関数を実現しなさい。

9.9 関数

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。

- 関数の外で定義された変数は参照できない。また、関数内の変数はすべてローカルである。
- 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。

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 行目での関数では 3 番目の引数が参照渡し、4 番目の引数がデフォルトの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セットされているかどうかを確かめるものである。
- 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。
- 11 行目から 13 行目までは変数に値を代入して、呼び出し元の変数が変わるかどうかのチェックをする。
- 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
```

9.9.1　可変関数

文字列が代入された変数を用いて、その文字列を変数名にできる可変変数と同様に、文字列が代入された変数の後に () をつけると、その文字列の関数を呼び出すことができる。

実行例 9.8 次のリストは可変変数の利用例である。

```
 1  <?php
 2  function add($a, $b) { return $a+$b;}
 3  function sub($a, $b) { return $a-$b;}
 4
 5  $a = 5;
 6  $b = 2;
 7  $f = "add";
 8  print "$f($a,$b) = " . $f($a,$b) ."\n";// add(5,2) = 7
 9
10  $f = "sub";
11  print "$f($a,$b) = " . $f($a,$b) ."\n";// sub(5,2) = 3
12  ?>
```

- 2 行目と 3 行目で 2 つの関数 add() と sub() を定義している。
- 7 行目で変数$f に文字列"add"を代入して、8 行目で可変関数として呼び出すと、関数 add が呼び出されていることがわかる。
- 同様に、変数$f に文字列"sub"を代入して、可変関数として呼び出すと、関数 sub が呼び出されていることがわかる。

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。

```
var 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 行目の間はヒアドキュメント形式で HTML 文書の初めの部分を出力させている。
- 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スーパーグローバル$_POST 内の連想配列としてアクセスができる。
- 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。

出力された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、ボタンが押されたときに直ちに、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。

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こなかった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、どのようなものが送られている確認にせよ。

`$_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 Web Storage は localStoarage と sessionStorage の 2 種類がある。localStorage 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おなじ localStorage を共有する。こ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。

```
 1 <!DOCTYPE html>
 2 <html>
 3 <head>
 4 <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
 5 <title>WebStorage --- localStorage</title>
 6 <script type="text/javascript">
 7 //<![CDATA[
 8 var Storage = window.localStorage;
 9 //var Storage = window.sessionStorage;
10 window.onload = function() {
11   var AccessList, Message = document.getElementById("message");
12   var D = new Date();
13   if(Storage["access"]) {
14     AccessList = JSON.parse(Storage["access"]);
15   } else {
16     AccessList = [];
17     appendMessage(Message, "初めてのアクセスです");
18   }
19   AccessList.unshift(D.getTime());
20   Storage["access"] = JSON.stringify(AccessList);
21   appendMessage(Message, "今までのアクセス時間です");
22   AccessList.forEach(function(D, i, A) {
23     appendMessage(Message, new Date(D));
24   });
25 }
26 function appendMessage(P, Mess) {
27   var div = document.createElement("div");
28   P.appendChild(div);
29   div.appendChild(document.createTextNode(Mess));
30 }
31 //]]
32 </script>
33 </head>
34 <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`<div>`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。
  - 引数で渡される関数の引数は 3 つあり、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`) 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。
  - `<div>`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 <script type="text/javascript">
2 //<![CDATA[
3 //var Storage = window.localStorage;
4 var Storage = window.sessionStorage;
5 window.onload = function() {
6   var AccessList, Message = document.getElementById("message");
7   appendMessage(Message, "新しいページです");
```

```
 8   if(Storage["access"]) {
 9     AccessList = JSON.parse(Storage["access"]);
10   } else {
11     AccessList = [];
12     appendMessage(Message, "初めてのアクセスです");
13   }
14   appendMessage(Message, "今までのアクセス時間です");
15   AccessList.forEach(function(D, index, A) {
16     if(index < 3) appendMessage(Message, new Date(D-0));
17   });
18 }
19 function appendMessage(P, Mess) {
20   var div = document.createElement("div");
21   P.appendChild(div);
22   div.appendChild(document.createTextNode(Mess));
23 }
24 //]]
25 </script>
26 </head>
27 <body>
28   <div  id="message"/>
29 </body>
30 </html>
```

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 10.3 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[1]。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` という機能を用いて実現されている。

[1] Ajax の名称を提案したブログ `http://www.adaptivepath.com/publications/essays/archives/000385.php` は 2014 年 12 月 8 日現在アクセスできないようである。

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>
 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       var i, option;
11       var Select = makeElm("select", {"id":id}, parent);
12       for(i=from; i<=to; i++) {
13         option = makeElm("option",{ "value":i}, Select);
14         makeTextNode(prefix+i+suffix,option);
15       }
16       return Select;
17     };
18     function makeElm(name, attribs, parent) {
19       var elm = document.createElement(name);
20       var attrib;
21       for(attrib in attribs) {
22         elm.setAttribute(attrib,attribs[attrib]);
23       };
24       if(parent) parent.appendChild(elm);
25       return elm;
26     }
27     function makeTextNode(text,parent) {
28         parent.appendChild(document.createTextNode(text));
29     };
30     var i, today = new Date();
31     var y = today.getFullYear();
32     var m = today.getMonth();
33     var d;
34     var Form  = document.getElementById("menu");
35     var Year  = makeSelectNumber(2000,2020,"","年","year", Form);
36     var Month = makeSelectNumber(1,12,"","月","month", Form);
37     var Days = [];
```

```
38    for(i=28; i<=31; i++) {
39      Days[i] = makeSelectNumber(1,i,"","日","day");
40    }
41    Year.value = y;
42    Month.value = m+1;
43    d = new Date(y, m+1,0).getDate();
44    Form.appendChild(Days[d]);
45    Form.children[2].value = today.getDate();
46    var changePulldown = function(){
47      var d2 = Form.children[2].value
48      d = new Date(Year.value, Month.value, 0).getDate();
49      if( d != Form.children[2].children.length) {
50        Form.replaceChild(Days[d],Form.children[2]);
51        d2 = Math.min(Form.children[2].length, d2);
52        Form.children[2].value = d2;
53      }
54      var xmlHttpObj = null;
55      if(window.XMLHttpRequest) {
56        xmlHttpObj = new XMLHttpRequest();
57      } else if(window.ActiveXObject ) {
58        xmlHttpObj = new ActiveXObject("Msxml2.XMLHTTP");//IE6
59      } else {
60        try {
61          xmlHttpObj = new ActiveXObject("Microsoft.XMLHTTP");//IE5
62        } catch(e) {
63        }
64      }
65      if(xmlHttpObj) {
66        xmlHttpObj.onreadystatechange = function(){
67          if(xmlHttpObj.readyState == 4 && xmlHttpObj.status == 200) {
68            document.getElementById("details").firstChild.nodeValue =
69                xmlHttpObj.responseText;
70          }
71        }
72        xmlHttpObj.open("GET",
73          "./aniversary.php?month=" + Month.value+ "&day="+d2,true);
74        xmlHttpObj.send(null);
75      }
76    }
77    Form.addEventListener("change", changePulldown,false);
```

```
78     changePulldown();
79   }
80 //]]>
81 </script>
82 </head>
83 <body>
84   <form id="menu"></form>
85   <p id="details"> </p>
86 </body>
87 </html>
```

- 以前のものとは 46 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。
- 47 行目から 52 行目は以前と同じプルダウンメニューの処理である。
- 53 行目から 63 行目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。
  - 古いバージョンの IE は別のオブジェクトで通信をするので、ブラウザが XMLHttpRequest メソッドを持つか確認し (54 行目)、持っている場合はそのオブジェクトを新規作成する (55 行目)。
  - 56 行目から 62 行目は古い IE のためのコードである。
  - このようにブラウザの機能の違いで処理を変えることをクロスブラウザ対策という。通常はブラウザがその機能を持つかどうかで判断する。
- XMLHttpRequest が生成できたら (65 行目)、このオブジェクトが生成する onreadystatechange イベントのイベントハンドラーを登録する (66 行目から 71 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[2]。
  - 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。
- 72 行目から 73 行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、通信の終了をのため 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 (77 行目)、最後に現在の日付データをサーバーに要求する。、

[2]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

表 10.1: XMLHttpRequest の通信の状態[3]

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。

- 得られたデータは 85 行目の p 要素の中に入れる (68 行目から 69 行目)。この要素の firstChild を指定しているので 85 行目には と の間に空白を設けて、テキストノードが存在するようにしている。

課題 10.4 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。残りのものについてもこの方法で利用できることを確認しなさい。

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($mm) >1) {
11     if(mb_convert_encoding($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$_GET 内に値があれば (isset()) が true になれば、その値を、そうでなければコマンドからの引数を設定している。
  - スーパーグローバル$argv はの先頭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[4]。
- 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]
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。

[4] 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。

  - 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[5]。
- 20 行目で得られた情報をストリームに出力している。

課題 10.5 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。

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。

[5] 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。現在のところjQueryにはver. 1のものとver. 2の系列が提供されている。大きな違いについては次のように解説されている。

> jQuery 2.x
> jQuery 2.x has the same API as jQuery 1.x, but does not support Internet Explorer 6, 7, or 8. All the notes in the jQuery 1.9 Upgrade Guide apply here as well. Since IE 8 is still relatively common, we recommend using the 1.x version unless you are certain no IE 6/7/8 users are visiting the site. Please read the 2.0 release notes carefully.

一番大きな違いはver. 2では古いInternet Explorerの対応を打ち切ったところである。

また、配布されているライブラリーはコメントなどがそのまま入っているものと、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の2種類がある。

11.2　jQueryの基本

11.2.1　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の要素群を返す。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。

[1] `jQuery.com`

- 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。

11.2.2　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。

11.3 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。

11.4 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。

11.5 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。"POST"のときは null
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

11.6 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-1.11.2.min.js"></script>
 8 <script type="text/ecmascript">
 9 //<![CDATA[
10 $(window).ready(function(){
11     function makeSelectNumber(from, to, prefix, suffix, id, parent){
12       var i, option;
13       var Select = $("<select/>", {"id":id});
14       if(parent) parent.append(Select);//$(parent).append(Select);
15       for(i=from; i<=to; i++) {
16         option = $("<option/>",{"value":i,"text":prefix+i+suffix});
17         Select.append(option);
18 //          $("<option/>",{"value":i,"text":prefix+i+suffix}).appendTo(Select);
19       }
20       return Select;
21     };
22     var i, today = new Date();
23     var y = today.getFullYear();
24     var m = today.getMonth();
25     var Form  = $("#menu");
26     var Year  = makeSelectNumber(2000,2020,"","年","year", Form);
27     var Month = makeSelectNumber(1,12,"","月","month", Form);
28     var Days = [];
29     for(i=28; i<=31; i++) {
30       Days[i] = makeSelectNumber(1,i,"","日","day");
31     }
32     Year.val(y);     //$("#year").val(y);
33     Month.val(m+1);  //$("#month").val(m+1);
34     var d = new Date(y, m+1,0).getDate();
35     Form.append(Days[d]);
36     $("#day").val(today.getDate());
37     var changePulldown = function(){
38       var d2 = $("#day").val();
39       d = new Date(Year.val(), Month.val(), 0).getDate();
40       if( d != $("#day option").length) {
41         $("#day").replaceWith(Days[d]);
42         $("#day").val(Math.min($("#day option").length, d2));
43       }
44       jQuery.ajax({
45         type:"GET",
46         url:      "./aniversary.php",
```

```
47         data:     "month=" + Month.val()+ "&day="+d2,
48         dataType: "text",
49         success : function(Data){
50           $("#details").text(Data);
51         },
52         error:function(){alert("error");}
53       });
54     };
55     Form.change(changePulldown);
56     changePulldown();
57 });
58 //]]>
59 </script>
60 </head>
61 <body>
62   <form id="menu"></form>
63   <p id="details"> </p>
64 </body>
65 </html>
```

- 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。
  - 13 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)。
  - 15 行目から 19 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 18 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。
- 22 行目から 31 行目も依然とほとんど同じである。25 行目では機能 3を用いて要素を参照している。
- 32 行目、33 行目と 36 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。
- 37 行目から 54 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のぞけは 38 行目から 43 行目までは以前と同じである。40 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。
  - 44 行目から 53 行目は Ajax の処理を定義している。ブラウザによる違いに対する対処や、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。

11.7 jQuery のコードを読む

次のリストは jQuery-1.11.3.js の冒頭の部分である[2]。

```
1  /*!
2   * jQuery JavaScript Library v1.11.3
3   * http://jquery.com/
4   *
5   * Includes Sizzle.js
6   * http://sizzlejs.com/
7   *
8   * Copyright 2005, 2014 jQuery Foundation, Inc. and other contributors
9   * Released under the MIT license
10  * http://jquery.org/license
11  *
12  * Date: 2015-04-28T16:19Z
13  */
14 (function( global, factory ) {
15   if ( typeof module === "object" && typeof module.exports === "object" ) {
16     // For CommonJS and CommonJS-like environments where a proper window is present,
17     // execute the factory and get jQuery
18     // For environments that do not inherently posses a window with a document
19     // (such as Node.js), expose a jQuery-making factory as module.exports
20     // This accentuates the need for the creation of a real window
21     // e.g. var jQuery = require("jquery")(window);
22     // See ticket #14549 for more info
23     module.exports = global.document ?
24       factory( global, true ) :
25       function( w ) {
26         if ( !w.document ) {
27           throw new Error( "jQuery requires a window with a document" );
28         }
29         return factory( w );
30       };
31   } else {
32     factory( global );
33   }
34 // Pass this if window is not defined yet
35 }(typeof window !== "undefined" ? window : this, function( window, noGlobal ) {
36 // Can't do this because several apps including ASP.NET trace
37 // the stack via arguments.caller.callee and Firefox dies if
38 // you try to trace through "use strict" call chains. (#13335)
39 // Support: Firefox 18+
40 //
41 var deletedIds = [];
42 var slice = deletedIds.slice;
43 var concat = deletedIds.concat;
44 var push = deletedIds.push;
45 var indexOf = deletedIds.indexOf;
46 var class2type = {};
47 var toString = class2type.toString;
48 var hasOwn = class2type.hasOwnProperty;
49 var support = {};
```

[2]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。

```
50  var
51    version = "1.11.3",
52    // Define a local copy of jQuery
53    jQuery = function( selector, context ) {
54      // The jQuery object is actually just the init constructor 'enhanced'
55      // Need init if jQuery is called (just allow error to be thrown if not included)
56      return new jQuery.fn.init( selector, context );
57    },
58    // Support: Android<4.1, IE<9
59    // Make sure we trim BOM and NBSP
60    rtrim = /^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,
61    // Matches dashed string for camelizing
62    rmsPrefix = /^-ms-/,
63    rdashAlpha = /-([\da-z])/gi,
64    // Used by jQuery.camelCase as callback to replace()
65    fcamelCase = function( all, letter ) {
66      return letter.toUpperCase();
67    };
```

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。またコメントを見るとクロスブラウザ対策が行われていることも見て取れる。

ソースコードの最小化のテクニックとしてグローバルなオブジェクトを短い変数名で置き換えることを行っている。たとえば、35 行目では冒頭の定義された関数の引数に、関数を渡している。その関数の仮引数は window となっている。これにより、開発時はわかりやすい変数名が使えるメリットがある。

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 v1.11.3 | (c) 2005, 2015 jQuery Foundation, Inc. | jquery.org/license */
 2  !function(a,b){"object"==typeof module&&"object"==typeof
 3      module.exports?module.exports=a.document?b(a,!0):function(a){if(!a.document)throw
 4      new Error("jQuery requires a window with a document");
 5    return b(a)}:b(a)}
 6      ("undefined"!=typeof window?window:this,function(a,b){
 7        var c=[],d=c.slice,e=c.concat,f=c.push,g=c.indexOf,h={},i=h.toString,
 8    j=h.hasOwnProperty,k={},l="1.11.3",
 9    m=function(a,b){return new  m.fn.init(a,b)},
10    n=/^[\s\uFEFF\xA0]+|[\s\uFEFF\xA0]+$/g,o=/^-ms-/,
11    p=/-([\da-z])/gi,q=function(a,b){return b.toUpperCase()};
```

短縮化のために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。

11.8 JavaScript ファイルの短縮化

11.8.1 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。[3]。

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. It is a true compiler for JavaScript. 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 11.1)。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。

[3] `https://developers.google.com/closure/compiler/`

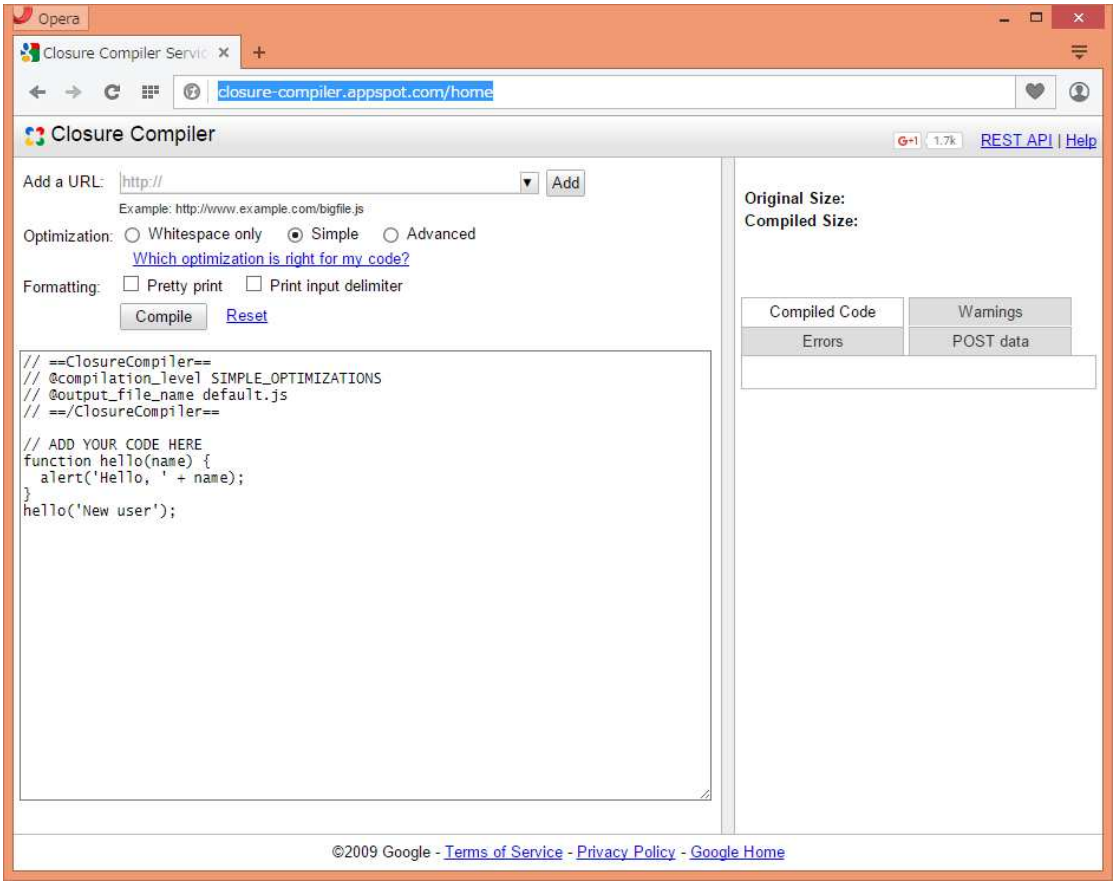

図 11.1: Closure Compiler のホームページ

11.8.2 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 11.2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"),
6   c=document.getElementsByClassName("click"),
7   d=b.children[1];
8   b.children[0].style.background="red";
```

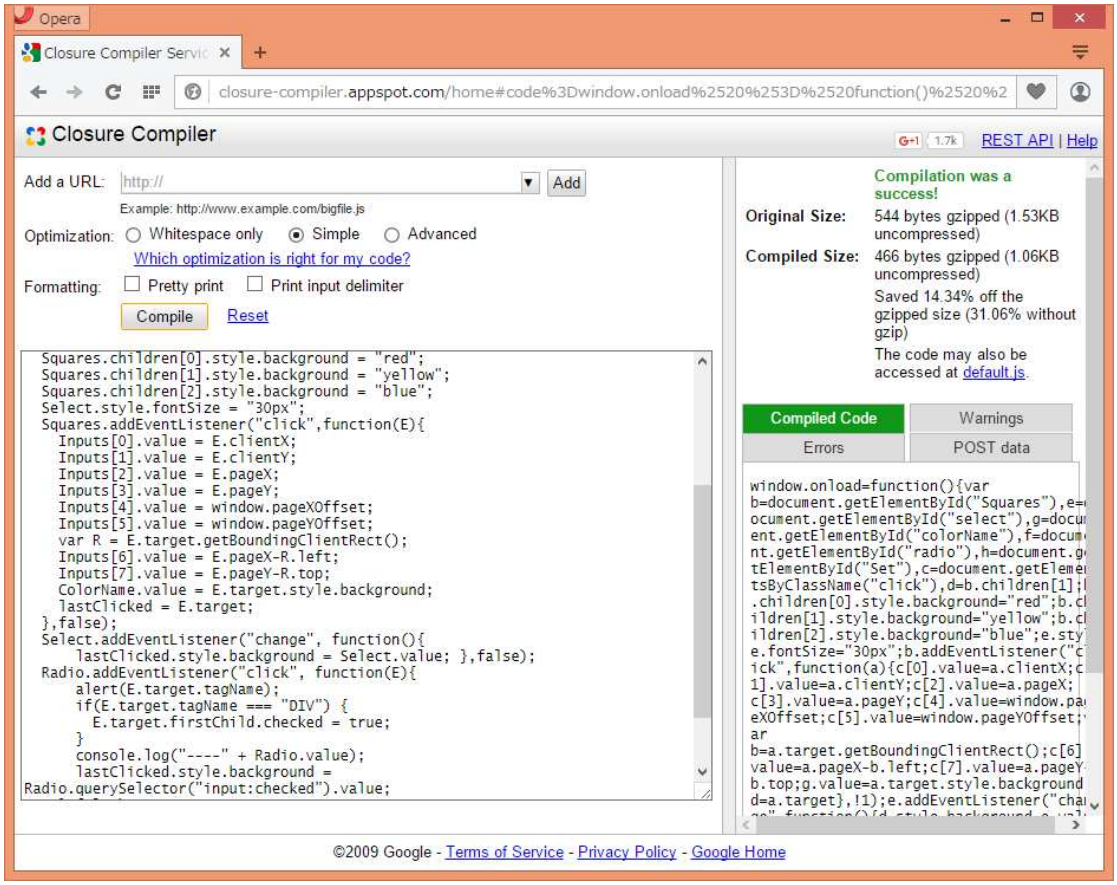

図 11.2: Closure Compiler の結果

```
 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;
```

```
23      d=a.target
24    },!1);
25    e.addEventListener("change",function(){
26      d.style.background=e.value},!1);
27    f.addEventListener("click",function(a){
28      alert(a.target.tagName);
29    "DIV"===a.target.tagName&&(a.target.firstChild.checked=!0);
30    console.log("----"+f.value);
31    d.style.background=f.querySelector("input:checked").value
32    },!1);
33    h.addEventListener("click",function(){
34      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 などはそのまま短縮化、共通化されていない。この部分も短くするためにはソースコードを変える必要がある。

ここでの例では単純に関数を置き換えただけではうまくいかないことがわかる。

ここでは document.getElementById をラップする関数を定義している。

```
function getElm(N){
  return document.getElementById(N);
}
var Squares   = getElm("Squares");
var Select    = getElm("select");
...
```

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を定義してその場で実行する関数の引数に document と"getElementById"を渡すことで解決できる[4]。

[4] document.getElementById を渡せば十分と思われるかもしれないが実行時にエラーが出てうまくいかなかった。

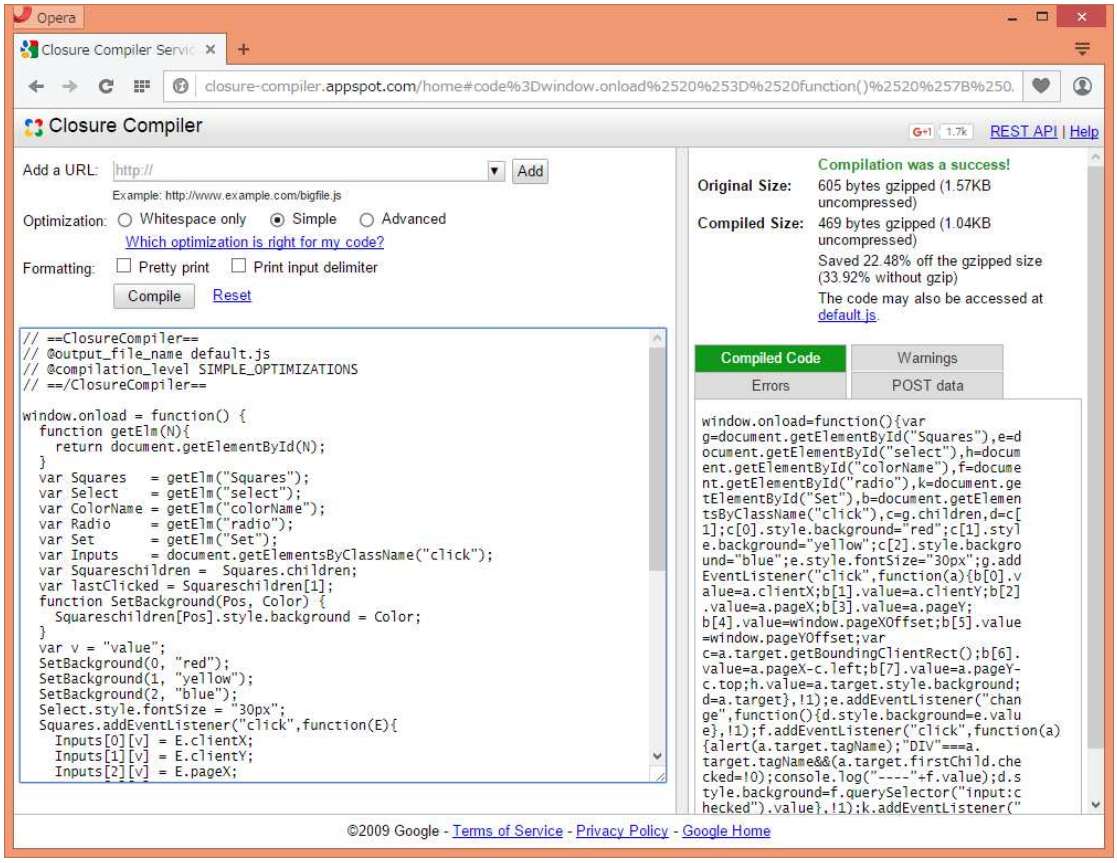

図 11.3: Closure Compiler の結果 (2)

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   = document[getElementById]("Squares");
    Select    = document[getElementById]("select");
    ColorName = document[getElementById]("colorName");
    Radio     = document[getElementById]("radio");
    Set       = document[getElementById]("Set");
      })(document,"getElementById");
```

図 11.4がその結果である。コンパイル後の結果が 1.02KB とわずかに小さくなっていることがわかる。

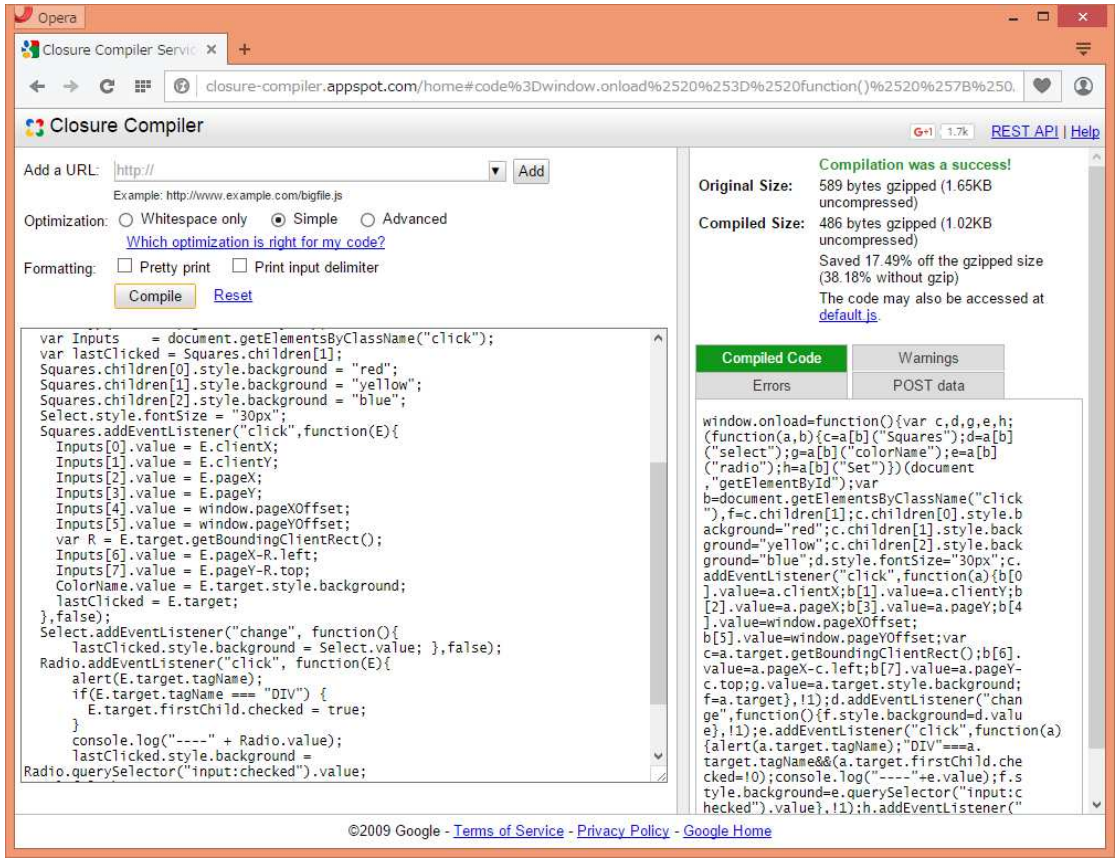

図 11.4: Closure Compiler の結果 (3)

第12回　バージョン管理

12.1　バージョン管理とは

12.1.1　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。

12.1.2　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。

12.1.3　バージョン管理ソフトの概要

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。

12.2　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。

12.2.1　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かった。

- GitHub 上ではファイルの変更履歴などが見える。

- 開発者に対して変更などの要求が可能 (pull request)

12.2.2　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。

12.2.3　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

```
git --config --global user.name "Foo "
```

メールアドレスの登録

```
git --config --global user.email "Foo@example.com"
```

SSH Key の登録

GitHub との接続には SSH による通信で行います。このは公開鍵暗号方式で行いう。`ssh-keygen -t rsa -C "Foo@exampl`
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。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カギが生成される。

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。

12.2.4 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。

12.2.5 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。

`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`	`-u` 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

`git add`