# ネットワークアプリケーション

## 第11回 クライアントサイドウェブプログラミング (3)

石井 健太郎

(423研究室・オフィスアワー水3限)

# スケジュール

9月15日　第1回「TCP/IPプロトコルスイート」
9月29日　第2回「ネットワークアプリケーションのプログラミングモデル」
10月6日　第3回「アプリケーションプロトコルの設計（1）」
10月13日　第4回「アプリケーションプロトコルの設計（2）」
10月20日　第5回「アプリケーションプロトコルの設計（3）」　演習（第3演習室）
10月27日　第6回「アプリケーションプロトコルの設計（4）」　演習（第3演習室）
11月10日　第7回「ウェブプログラミングについて」
11月17日　第8回「クライアントサイドウェブプログラミング（1）」

# スケジュール

| | | |
|---|---|---|
| 11月24日 | 第9回「アプリケーションプロトコルの設計（5）」 | 演習（第3演習室） |
| 12月1日 | 第10回「クライアントサイドウェブプログラミング（2）」 | 演習（第3演習室） |
| 12月8日 | 第11回「クライアントサイドウェブプログラミング（3）」 | |
| 12月15日 | 第12回「クライアントサイドウェブプログラミング（4）」 | |
| 12月22日 | 第13回「クライアントサイドウェブプログラミング（5）」 | 演習（第3演習室） |
| 1月12日 | 第14回「クライアントサイドウェブプログラミング（6）」 | 演習（第3演習室） |
| 1月19日 | 第15回「まとめと演習」 | 演習（第3演習室） |

# 提出課題(再掲)

- SimpleBinaryServer/SimpleBinaryClient は
  バイナリデータを送受信することで,
  マウスの座標・メッセージを交換するプログラム(未完)である
  - サーバ側・クライアント側ともに,
    送信部分は実装されているが, 受信部分は実装されていない
- 受信部分を実装し, SimpleTextServer/SimpleTextClient と
  同様に動作するように完成させる

- 提出期限: 12月22日(火)深夜まで
- 提出方法
  - サーバ・クライアントともに, 1つのフォルダにいれて,
    <学籍番号>.zip というファイルに圧縮して提出
  - 課題提出用ドライブ(Xドライブ) SimpleTCPBinaryApplication フォルダに置く

# 提出課題(再掲)

```
//- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
@Override
public void run() {
    if (this.socket == null || this.input == null || this.output == null)
        return;

    this.enabled = true;

    try {
        while (this.enabled) {
            char command = this.input.readChar();
            if (command == 'N') {
                //implement here

            } else if (command == 'S') {
                //implement here

            } else if (command == 'H') {
                // implement here

                this.application.setHoge();
            } else {
                System.err.println("Invalid command! : " + command);
            }
        }
    } catch (IOException ioe) {
        ioe.printStackTrace();
    }
}
```

- ヒント
  - SimpleBinarySocket.java の run () で受信部分を記述している
  - this.input.readChar () で1文字受信できるが，int型のデータや文字データを受信するにはどうすればよいかをクラスライブラリのドキュメントを読んで考える

- http://docs.oracle.com/javase/jp/8/api/java/io/DataInputStream.html

- 今日は以下について学びます
  - JavaScriptの基本構文
  - Document Object Model（DOM）

- （参考）ドットインストール JavaScript入門
  - http://dotinstall.com/lessons/basic_javascript_v2

# データ型(再掲)

- 文字列
- 数値
- 真偽値(true / false)
- オブジェクト
  - 配列
  - 関数
  - 組み込みオブジェクト
- undefined 定義されていない
- null なにもない

# 数値と演算子(再掲)

- **数値**
  10
  2.5
  −2.5
- **演算子**
  ＋　－　＊　／　％
  ＋＝　－＝　＊＝　／＝
  ＋＋　－－

【注意】+ は「文字列の結合の演算子」でもある。左右のどちらかが文字のとき、文字に変換される

```
var x;
x = "1" + 2;
alert(x);
x = +"1" + 2;
alert(x);
```

のように単項演算子として使うと文字列を数値に変換する関数として使うこともできる

# 文字列(再掲)

- 表現方法と特殊文字
- 「"」か「'」で囲む

```
x = "hello world";
x = 'hello world';
x = "it's a pen";
x = 'it¥'s a pen';
```

- エスケープ文字は「¥」(円マーク)
- 特殊文字の改行やタブは「¥n」「¥t」

# 制御構造(再掲)

- if文 C言語と同じ
- for文 C言語と同じ(変数の宣言はvarに変わる)

```
for (var i=0; i<20; i++) {
  console.log(i);
}
```

- while文、do while文　　C言語と同じ
- break文、continue文　　C言語と同じ
- switch文
- 条件分岐の中にある比較演算子は二種類ある

=== !==　型も含めて一致するか比較

== !=　異なる型の場合には変換してから比較

# 関数(再掲)

- C言語と異なり、返り値の型の宣言は不要
- その代わり、functionというキーワードを書く
- 引数の型宣言も不要

```
var x = 100;
var y = 200;

function max(x, y) {
  if (x>=y) return x; else return y;
}

alert(max(x,y));
```

標準的な書き方

```
var x = 100;
var y = 200;

var max = function (x, y) {
  if (x>=y) return x; else return y;
}

alert(max(x,y));
```

このような書き方もできる

# 配列

- 二つの表現方法

```
var student = new Array(3);
student[0] = "櫻井";
student[1] = "松本";
student[2] = "大野";
```

```
var student = ["櫻井","松本","大野"];
```

- 使い方

```
for (var i=0; i<3;  i++) {
  alert(student[i]);
}
```

```
for (var i in student) {
  alert(student[i]);
}
```

配列の各要素を処理するには"in"が便利

# オブジェクト（連想配列）

- キーと値の組

```
var hobby = new Object();
hobby['櫻井'] = "ありません";
hobby['松本'] = "買い物";
hobby['大野'] = "釣り";
//hobby.櫻井 = “ありません”; //でもOK
```

```
var hobby = {
  櫻井:"ありません",
  松本:"買い物",
  大野:"釣り"
}
```

- 使い方

```
for (key in hobby) {
  alert(key + "の趣味は“ + hobby[key]);
  //alert(key + “の趣味は“ + hobby.key); //ではNG! keyというメンバー変数へのアクセス
}
```

# オブジェクト（連想配列）

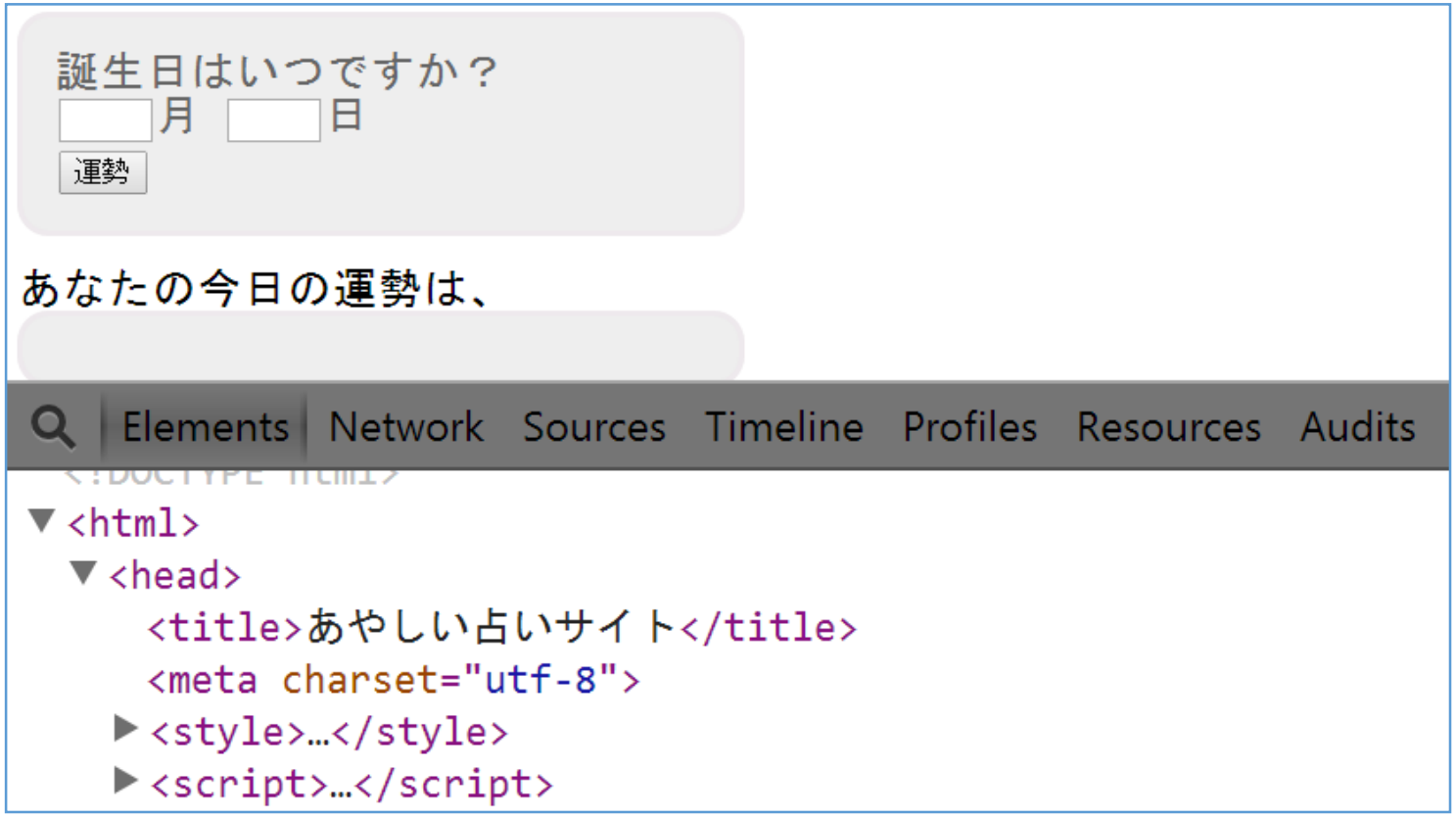
- 「連想配列」の「配列」の例

```
var student = new Array(3);
student[0] = {
  名前:"櫻井",
  趣味:"なし",
  特技:"ラップ"
}
student[1] = {
  名前:"松本",
  趣味:"買い物",
  特技:"演技"
}
student[2] = {
  名前:"大野",
  趣味:"釣り",
  特技:"ダンス"
}
```

```
for (i in student) {
  alert(student[i]["名前"]
  +"の趣味は"
  + student[i]["趣味"]);
}
```

# Document Object Model（DOM）

- HTMLドキュメントの構造
- デベロッパーツールでElementsタブで構造が分かる



# Document Object Model（DOM）

```
<!DOCTYPE html>
<html>
<head>
<title>私の占いサイト</title>
<meta charset="utf-8" />
<script>
（次のページ）
</script>
</head>
<body>
 <form name="form1">誕生日はいつですか？<br>
   <input type="text" name="month" size=2>月<br>
   <input type="text" name="date" size=2>日<br>
   <input type="button" name="unsei" value="運勢">
 </form>
 <div>
  あなたの今日の運勢は、<br><br>
  <div id="target"></div>
 </div>
</body>
</html>
```

# Document Object Model（DOM）

```
window.onload = function () {  // ロードしたときに動かす処理を定義
  document.form1.unsei.onclick = unsei;  // クリックしたらunseiを呼ぶ
}

var u = ["<p>大吉</p>", "<p>中吉</p>", "<p>小吉</p>"]; // 配列の要素を定義

function getRandom(max) {      // 0からmax-1までの乱数を発生させる関数
        return Math.floor(Math.random()*max);
}

function unsei() {
        var m1 = document.form1.month.value;
        var d1 = document.form1.date.value;
        alert(m1+"月"+d1+"日ですね"); // 誕生日を確認
        var r = getRandom(u.length);
        var target = document.getElementById("target");

        target.innerHTML = u[r];  // targetに文字列u[r]を埋め込む
}
```

# 変数のスコープについて

- **関数内で宣言した変数は、関数の外からは見えない**

```
function main() {
  var x = "関数の中";
}
alert(x);
```

ローカル変数

- **関数の外で宣言した変数は、関数の中からは見える**

```
var x = "関数の外";
function main() {
  alert(x);
}
```

グローバル変数

# 変数のスコープについて

- C言語と異なり、関数の中で宣言があると
  <u>宣言の前</u>で利用してもローカル変数となることに注意

```
window.onload = function () {
  main();
}

var x="関数の外";
function main() {
  alert(x);
  var x = "関数の中";
  alert(x);
}
```

ローカル変数
（代入前なのでundefined）

ローカル変数
（代入後なので"関数の中"）

- 来週12月15日（火）も1501教室に集まってください