# JavaScript によるゲームプログラミング

いで@いで庵

## 1 動機

プログラミングを始めた頃は C 言語をやっていたのに (その前は HTML と CSS) いつの間にかたどり着いてしまったのが JavaScript。いろいろと異端な言語である故、敢えて無茶をさせたくなるもの。

その無茶のうち JS 云々関係なしにやりたかったのがゲームプログラミング。そんなわけで JavaScript でゲームプログラミングなんて道へと踏み出してしまった。

## 2 HTML,CSS, そして JavaScript

JavaScript によるゲームプログラミングについて述べていく前に今回の話の土台となる技術である HTML,CSS,JavaScript について個々の技術の紹介並びにこれらの関係性について述べていく。

### 2.1 HTML(Hyper Text Markup Language)

HTML はデータ記述のための言語で、文章にタグ付けを行うことで文章の構造を明確化する。HTML 文書は html 要素をルートとする木構造を形成しており、HTML 文書中のタグで囲われた要素は木構造のノードに対応する。

### 2.2 CSS (Cascading Style Sheets)

HTML(に限らず XML にも用いられるが) を Web ブラウザに表示する際、素の HTML に対して装飾を施すための言語。かつての HTML は論理的構造の記述と装飾の記述が混在していたが、CSS の登場によって HTML による論理的構造記述と CSS による装飾記述とに分割されるようになった。

ただし今現在もなお各種ブラウザ間での CSS 解釈に相違が見られる。

### 2.3 JavaScript

JavaScript は主に Web ブラウザ上で動くスクリプト言語である。Java とは関係ない。オブジェクト指向言語であるが C++ や Java とは違ってクラスが存在せずインスタンス単位で継承が行われる (プロトタイプ指向)。関数がファーストクラスなオブジェクトであり、クロージャや無名関数があるなど関数型言語の影響が強い。

Web ブラウザ上での動作の場合、DOM(Document Object Model) を用いることで動的に HTML の木構造や個々のノードに対する操作が出来る。

### 2.4 HTML,CSS,JavaScript の関係

HTML　骨組み

CSS　基本デザイン

JavaScript　機能実装。DOM を経由して HTML の木構造を操作したりノードのプロパティを操作することでスタイルを変更したりする。

## 3 ゲームプログラムの基本構造

一般にゲームプログラムの構造は下記のようになる。

```
while(/*ゲームを継続する条件*/){
  //プレイヤーから入力を取得
  //ゲームの状態を更新。
  //描画
  //音声再生 ex.SE、BGM...etc
}
```

ゲーム状態の更新がゲームプログラムの本質的な部分。この辺は (おおよそ) 言語によらない。ゲームループ、入力の取得、描画、音声再生が言語によって実装が変わってくる。

これらの要素について JavaScript による実装方法をこれから述べていく。

## 4 ゲームループ

C とかでよくあるのはこういう書き方。

```
while(/*ゲーム継続条件*/){
  //ゲーム継続条件を満たしていたらループする。
  //入力受け取ったりゲーム状態更新したり描画したり
  sleep( waitTime );//waitTime ミリ秒待機する。
}
```

しかし JavaScript には sleep() のような関数は存在しない。

またループ構文の実行中は DOM 操作の結果が描画に反映されず、長時間 (数十秒ないしそれ以上) 1 つの JavaScript プロセスの実行が続いている場合、Web ブラウザがスクリプトの強制終了を促すダイアログを表示する (あるいは最悪の場合ブラウザが「応答なし」になる) ため、この方法ではゲームは正常に動作しない。

JavaScript でゲームループを構築する場合は以下のような記述になる。

```
setTimeout( function(){
  if(/*ゲーム終了条件*/){
    //ゲーム終了条件を満たしたら
    //ゲームの更新とかせずに即終了。
    return;
  }
  //入力受け取ったりゲーム状態更新したり描画したり
  setTimeout(arguments.callee,waitTime);
  //次のループステップを設定。
},waitTime );
```

setTimeout は第二引数 waitTime ミリ秒後に第一引数に指定した関数を呼び出す関数である。

外側の setTimeout の第一引数にループの 1 ステップ分の処理を記述した関数を渡す。

その関数の最後の setTimeout() 呼び出しでは arguments.callee(外側の setTimeout に渡している関数自身) を渡すことで次のループステップを設定している。

このように記述することで waitTime 秒待機して実行というループが構築される。このようにして構成されたループは各回が別のプロセスとなっており、ブラウザによる強制終了ダイアログ表示を回避できる。

また、DOM 操作による描画はループの各回の処理の終了後に反映される。

## 5 入力取得

ユーザーが何かアクションをするとイベントというものが発生する。イベントの発生ごとに呼び出す関数 (コールバック関数) を設定することができる。コールバック関数からはイベントの詳細な情報へのアクセスが可能である。

よって入力を取得する際は取得したい入力に関するイベントに入力を監視する関数を設定するのが基本方針となる。ただし、イベント処理に関してはブラウザ間で相違が見られるため、それらの相違を踏まえつつ各種入力取得について述べていく。

### 5.1 キー入力の取得

キーボード入力に関するイベントは以下の 3 種類

- keydown
- keypress
- keyup

それぞれキーを押した瞬間、キーを押し続けている間、キーを離した瞬間に発生するイベントである

が、しかしブラウザや OS によってイベントの発生タイミングや、発生するイベントの種類が異なる場合がある。

例を挙げていくと、

- Opera ではキーを押し続けている間は keydown イベントは発生しないが、Windows 上の他のブラウザ (IE,Firefox,Safari,GoogleChrome) では keydown イベントが発生し続ける。(他の OS 上では発生しない)
- 文字でないキー (Shift キー,Ctrl キーなど) を押し続けたとき Oepra では keypress が発生し続けるが、他のブラウザでは keydown が発生し続ける、keypress は発生しない。
- Firefox では方向キーを押し続けている間 keydown と keypress が発生し続けるが、IE,Safari,GoogleChrome(いずれも Windows 上) では keydown が発生し続け keypress は発生しない、Opera では keypress が発生し続け keydown は発生しない。

などがある。

傾向として keypress イベントに関しては発生が保障できないという点が挙げられる。

しかしながらゲームを作成する上で「今キーが押されている」状態かどうかを入力として用いることが多いため、keypress イベントに変わるキー入力の管理機構が必要になる。幸い (というか当たり前だが) キーを押した瞬間 (それ以後に発生しているかどうかは問題ではない) に keydown イベントが発生することとキーを離した瞬間に keyup イベントが発生する点はどのブラウザでも共通しているので、この 2 つのイベントを監視することで「今キーが押されている」状態かどうかを取得する。

下記にその JavaScript コードを示す。

```
var createKeyInputObject = function(){
  var keyState = [];//キーの状態を持つ配列
  for(var i=0;i<128;i++){
    keyState.push(false);
  }
  var updateKeyState = function(keyCode,isHit){
    keyState[keyCode] = isHit;
  }
  return {
    init:function(){//初期化
      //キーを押したときのイベントの設定
      document.onkeydown = function(ev){
        //イベントパラメータ取得。
        //IE では window.event,
        //それ以外では引数から受け取る。
        var eve = ev ? ev : window.event;
        updateKeyState(eve.keyCode,true);
        //ブラウザデフォルトのキーイベントを殺す。
```

```
      return false;
    }
    //キーを離したときのイベントの設定
    document.onkeyup = function(ev){
      var eve = ev ? ev : window.event;
      updateKeyState(eve.keyCode,false);
      return false;
    }
  },
  getState:function(){//キー状態取得
    return keyState;
  }
 };
}
```

実際にキー入力の状態を取得するときは以下のようなコードで行う。

```
//keyInput オブジェクト生成
var input = createKeyInputObject();
input.init();//初期化。
//現在のキーの状態を取得
var keyState = input.getState();
alert(keyState[90]);
//z キー（キーコード:90）を押している状態なら true,
//押してない状態なら false が表示される。
```

Shift キーや Ctrl キーなどを調べる場合は加えてイベントパラメータの shiftKey プロパティや ctrlKey プロパティなどを調べるコードを updateKeyState() に追加する必要がある。

### 5.2 マウス入力

マウス入力に関するイベントとしては

mousedown　マウスボタンを押した瞬間に発生
mouseup　マウスボタンを離した瞬間に発生
mousemove　マウスが動いているときに発生
mouseover　マウスがイベントを割り当てられた要素上に乗ったときに発生
mouseout　マウスがイベントを割り当てられた要素上から離れたときに発生

がある。

以下、マウスボタンの取得と座標取得について述べていく。

#### 5.2.1 マウスボタン入力の取得

キー入力の場合と異なり、基本的にブラウザ間のイベント発生タイミング等の差異はない。ただし右クリックの取得についてはブラウザによって可能なものと不可能なものとがあり、取得に用いるイベントにも差異がある。よって右クリックを利用するアプリケーションの作成は推奨しない。

一般にボタン入力の取得に関しては mousedown イベントと mouseup イベントを監視すればよい。

#### 5.2.2 マウス座標の取得

マウス座標の取得に関してはブラウザ間の相違が多い。今回は Web ページ左上を原点とするピクセル単位のページ座標についてのみを考える。マウス座標の取得の際は以下の関数を用いてブラウザ間の差異を吸収する。

```
var getMousePositionByPage = function(e){
  return window.createPopup ? {
    x : e.x + document.body.scrollLeft,
    y : e.y + document.body.scrollTop
  }:{
    x : e.pageX,
    y : e.pageY
  };
}
```

実際の使用例は以下のようになる。

```
document.onmousemove = function(ev){
  var eve = ev ? ev : window.event;
  var position = getMousePositionByPage(eve);
  //position.x:ページの左上を原点とするマウスの X 座標
  //position.y:ページの左上を原点とするマウスの Y 座標
}
```

## 6 描画

JavaScript で画像を描画する方法としては

- DOM で動的に HTML を操作する
- canvas 要素の API を使う

といったものが挙げられる。

以下ではこの 2 つの描画方法についてそれぞれ述べていく。

### 6.1 DOM 操作による描画

HTML の汎用ブロック要素である div 要素を動的ないし静的に HTML の木構造に追加し、style プロパティを操作することでその div 要素に適用されている CSS を動的に変更することが出来る。これを利用して画像を描画する。

以下サンプルコード。

```
<!DOCTYPE html>
<html>
<head>
<title>draw by DOM sample</title>
<script type="text/javascript">
var imageName = "sample.png"
window.onload = function(){
```

```
  //div 要素を動的に生成
  var testDiv = document.createElement("div");
  //ページ左上を原点とする絶対位置指定に設定;
  testDiv.style.position = "absolute";
  //X 座標を 300px に設定
  testDiv.style.left = 200 + "px";
  //Y 座標を 200px に設定
  testDiv.style.top = 200 + "px";
  //幅を 100px に設定
  testDiv.style.width = 100 + "px";
  //高さを 100px に設定
  testDiv.style.height = 100 + "px";
  //背景画像を imageName に指定した画像ファイルに設定
  testDiv.style.backgroundImage = "url(" +
                                  imageName + ")";
  //div#base に testDiv を子要素として追加。
  document.getElementById("base")
          .appendChild(testDiv);
}
</script>
</head>
<body>
<div id="base">
</div>
</body>
</html>
```

利点としては

- CSS で許される制御は全般可能
- 文字を表示したいときは innerHTML にテキストを流し込む、document.createTextNode() でテキストノードを生成して appendChild するなどで割と手軽にできる
- スタイル記述を静的 CSS に切り離しスタイル変更をすべてクラス変更で対応するようにすればJavaScript コードとスタイル記述との分離が可能。

難点としては

- 基本的に DOM 操作は時間を食う。
- 大量の画像を表示するのは不適。
- 画像に対して回転などの操作はできない。
- 直線を引くなどの幾何学描画は出来ない。
- 扱うプロパティによってはブラウザによって挙動が異なる場合がある。(CSS 実装の相違・バグ)

その他特徴としては

- 描画が JavaScript に対して非同期に行われる。

といった点が挙げられる。

### 6.2 canvas 要素による描画

HTML の次期仕様 HTML5 ではラスタ描画用要素として canvas 要素という要素が追加されている。JavaScript 向けに描画 API が用意されており、FireFox、Opera、Safari、GoogleChrome で先行実装されている。

canvas 要素の描画 API を用いて画像を描画する場合は画像ファイルをあらかじめロードしておく必要がある(DOM 操作の場合は JavaScript のプロセスとは独立して描画が行われるため画像ロード終了まで JavaScript 側が待機する必要がない)。複数の画像を描画することを考えると画像のローディングを一元的に担うオブジェクトを用意するのが望ましい。

以下に画像ローダオブジェクトの実装を示す。

```
var createImageLoader = function(){
  var imagePool = [];
  var loadedImageCount = 0;
  var registerImage = function(file){
    var im = new Image();
    im.onload = function(){
      loadedImageCount++;
    }
    //キャッシュ対策
    im.src = file + "?"+ new Date().getTime();
    imagePool.push(im);
    return imagePool.length - 1;
  }
  var getImage = function(index){
    return imagePool[index];
  }
  return {
    //ロードする画像の登録
    registerImages:function(images){
      var indexes = {};
      for(obj in images){
        if(images.hasOwnProperty(obj) &&
           typeof(images[obj])!=="function"){
          indexes[obj] = registerImage(images[obj]);
        }
      }
      return indexes;
    },
    //画像がすべてロードされたかどうかを調べる
    checkIsLoaded:function(){
      return (imagePool.length === loadedImageCount);
    },
    //ロードした画像の Image オブジェクトを取得する。
    getImages:function(indexes){
      var images = {};
      for(obj in indexes){
        if(indexes.hasOwnProperty(obj) &&
```

```
          typeof(indexes[obj])!=="function"){
        images[obj] = getImage(indexes[obj]);
      }
    }
    return images;
  }
 }
}
```

実際の使用例は以下のようになる。

```
<!DOCTYPE html>
<html>
<head>
<title>ImageLoader test</title>
<script type="text/javascript"
        src="imageLoader.js"></script>
<script type="text/javascript">
window.onload = function(){
  setTimeout(function(){
    var img = {
      img1 : "sample1.png",
      img2 : "sample2.png"
    }
    var imageLoader = createImageLoader();
    //ロードする画像を登録
    img = imageLoader.registerImages(img);
    return function(){
      //ロードが完了したら描画してループを抜ける。
      if(imageLoader.checkIsLoaded()){
        img = imageLoader.getImages(img);
        //canvas 要素 API のコンテクスト取得。
        var context = document.getElementById("cv")
                              .getContext("2d");
        //img.img1 を描画
        context.drawImage(img.img1,10,0);
        //img.img2 を描画
        context.drawImage(img.img2,20,30);
      }else{
        setTimeout(arguments.callee,10);
      }
    }
  }(),10);
}
</script>
</head>
<body>
<canvas width="100" height="100" id="cv"></canvas>
</body>
</html>
```

canvas 要素の API による描画の利点としては

- 描画が高速
- 画像に対して回転などの操作ができる
- 大量の画像の描画にもある程度は耐えられる
- 幾何学描画についても API が用意されている

難点としては

- ブラウザによって API の対応具合にばらつきがある。
- ブラウザシェアトップの IE が canvas 要素に対応していない。(explorerCanvas など IE で canvas 要素を使えるようにする取り組みはある。)
- 文字描画の API を実装しているブラウザがほとんどない。(現時点で GoogleChrome のみ。対応具合も中途半端。)

その他の特徴としては

- 描画は JavaScript に同期する

といった点が挙げられる。

### 6.3 実際のゲーム開発時の事例

これまでの JavaScript によるゲーム開発での描画選択の事例を挙げておく。

#### 6.3.1 JavaScript で 3D ブロック崩し (2008 工大祭)

描画はすべて DOM 操作で行った。

- 描画するものが比較的少なかった。
- 速度はさほど問題にならなかった。(FPS が低くてもさほど問題にならなかった)

#### 6.3.2 JavaScript で東洋的な弾幕 STG を作る方針で (2009 新歓展示)

描画はほとんど Canvas 要素で行った。

ただし会話の文章は DOM 操作で div 要素にテキストを流し込んで表示した。

- 開発中に変更頻度の高い文章の場合、いちいち文字列を画像化して表示するのはダルい。
- このころは GoogleChrome も文字描画に対応していなかった。

#### 6.3.3 jshooting(2009 工大祭)

描画はほとんど Canvas 要素で行った。

ただし会話の文章は DOM 操作で html を弄って行った。

また、背景描画も DOM 操作で行った。

- 背景は描画面積が広いため Canvas 要素で描画すると時間がかかり、その分他の処理を待たせてしまう → DOM 操作なら非同期なので JavaScript の処理を止めずに済む。
- タイルパターンの背景の描画は CSS で扱うと楽。(background-repeat プロパティを用いる。)

- 背景スクロールの描画は CSS で扱うと楽。(background-position プロパティを用いる。)

## 7 音声再生

JavaScript には音声再生などを行う統一的な API が存在しない。そのためどうにかして JavaScript で音を鳴らせないだろうかと模索していた。

方法として試したものは下記の通り。

- bgsound 要素を DOM 経由で叩く (IE,Opera でのみ動作)
- embed 要素を DOM 経由で叩く
- HTML5 の audio 要素の API を使用する (Opera, Safari でのみ動作)
- Flash(ActionScript) と連携する

これらのうち上 3 つについては動作するブラウザが限定される、音の遅延が激しいなどの問題に直面し放棄した。結局 Flash との連携によって音声再生を実現した。

以下では Flash(ActionScript) と連携して JavaScript で音声再生する方法について述べていく。

### 7.1 ActionScript と JavaScript

本題に入る前に、ActionScript と JavaScript との関係について述べておく。

ActionScript と JavaScript はともに ECMAScript と呼ばれる規格をベースにしているいわば兄弟のような言語である。しかしながら JavaScript が完全に動的型付けであるのに対し ActionScript は変数宣言時に型宣言を要求する、クラスを持たない JavaScript に対し AcitonScript は Java のようなクラスを持っているといった相違が見られ、ActionScript は JavaScript よりも Java っぽい印象を受ける。

### 7.2 Flash と JavaScript の連携

JavaScript と ActionScript の連携を行うには ActionScript の ExternalInterface を用いる。

JavaScript から ActionScript のメソッドを呼ぶ際には ActionScript 側であらかじめ JavaScript から呼び出せるようにするメソッドを ExternalInterface.addCallback() で登録し、JavaScript 側は flash オブジェクトを介して所望のメソッドを呼び出す。

ActionScript から JavaScript の関数を呼び出す場合は ExternalInterface.call() で所望の関数を呼び出す。ただし呼び出せる関数は JavaScript のグローバルスコープから可視なものに限られる。JavaScript 側では特に処理を行う必要はない。

実際に ActionScript と JavaScript で連携を行う際、特に JavaScript から ActionScript のメソッドを呼び出す必要のある場合では、そのメソッドを呼び出した時点で Flash がロードされている必要があるため、Flash のロードが終了したかどうかを監視し、Flash のロードが終了してから ActionScript のメソッドを呼び出すことが保障されるようにコードを記述する必要がある。

以下にその例を示す。

```
//ActionScript 側コード。swf ファイルのファイル名は
test.swf とする。
package{
  import flash.display.Sprite;
  import flash.external.ExternalInterface;
  public class Main extends Sprite
  {
    public function Main():void
    {
      //testFunc を JS 側から呼び出せるように設定。
      ExternalInterface.addCallback("testFunc",testFunc);
      //JS 側の関数 onFlashLoad() を呼び出す。
      ExternalInterface.call("onFlashLoad");
    }
    public function testFunc(str:String):String
    {
      return str + "呼んだ？";
    }
  }
}

//JS 側 (HTML 側) コード
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
var swfId = "test";
var swfFile = "test.swf";
var swfObj = null;
//Flash ロード終了時に呼び出される関数。
var onFlashLoad = function(){
  //IE とそれ以外のブラウザでは
  //swf オブジェクトの取得経路が違う。
  swfObj = (navigator.appName.indexOf("Microsoft") != -1) ?
           window[swfId] : document[swfId];
  //ActionScript 側の testFunc を呼んでみる。
  var result = swfObj.testFunc("お〜い、Flash。：");
  alert(result);
  //"お〜い、Flash。：呼んだ？"と表示される。
}
window.onload = function(){
//IE とそれ以外のブラウザでは swf 挿入に用いる
//HTML 記述が異なるので JavaScript でブラウザ判別を行い
//動的に swf 挿入記述を追加する。
  var insertStr = "";
  if(navigator.appName.indexOf("Microsoft")!==-1){
    insertStr += '<object id="' + swfId +
```

```
                '" classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000">' +
                '<param name="movie" value="' +
                swfFile +
                '" /><param name="allowScriptAccess" value="always" /></object>';
  }else{
    insertStr = '<object type="application/x-shockwave-flash" id="' + swfId +
                '" data=' + swfFile +
                '><param name="allowScriptAccess" value="always" /></object>';
  }
  document.body.innerHTML += insertStr;
}

</script>
</head>
<body>
</body>
</html>
```

### 7.3 JavaScript で音声再生

JavaScript で音声を再生する場合は、Flash の Sound クラス等のラッパーを用意し ExternalInterface.addCallback() で JavaScript 側に提供すればよい。

Flash 側の仕様上、再生できる音声メディアは mp3 形式に限られる。また、SoundChannel の上限は 32 チャンネルである。32 チャンネルすべてで音声が再生されている状態でさらに音声を再生させようとするとバグを誘発するので SoundChannel の管理をしっかり行い、32 チャンネルすべて使用されている場合はチャンネルの空きができるまで新規の音声再生を放棄するか、あるいは強制的に SoundChannel のどれかの再生を停止して新規の音声再生を行うと言った措置を取る必要がある。

## 8 終わりに

クライアントサイドアプリケーションは HTML の次期仕様 HTML5 の勧告・普及に伴ってその可能性が一気に広がっていくと考えられる。今後の技術進展とともに JavaScript の新たな可能性を切り開いていきたい。

## 参考文献


[1] keydown、keypress イベントのブラウザ毎の挙動の違い - Enjoy*Study
http://d.hatena.ne.jp/onozaty/20070801/p1

[2] 第五章　クライアントサイドの技術：マウスの座標を取得する - OpenSpace
http://www.openspc2.org/JavaScript/Ajax/Ajax_study/chapter05/013/index.html