

[概要](#)

[初期化、メインクラス作成](#)

[HTMLファイルの用意](#)

[自分が記述するJSファイルの用意](#)

[メインクラスコンストラクタに引数を渡したい場合](#)

[画面設定](#)

[Canvasの情報を取得 - System#getWidth\(\), System#getHeight\(\)](#)

[フルスクリーンモード - System#setFullScreen\(\)](#)

[ステージを取得 - System#getStage\(\)](#)

[実測FPSを取得 - System#getFps\(\)](#)

[任意クラス作成](#)

[基本](#)

[継承](#)

[クラス階層](#)

[親子関係の操作](#)

[表示オブジェクトの追加 - DisplayObjectContainer#addChild\(\)](#)

[子を削除 - DisplayObjectContainer#removeChild\(\)](#)

[特定深度に表示オブジェクトを追加 - DisplayObjectContainer#addChildAt\(\)](#)

[全ての子を削除 - DisplayObjectContainer#\\_removeAllChild\(\)](#)

[指定した表示オブジェクトが子であるかどうかを調べる - DisplayObjectContainer#contains\(\)](#)

[表示範囲を決めるマスクをかける - DisplayObjectContainer#setMask\(\), DisplayObjectContainer#clearMask\(\)](#)

[表示オブジェクトの操作](#)

[移動 - DisplayObject#setX\(\), DisplayObject#setY\(\)](#)

[座標を取得 - DisplayObject#getX\(\), DisplayObject#getY\(\)](#)

[サイズ\(幅, 高さ\)を指定 - DisplayObject#setWidth\(\), DisplayObject#setHeight\(\)](#)

[サイズ\(幅, 高さ\)を取得 - DisplayObject#getWidth\(\), DisplayObject#getHeight\(\)](#)

[scale値\(拡大縮小率\)を指定 - DisplayObject#setScaleX\(\), DisplayObject#setScaleY\(\)](#)

[scale値\(拡大縮小率\)を取得 - DisplayObject#getScaleX\(\), DisplayObject#getScaleY\(\)](#)

[オブジェクトの表示/非表示 - DisplayObject#setVisible\(\)](#)

[オブジェクトの表示/非表示状態の取得 - DisplayObject#setVisible\(\)](#)

[透明度を指定 - DisplayObject#setAlpha\(\)](#)

[透明度を取得 - DisplayObject#getAlpha\(\)](#)

[角度を指定\(回転させる\) - DisplayObject#setRotation\(\)](#)

[角度を取得 - DisplayObject#getRotation\(\)](#)

[原点からの座標\(絶対座標\)を取得 - DisplayObject#getAlignX\(\), DisplayObject#getAlignY\(\)](#)

[グローバル座標をローカル座標に変換 - DisplayObject#globalToLocal\(\)](#)

[ローカル座標をグローバル座標に変換 - DisplayObject#localToGlobal\(\)](#)

[親オブジェクトを取得 - DisplayObject#getParent\(\)](#)

[図形や線を描くShapeクラス](#)

[Shapeオブジェクトの作成](#)

[塗りつぶし指定 - Shape#beginFill\(\), Shape#endFill\(\)](#)

[線\(ストローク\)指定 - Shape#beginStroke\(\), Shape#endStroke\(\)](#)

[線を描く - Shape#lineTo\(\)](#)

[円を描く - Shape#drawCircle\(\)](#)

[矩形を描く - Shape#drawRect\(\)](#)

[画像を扱う](#)

[画像の取得、表示](#)

[画像の複製 - Image#duplicate\(\)](#)

[画像の拡大縮小 - Image#changeScale\(\)](#)  
[画像のサイズ変更 - Image#changeSize\(\)](#)  
[画像に色を加える - Image#changeColor\(\)](#)  
[画像サイズを取得 - Image#getWidth\(\), Image#getHeight\(\)](#)  
[画像ファイルパスを取得 - Image#getPath\(\)](#)

## [MovieClip](#)

### [キーフレーム](#)

#### [イージング](#)

[任意フレームから再生 - MovieClip#gotoAndPlay\(\)](#)

[任意フレームで停止 - MovieClip#gotoAndStop\(\)](#)

## [SheetMovieClip](#)

### [アニメーション再生](#)

[指定したフレームから再生 - SheetMovieClip#gotoAndPlay\(\)](#)

[指定したフレームで停止 - SheetMovieClip#gotoAndStop\(\)](#)

## [テキスト](#)

[文字列を表示する - TextField#setText\(\)](#)  
[フォント情報をセットする - TextField#setFont\(\)](#)  
[フォントの色を指定 - TextField#setColor\(\)](#)

## [Ajax](#)

[動的にデータを読み込む](#)

## [タッチイベント](#)

[各タッチ処理](#)

## [Utils関数](#)

[文字列の出力 - trace\(\)](#)  
[thisの参照を決める - bind\(\)](#)  
[配列のコピー - copyArray\(\)](#)

## [ユーザーエージェント](#)

---

## 概要

AS3ライクに記述できるJavaScriptゲームライブラリです。(ライセンスはMIT License)  
ダウンロード - <https://github.com/DeNADev/Arctic.js>

## 初期化、メインクラス作成

### HTMLファイルの用意

```
<html>
<head>
<script type="text/javascript" src="js/arctic.js"></script>
<script type="text/javascript" src="js/game.js"></script>
</head>
<body>
<canvas id="canvas"></canvas>
</body>
</html>
```

HTMLファイルを用意して、ライブラリであるarctic.js(arctic.min.js)、自分がコードを書く任意js(game.js)を読み込ませます。  
canvasタグを配置し、id名を付けておきます。

## 自分が記述するJSファイルの用意

```
(function() {
// arcGameクラスを継承したメインクラス
var Main = arc.Class.create(arc.Game, {
// コンストラクタ(必要なら引数を指定)
initialize:function()
{
},
// ENTER_FRAME(毎フレーム呼ばれるメソッド)
update:function()
{
}
});

// DOMの構築が終わった後にゲーム開始
window.addEventListener("DOMContentLoaded", function(event) {
// arc.System(幅, 高さ, canvasタグのid名);
var system = new arc.System(320, 320, "canvas");
system.setGameClass(Main); // メインクラス名を指定

// 読み込みもうとするデータ群を一つ読みこむたびに呼ばれる
system.addEventListener(arc.Event.PROGRESS, function(event) {
// event.loaded = 現在読み込みが終わったデータ数
// event.total = 読みこもうとするデータ群のトータル数
arc.util.trace(event.loaded + "/" + event.total);
});

// データ読み込みが終わった時に呼ばれる
system.addEventListener(arc.Event.COMPLETE, function() {
});

// 読み込みたいデータをhtmlファイルからの相対パスで記述、配列で指定する
system.load(["images/bg.jpg"]);

// 読み込むデータが無いならload()の代わりにstart()を記述する。
// 逆に、読み込むデータがあるならstart()を記述する必要はない
// system.start();
}, false);
})();
```

## メインクラスコンストラクタに引数を渡したい場合

```
(function() {
var Main = arc.Class.create(arc.Game, {
initialize:function(params)
{
arc.util.trace(params.title, params.subTitle);
}
});

window.addEventListener("DOMContentLoaded", function(event) {
```

```
var system = new arc.System(320, 320, "canvas");
system.setGameClass(Main, {title:"JSクエスト", subTitle:"そして伝説へ..."});
    system.start();
    }, false);
    })();
```

メインクラスのコンストラクタ(initialize)に引数を指定することも出来ませんが、一つの引数しか指定が出来ません。複数の値を渡したい場合は、オブジェクトリテラルや配列で指定する必要があります。

## 画面設定

### Canvasの情報を取得 - System#getWidth(), System#getHeight()

外部から

```
window.addEventListener("DOMContentLoaded", function(event) {
    var system = new arc.System(320, 320, "canvas");
    system.setGameClass(Main);
    arc.util.trace(system.getWidth(), system.getHeight()); // 幅, 高さ
    }, false);
    })();
```

内部から

```
var Main = arc.Class.create(arc.Game, {
    initialize: function()
    {
        arc.util.trace(this._system.getWidth()); // Canvasの幅
        arc.util.trace(this._system.getHeight()); // Canvasの高さ
    }
});
```

### フルスクリーンモード - System#setFullScreen()

```
this._system.setFullScreen();
```

キャンバスの幅がデバイスの幅になるように調整される。(回転した時も適用)

### ステージを取得 - System#getStage()

```
this._system.getStage();
```

表示リストのルートオブジェクトであるステージを取得する。

### 実測FPSを取得 - System#getFps()

```
(function() {
    var Main = arc.Class.create(arc.Game, {
        initialize: function()
        {
```

```
        },
        update: function()
        {
            arc.util.trace(this._system.getFps());
        }
    });
```

## 任意クラス作成

### 基本

```
var Actor = arc.Class.create({
    _name: null,

    // コンストラクタ
    initialize: function()
    {
        // クラス作成時に呼ばれる
    },

    // setter/getter
    setName: function(name)
    {
        this._name = name;
    },

    getName: function()
    {
        return this._name;
    }
});

var actor = new Actor()
actor.setName("勇者");
arc.util.trace(actor.getName()); // 勇者
```

create()に指定するのはオブジェクト(リテラル)なので、カンマ区切りで記述しないとエラーが出る。

### 継承

```
var Actor = arc.Class.create({
    _name: null,
    _str: null,

    attack: function()
    {
        arc.util.trace(_name + "は" + _str + "のダメージを与えた!!");
    }
});

// create()の第一引数に親クラス名を記述する
var Yuusya = arc.Class.create(Actor, {
    initialize: function()
    {
        _name = "勇者";
    }
});
```

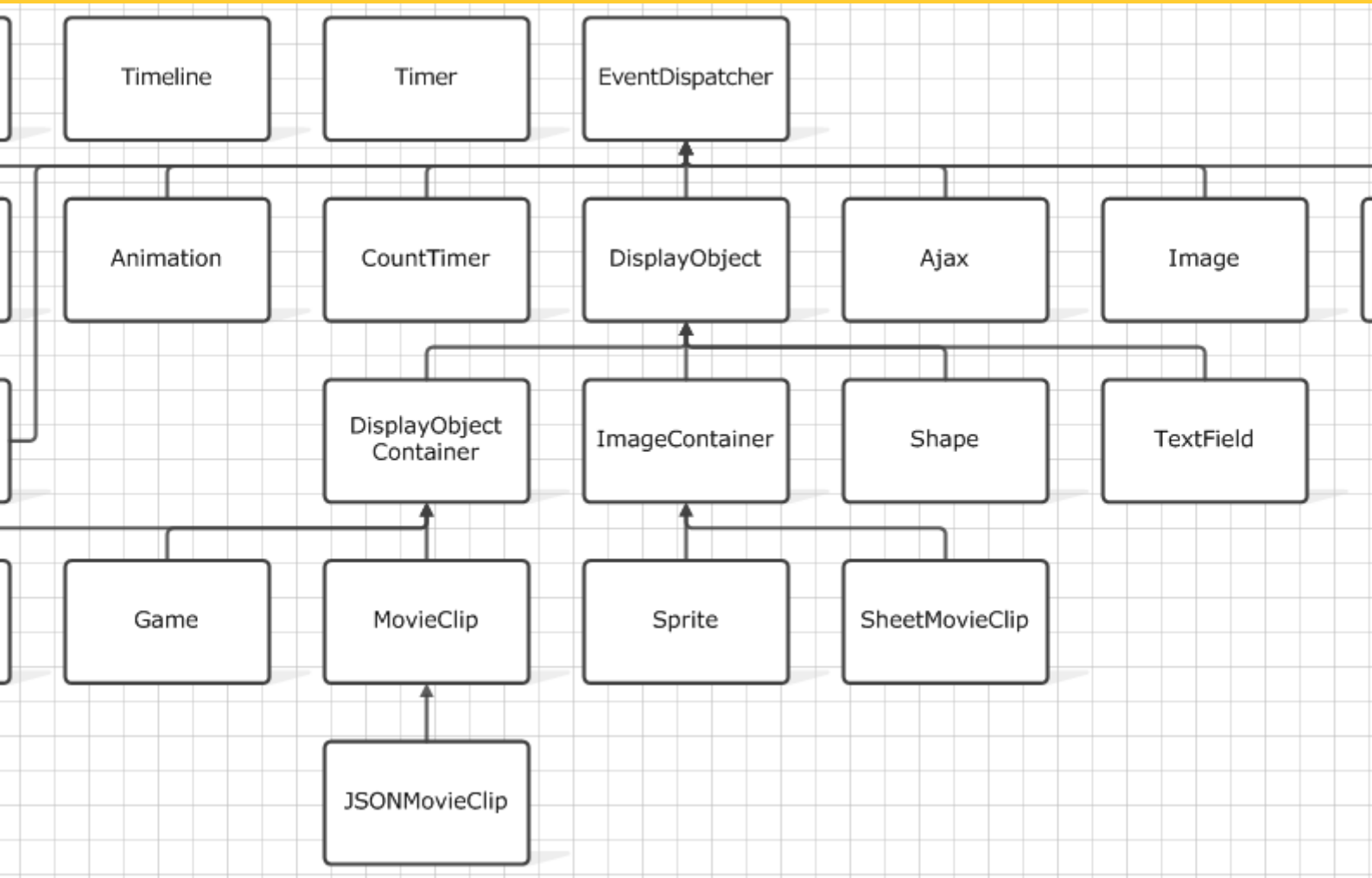
```

        _str = 30;
    }
});

var yuusya = new Yuusya();
yuusya.attack(); // 勇者は30のダメージを与えた!!

```

## クラス階層



## 親子関係の操作

### 表示オブジェクトの追加 - DisplayObjectContainer#addChild()

```

(function() {
    var Main = arc.Class.create(arc.Game, {
        initialize: function(params) {
            var image = new arc.display.Sprite(this._system.getImage("images/icon.png"));
            this.addChild(image);
        }
    });

    window.addEventListener("DOMContentLoaded", function(event) {
        var system = new arc.System(320, 320, "canvas");
    });

```

```
system.setGameClass(Main);
system.load(["images/icon.png"]);
}, false);
})();
```

A.addChild(B)で、BをAの子として追加する。

```
var shape = new arc.display.Shape();
shape.beginFill(0xED1A3D);
shape.drawCircle(100, 100, 100);
shape.endFill();
this.addChild(shape);

var shape2 = new arc.display.Shape();
shape2.beginFill(0x009AD6);
shape2.drawCircle(150, 100, 100);
shape2.endFill();
this.addChild(shape2);
```

Flashと同じように同階層での重ね順があり、後からaddChild()したオブジェクトほど上に表示されるようになっている。

### 子を削除 - DisplayObjectContainer#removeChild()

```
this.removeChild(オブジェクト);
```

引数指定した表示オブジェクトをremoveChildの呼び出し元オブジェクトから削除する。  
addChild()していないオブジェクトを指定してもエラーは出ない。

### 特定深度に表示オブジェクトを追加 - DisplayObjectContainer#addChildAt()

```
// shapeの深度 = 1
var shape = new arc.display.Shape();
shape.beginFill(0xED1A3D);
shape.drawCircle(100, 100, 100);
shape.endFill();
this.addChild(shape);

// shape2の深度 = 0
var shape2 = new arc.display.Shape();
shape2.beginFill(0x009AD6);
shape2.drawCircle(150, 100, 100);
shape2.endFill();
this.addChildAt(shape2, 0); // 表示オブジェクト, 深度
```

子の表示順序の事を深度と呼び、addChild()するたびに0から連番を振っていく。  
addChildAt()では追加する際、深度を一緒に指定することが出来る。  
AS3と違い、マイナス値や子の総数以上の値を指定することが可能？(おそらく非推奨)

addChild()のみ使用、もしくはaddChildAt()で正の深度しか指定していない場合は、深度0を指定することで常に表示オブジェクトをそのレイヤーでの最前面に描画することが可能。

### 全ての子を削除 - DisplayObjectContainer#\_removeAllChild()

```
this._removeAllChild();
```

A.\_removeAllChild()で、Aの子を全て削除する。

## 指定した表示オブジェクトが子であるかどうかを調べる - DisplayObjectContainer#contains()

```
var shape = new arc.display.Shape();
shape.beginFill(0xED1A3D);
shape.drawCircle(100, 100, 100);
shape.endFill();
this.addChild(shape);

arc.util.trace(this.contains(shape)); // true
```

A.contains(B)で、Aの子がBであればtrue、でなければfalseが返却される。

## 表示範囲を決めるマスクをかける - DisplayObjectContainer#setMask(), DisplayObjectContainer#clearMask()

```
var shape = new arc.display.Shape();
shape.beginFill(0xED1A3D);
shape.drawCircle(100, 100, 100);
shape.endFill();
this.addChild(shape);

this.setMask(0, 0, 80, 80); // (0, 0) ~ (80, 80)までの範囲しか表示しない

// this.clearMask(); // マスクの解除
```

もぐらたたきゲームで穴の下が表示されないようにするなどのような表示範囲を限定したい場合に利用できる。マスクを解除する場合は、clearMask()を呼び出す。

## 表示オブジェクトの操作

DisplayObjectを継承しているクラスの操作

### 移動 - DisplayObject#setX(), DisplayObject#setY()

```
(function() {
var Main = arc.Class.create(arc.Game, {
  initialize: function()
  {
var shape = new arc.display.Shape();
shape.beginFill(0xED1A3D, 1.0);
shape.drawCircle(0, 0, 30);
shape.endFill();
this.addChild(shape);

// shapeで描いた円を(100, 50)まで移動する
shape.setX(100);
shape.setY(50);
}
});
```

```
window.addEventListener("DOMContentLoaded", function(event) {  
    var system = new arc.System(320, 320, "canvas");  
    system.setGameClass(Main);  
    system.start();  
}, false);  
})();
```

setX(移動先x座標), setY(移動先y座標);

### 座標を取得 - DisplayObject#getX(), DisplayObject#getY()

```
var shape = new arc.display.Shape();  
shape.beginFill(0xED1A3D, 1.0);  
shape.drawCircle(0, 0, 30);  
shape.endFill();  
this.addChild(shape);  
  
// shapeで描いた円を(100, 50)まで移動する  
shape.setX(100);  
shape.setY(50);  
  
arc.util.trace(shape.getX(), shape.getY()); // 100, 50
```

### サイズ(幅, 高さ)を指定 - DisplayObject#setWidth(), DisplayObject#setHeight()

```
var shape = new arc.display.Shape();  
shape.beginFill(0xED1A3D, 1.0);  
shape.drawCircle(0, 0, 30);  
shape.endFill();  
this.addChild(shape);  
  
// shapeのサイズを(200, 250)にする  
shape.setWidth(200);  
shape.setHeight(250);
```

指定したサイズに合わせて描画しているものが拡大縮小されます。

### サイズ(幅, 高さ)を取得 - DisplayObject#getWidth(), DisplayObject#getHeight()

```
var shape = new arc.display.Shape();  
shape.beginFill(0xED1A3D, 1.0);  
shape.drawCircle(0, 0, 30);  
shape.endFill();  
this.addChild(shape);  
  
arc.util.trace(shape.getWidth(), shape.getHeight()); // 30, 30
```

### scale値(拡大縮小率)を指定 - DisplayObject#setScaleX(), DisplayObject#setScaleY()

```
var shape = new arc.display.Shape();  
shape.beginFill(0xED1A3D, 1.0);
```

```
shape.drawCircle(0, 0, 30);
shape.endFill();
this.addChild(shape);

// 横方向に2倍, 縦方向に3倍に拡大
shape.setScaleX(2.0);
shape.setScaleY(3.0);
```

## scale値(拡大縮小率)を取得 - DisplayObject#getScaleX(), DisplayObject#getScaleY()

```
var shape = new arc.display.Shape();
shape.beginFill(0xED1A3D, 1.0);
shape.drawCircle(0, 0, 30);
shape.endFill();
this.addChild(shape);

// 横方向に2倍, 縦方向に3倍に拡大
shape.setScaleX(2.0);
shape.setScaleY(3.0);

arc.util.trace(shape.getScaleX(), shape.getScaleY()); // 2, 3
```

## オブジェクトの表示/非表示 - DisplayObject#setVisible()

```
var shape = new arc.display.Shape();
shape.beginFill(0xED1A3D, 1.0);
shape.drawCircle(0, 0, 30);
shape.endFill();
this.addChild(shape);

shape.visible(true); // 表示
shape.visible(false); // 非表示
```

## オブジェクトの表示/非表示状態の取得 - DisplayObject#setVisible()

```
arc.util.trace(shape.getVisible());
```

表示しているならtrue、非表示ならfalseが返却されます。

## 透明度を指定 - DisplayObject#setAlpha()

```
var shape = new arc.display.Shape();
shape.beginFill(0xED1A3D, 1.0);
shape.drawCircle(0, 0, 30);
shape.endFill();
this.addChild(shape);

shape.setAlpha(0.5);
```

setAlpha(value)で透明度を指定します。(value = 0.0 ~ 1.0)

## 透明度を取得 - DisplayObject#getAlpha()

```
shape.getAlpha();
```

## 角度を指定(回転させる) - DisplayObject#setRotation()

```
var image = this._system.getImage("images/icon.png");  
var sprite = new arc.display.Sprite(image);  
this.addChild(sprite);  
sprite.setRotation(45); // spriteの(x, y)を軸に45度回転
```

Shapeで描いたものはsetRotation()を使用しても回転しない?(要検証)

## 角度を取得 - DisplayObject#getRotation()

```
arc.util.trace(sprite.getRotation());
```

## 原点からの座標(絶対座標)を取得 - DisplayObject#getAlignX(), DisplayObject#getAlignY()

```
arc.util.trace(sprite.getAlignX(), sprite.getAlignY());
```

## グローバル座標をローカル座標に変換 - DisplayObject#globalToLocal()

```
(function() {  
    var Main = arc.Class.create(arc.Game, {  
        initialize: function(params)  
        {  
            this.setX(10);  
            var shape = new arc.display.Shape();  
            arc.util.trace(this.globalToLocal(20, 20)); // [10, 20]  
        }  
    });  
});
```

A.globalToLocal(x, y)で、グローバル座標(x, y)をAのローカル座標に変換したものを[x, y]として返却します。

## ローカル座標をグローバル座標に変換 - DisplayObject#localToGlobal()

```
this.setX(10);  
var shape = new arc.display.Shape();  
arc.util.trace(this.localToGlobal(0, 0)); // [10, 0]とおもいきや[0, 0]が返ってくる
```

A.localToGlobal(x, y)で、Aのローカル座標(x, y)をグローバル座標に変換したものを[x, y]として返却します...が、ちょっと期待通りの値が返ってこないのがバグ?

## 親オブジェクトを取得 - DisplayObject#getParent()

```
        (function() {
    var Main = arc.Class.create(arc.Game, {
        initialize:function(params)
        {
            var shape = new arc.display.Shape();
            this.addChild(shape);
            arc.util.trace(shape.getParent() instanceof Main); // true
        }
    });
```

## 図形や線を描くShapeクラス

### Shapeオブジェクトの作成

```
var shape = new arc.display.Shape();
```

## 塗りつぶし指定 - Shape#beginFill(), Shape#endFill()

```
var shape = new arc.display.Shape();
shape.beginFill(0xED1A3D, 0.4); // 塗りつぶしの色, 透明度
shape.drawCircle(100, 100, 100);
shape.drawRect(20, 30, 40, 50);
shape.endFill(); // 塗りつぶしの設定を閉じる
this.addChild(shape);
```

## 線(ストローク)指定 - Shape#beginStroke(), Shape#endStroke()

```
var shape = new arc.display.Shape();
shape.beginStroke(2.0, 0x0, 1.0); // 線の太さ, 色, 透明度
shape.drawCircle(100, 100, 100);
shape.drawRect(20, 30, 40, 50);
shape.endStroke(); // 線の設定を閉じる
this.addChild(shape);
```

## 線を描く - Shape#lineTo()

```
var shape = new arc.display.Shape();
shape.beginStroke(2.0, 0xED1A3D, 1.0); // 線の太さ, 色, 透明度
shape.moveTo(100, 100);
shape.lineTo(200, 200); // (100, 100) ~ (200, 200)まで太さ2.0の赤線を引く
shape.endStroke(); // 線の設定を閉じる
this.addChild(shape);
```

## 円を描く - Shape#drawCircle()

```
var shape = new arc.display.Shape();
shape.beginStroke(2.0, 0xED1A3D, 1.0); // 線の太さ, 色, 透明度
shape.drawCircle(100, 100, 50); // (100, 100)を中心とした半径50pxの円を描く
shape.endStroke(); // 線の設定を閉じる
this.addChild(shape);
```

drawCircle(円の中心x座標, 円の中心y座標, 半径)

## 矩形を描く - Shape#drawRect()

```
var shape = new arc.display.Shape();
shape.beginStroke(2.0, 0xED1A3D, 1.0); // 線の太さ, 色, 透明度
shape.drawRect(100, 100, 50, 50); // (100, 100) ~ (150, 150)範囲の矩形を描く
shape.endStroke(); // 線の設定を閉じる
this.addChild(shape);
```

drawRect(矩形の左上x座標, 矩形の左上y座標, 幅, 高さ)

## 画像を扱う

### 画像の取得、表示

```
(function() {
    var Main = arc.Class.create(arc.Game, {
        initialize: function(params) {
            var image = this._system.getImage("images/icon.png"); // 画像の取得
            this.addChild(new arc.display.Sprite(image)); // Spriteに内包、画面に表示
        }
    });

    window.addEventListener("DOMContentLoaded", function(event) {
        var system = new arc.System(320, 320, "canvas");
        system.setGameClass(Main);
        system.load(["images/icon.png"]);
    }, false);
})();
```

### 画像の複製 - Image#duplicate()

```
duplicate: function(){
    var newImg = new display.Image(this._data);
    //newImg._setData(this._data);
    return newImg;
},
```

画像を複製するduplicate()ですが、バグがあるので上記のように書き換えてください。(今後、修正された場合は書き換えなくていいです)

```
var src = this._system.getImage("images/icon.png");
    var dest = src.duplicate();
    this.addChild(new arc.display.Sprite(src));
    var destSprite = new arc.display.Sprite(dest);
    destSprite.setX(100);
    this.addChild(destSprite);
```

読み込んだicon.pngを複製しています。AS3でいうとBitmapData#clone()が機能的に近いでしょうか。

### 画像の拡大縮小 - Image#changeScale()

```
var image = this._system.getImage("images/icon.png");
    image.changeScale(2.0, 4.0); // 2倍, 4倍
    this.addChild(new arc.display.Sprite(image));
```

Image#changeScale(x方向のscale値, y方向のscale値)

### 画像のサイズ変更 - Image#changeSize()

```
var image = this._system.getImage("images/icon.png");
    image.changeSize(300, 50); // 300*50pxに変更する
    this.addChild(new arc.display.Sprite(image));
```

changeSize(横サイズpx, 縦サイズpx)

### 画像に色を加える - Image#changeColor()

```
var image = this._system.getImage("images/icon.png");
    image.changeColor(0xED1A3D, 0.5); // 色, 適用率(0.0~1.0)
    this.addChild(new arc.display.Sprite(image));
```

changeColor(色, 適用率)で、画像に色を加えます。

適用率が0.0だと指定した色がまったく適用されず、1.0だと指定した色そのものになります。

具体的な計算式としては、色をRGBに分解し、色の差 \* 適用率を画像の各RGBに加えます。(src + (dest - src) \* density)

### 画像サイズを取得 - Image#getWidth(), Image#getHeight()

```
var image = this._system.getImage("images/icon.png");
    arc.util.trace(image.getWidth(), image.getHeight());
```

### 画像ファイルパスを取得 - Image#getPath()

```
var image = this._system.getImage("images/icon.png");
    arc.util.trace(image.getPath());
```

[file:///](#) ~ 形式のパスを取得します。

## MovieClip

フレーム番号とプロパティを指定することで擬似的なタイムラインを作成することが出来るクラス。

### キーフレーム

```
        (function() {
            var Main = arc.Class.create(arc.Game, {
                initialize: function()
                {
                    // 対象のオブジェクトを用意
                    var shape = new arc.display.Shape();
                    shape.beginFill(0xED1A3D, 1.0);
                    shape.drawRect(0, 0, 100, 100);
                    shape.endFill();

                    // FPS, ループ再生を行うか, addChild()などで追加された際に自動再生するかどうか
                    var mc = new arc.display.MovieClip(30, true, true);
                    // MovieClipオブジェクトに対象のオブジェクトを指定する
                    mc.addChild(shape, {
                        // 10フレーム目に非表示
                        10: {visible: false},
                        // 20フレーム目に表示
                        20: {visible: true}
                    });
                    // MovieClipをaddChild(), コンストラクタ第三引数がtrueだと自動再生される
                    this.addChild(mc);

                    // mc.play(); // 任意のタイミングで再生したい場合はplay()を呼び出す
                }
            });

            window.addEventListener("DOMContentLoaded", function(event) {
                var system = new arc.System(320, 320, "canvas");
                system.setGameClass(Main);
                system.start();
            }, false);
        })();
```

### イージング

```
        var mc = new arc.display.MovieClip(30, false, true);
        mc.addChild(shape, {
            // 5~20の15フレーム掛けて、サイズを0.5~1.0までイージング
            5: {scaleX: 0.5, scaleY: 0.5, transition: arc.anim.Transition.SINE_OUT},
            20: {scaleX: 1.0, scaleY: 1.0}
        });
        this.addChild(mc);
```

### 任意フレームから再生 - MovieClip#gotoAndPlay()

```
mc.gotoAndPlay(15); // 15フレーム目から自動再生
```

## 任意フレームで停止 - MovieClip#gotoAndStop()

```
mc.gotoAndStop(15); // 15フレーム目で停止
```

今の仕様では15フレーム目に飛んでも、15フレームまでの処理は実行されない。

## SheetMovieClip



のようなアニメーションパターンが並んだ画像を切り出し、再生を行うクラス。

## アニメーション再生

```
(function() {
    var Main = arc.Class.create(arc.Game, {
        initialize: function()
        {
            // アニメーション画像を読み込む
            var mcImage = this._system.getImage('images/chip.png');
            // MovieClip作成
            var mc = new arc.display.SheetMovieClip(mcImage, 14, 30, true, false);
            this.addChild(mc);
            // 再生
            mc.play();
        }
    });

    window.addEventListener("DOMContentLoaded", function(event) {
        var system = new arc.System(320, 320, "canvas");
        system.setGameClass(Main);
        system.load(["images/chip.png"]);
    }, false);
})();
```

SheetMovieClip(data, frameWidth, fps, shouldLoop, shouldHide);

- data アニメーション画像が入ったImageオブジェクト
- frameWidth 一フレームの幅(アニメーション1パターンの幅)
- fps アニメーションを切り替えるFPS。例えば10FPSなら1秒間に10回アニメーションを切り替える
- shouldLoop ループ再生するならtrue, しないならfalse
- shouldHide 停止中は非表示にするならtrue, しないならfalse

## 指定したフレームから再生 - SheetMovieClip#gotoAndPlay()

```
var mcImage = this._system.getImage('images/chip.png');
var mc = new arc.display.SheetMovieClip(mcImage, 14, 30, true, false);
this.addChild(mc);
// 2フレーム目から再生
mc.gotoAndPlay(2);
```

## 指定したフレームで停止 - SheetMovieClip#gotoAndStop()

```
var mcImage = this._system.getImage('images/chip.png');
var mc = new arc.display.SheetMovieClip(mcImage, 14, 30, true, false);
    this.addChild(mc);
    // 2フレーム目で停止
    mc.gotoAndStop(2);
```

## テキスト

### 文字列を表示する - TextField#setText()

```
var tf = new arc.display.TextField();
tf.setText("勇者の攻撃!!");
    this.addChild(tf);
```

デフォルトでは、"sans-serif 10px"で表示されます。

### フォント情報をセットする - TextField#setFont()

```
var tf = new arc.display.TextField();
tf.setFont("メイリオ", 20, true);
tf.setText("勇者の攻撃!!");
    this.addChild(tf);
```

setFont(フォント名, フォントサイズ, 太字にするか(true/false))

AS3と違い、テキストを設定した後でフォント情報をセットしても、テキストにフォント情報が反映されます。

### フォントの色を指定 - TextField#setColor()

```
var tf = new arc.display.TextField();
tf.setColor(0xED1A3D); // 赤
tf.setText("勇者の攻撃!!");

    this.addChild(tf);
```

## Ajax

### 動的にデータを読み込む

```
(function() {
var Main = arc.Class.create(arc.Game, {
    initialize:function()
    {
```

```

        var ajax = new arc.Ajax();
        // 読み込みが終わったときに呼び出すメソッドを指定する
        ajax.addEventListener(arc.Event.COMPLETE, this.onComplete);
        // GET/POSTの指定, デフォルトは"GET"
        ajax.setMethod("POST");
        // URLのデータを動的に読み込む
        ajax.load("ajax.txt");
    },

    onComplete: function(event)
    {
        arc.util.trace(this.getResponseText()); // ajax.txtの中身をテキストとして扱う
        arc.util.trace(this.getResponseJSON()); // ajax.txtの中身をJSONとして扱う
        arc.util.trace(this.getURL()); // URL(ajax.txt)を取得
        this.unload(); // 読み込みが終わったので, Ajaxのイベント/データを削除
    }
});

```

## タッチイベント

### 各タッチ処理

```

(function() {
    var Main = arc.Class.create(arc.Game, {
        initialize: function()
        {
            var shape = new arc.display.Shape();
            shape.beginFill(0xED1A3D, 1.0);
            shape.drawRect(0, 0, 100, 100);
            shape.endFill();
            this.addChild(shape);

            this.addEventListener(arc.Event.TOUCH_START, this.touchStartHandler);
            this.addEventListener(arc.Event.TOUCH_MOVE, this.touchMoveHandler);
            this.addEventListener(arc.Event.TOUCH_END, this.touchEndHandler);
        },

        touchStartHandler: function(event)
        {
            arc.util.trace("指が触れた");
        },

        touchMoveHandler: function(event)
        {
            arc.util.trace("タッチ中に指が動いている");
        },

        touchEndHandler: function(event)
        {
            arc.util.trace("指が離れた");
        }
    });

    window.addEventListener("DOMContentLoaded", function(event) {
        var system = new arc.System(320, 320, "canvas");
        system.setGameClass(Main);
        system.start();
    }, false);
})();

```

arc.Event.TOUCH\_START 画面に指が触れたときに登録したメソッドを呼び出す  
arc.Event.TOUCH\_MOVE 画面上で指を動かしたときに登録したメソッドを呼び出す  
arc.Event.TOUCH\_END 画面から指が離れたときに登録したメソッドを呼び出す

## Utils関数

### 文字列の出力 - trace()

```
arc.util.trace("test");  
arc.util.trace("javascript", "flash");
```

渡した文字列引数をFirebugなどで利用されるconsole.log()で出力。  
可変長引数なので、カンマ区切りで複数の文字列を指定可能。

### thisの参照を決める - bind()

```
function putMessage(str)  
{  
  arc.util.trace(this.message);  
}  
  
var Player = function ()  
{  
  this.message = "勇者のターン";  
}  
  
var Enemy = function ()  
{  
  this.message = "敵のターン";  
}  
  
var player = new Player();  
var enemy = new Enemy();  
arc.util.bind(putMessage, player)(); // 勇者のターン  
arc.util.bind(putMessage, enemy)(); // 敵のターン
```

arc.util.bind(関数オブジェクト, オブジェクト)

で、関数内のthisでオブジェクトを参照できるようにバインドする

### 配列のコピー - copyArray()

```
var src = ["HP", "MP"];  
var dest = arc.util.copyArray(src);  
dest[1] = "TP";  
  
arc.util.trace(src); // ["HP", "MP"]  
arc.util.trace(dest); // ["HP", "TP"]
```

## ユーザーエージェント

```
arc.util.trace(arc.ua.isiPhone); // iPhone
arc.util.trace(arc.ua.isiPhone4); // iPhone4
arc.util.trace(arc.ua.isiPad); // iPad
arc.util.trace(arc.ua.isiOS); // iOS(iPhoneかiPadだったら)
arc.util.trace(arc.ua.isiOS3); // iOS3
arc.util.trace(arc.ua.isAndroid); // Android
arc.util.trace(arc.ua.isAndroid2_1); // Android2.1
arc.util.trace(arc.ua.isMobile); // モバイル端末(iOSかAndroidだったら)
```

各プロパティの端末ならtrue、でなかったらfalseが返却されます。