

# JavaScriptで オブジェクト指向

---

2009/3/12

WEBシステムグループ第八技術チーム

嶋崎 聖

復習

# 前回の復習（リテラル）

---

配列

```
arr = ["value"];
```

オブジェクト

```
Obj = { key:"value"};
```

関数

```
func =  
function(){//処理 };
```

# 前回の復習（プロパティ）

---

```
var obj = {} ; // new Object()のリテラル  
obj.prop = "hoge";  
alert(obj.prop); // hoge
```

- オブジェクトにはプロパティを追加出来る
- オブジェクトとプロパティの関係はドットで表す

# 前回の復習（メソッド）

---

```
var obj = {} ; // new Object()のリテラル  
obj.meth = function(){ alert("hoge"); };  
obj.meth() // hoge
```

- オブジェクトにはメソッドを追加出来る
- オブジェクトとメソッドの関係はドットで表す

復習

終わりの

# アジェンダ

---

- JavaScriptのオブジェクト指向
- オブジェクト指向実装
  - 継承
  - オーバーロード
  - オーバーライド
  - プライベート変数

# JavaScriptの オブジェクト指向

# オブジェクト指向の分類

---

- クラスベース

設計図（クラス）があり、  
それをもとに実体（インスタンス）を作成し、  
オブジェクトを操作する。

- プロトタイプベース

基礎（プロトタイプ）があり、  
それをもとに追加や変更をし、  
オブジェクトを操作する。

JavaScriptはこちら。

# JavaScriptのオブジェクト指向

---

- インスタンス作成
  - 関数に「new」を付ける

```
function hoge(){}  
var inst = new hoge();
```

- でもこれならインスタンスを作る意味がない  
(だってただのオブジェクトだから)

# コンストラクタ

---

- オブジェクトの初期値（みたいなもの）

```
function hoge(){  
  this.x = "123";  
  this.y =  
    function(){alert(this.x);};  
}  
  
inst = new hoge();  
inst.y(); // 123
```

- thisは呼び出し元オブジェクトをさす「キーワード」

# コンストラクタ 2

---

- でも同じことを何度も処理するのは効率が悪い

```
function hoge(){
  this.x = "123";
  this.y =
    function(){alert(this.x);};
}
inst1 = new hoge();
inst1.y(); // 123
inst2 = new hoge();
inst2.x = "456";
inst2.y(); // 456
```

- yメソッドはまったく同じ（個別に所有する必要がない）

# prototypeプロパティ

---

- 関数にprototypeプロパティをセットする

```
function hoge(){}  
hoge.prototype.x = "123";  
var inst = new hoge();  
alert(inst.hoge); // 123
```

- インスタンスがprototypeプロパティを参照する

# prototypeプロパティ

---

- 関数でもOK

```
function hoge(){}  
hoge.prototype.x =  
    function(){alert("hoge");};  
var inst = new hoge();  
inst.x() // hoge
```

- メソッドであっても同様に、  
インスタンスがprototypeプロパティを参照する

# prototypeプロパティ

---

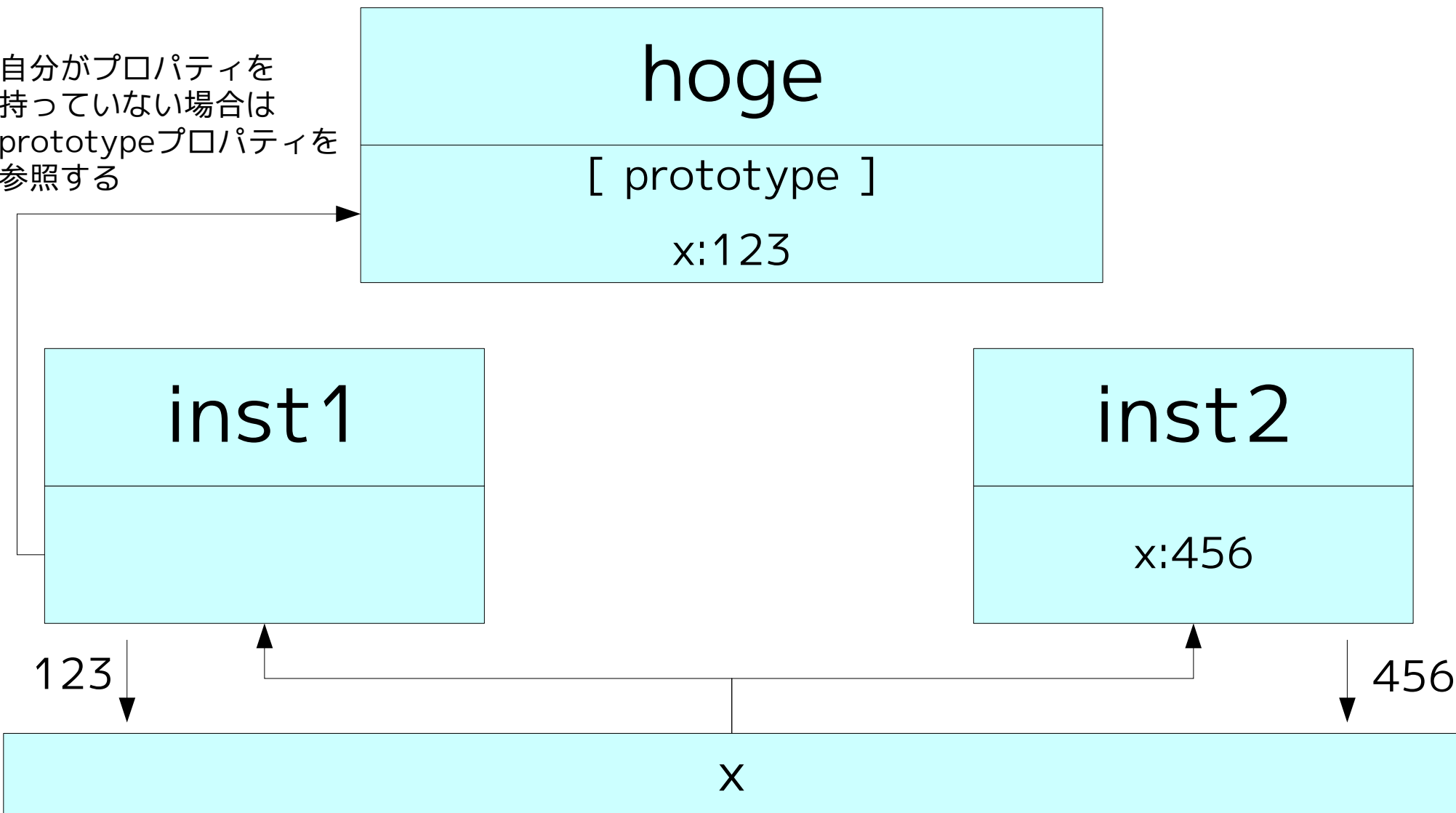
- 複数インスタンスを作成すると伝わるかも

```
function hoge(){}  
hoge.prototype.x = "123";  
inst1 = new hoge();  
inst2 = new hoge();  
inst2.x = "456";  
alert(inst1.x); // 123  
alert(inst2.x); // 456
```

- オブジェクトのプロパティに値をセットすると、そちらを使用するようになる。

# prototypeプロパティモデル

自分がプロパティを持っていない場合は prototype プロパティを参照する



# JavaScriptの オブジェクト指向の 実装

# 繼承

# 継承

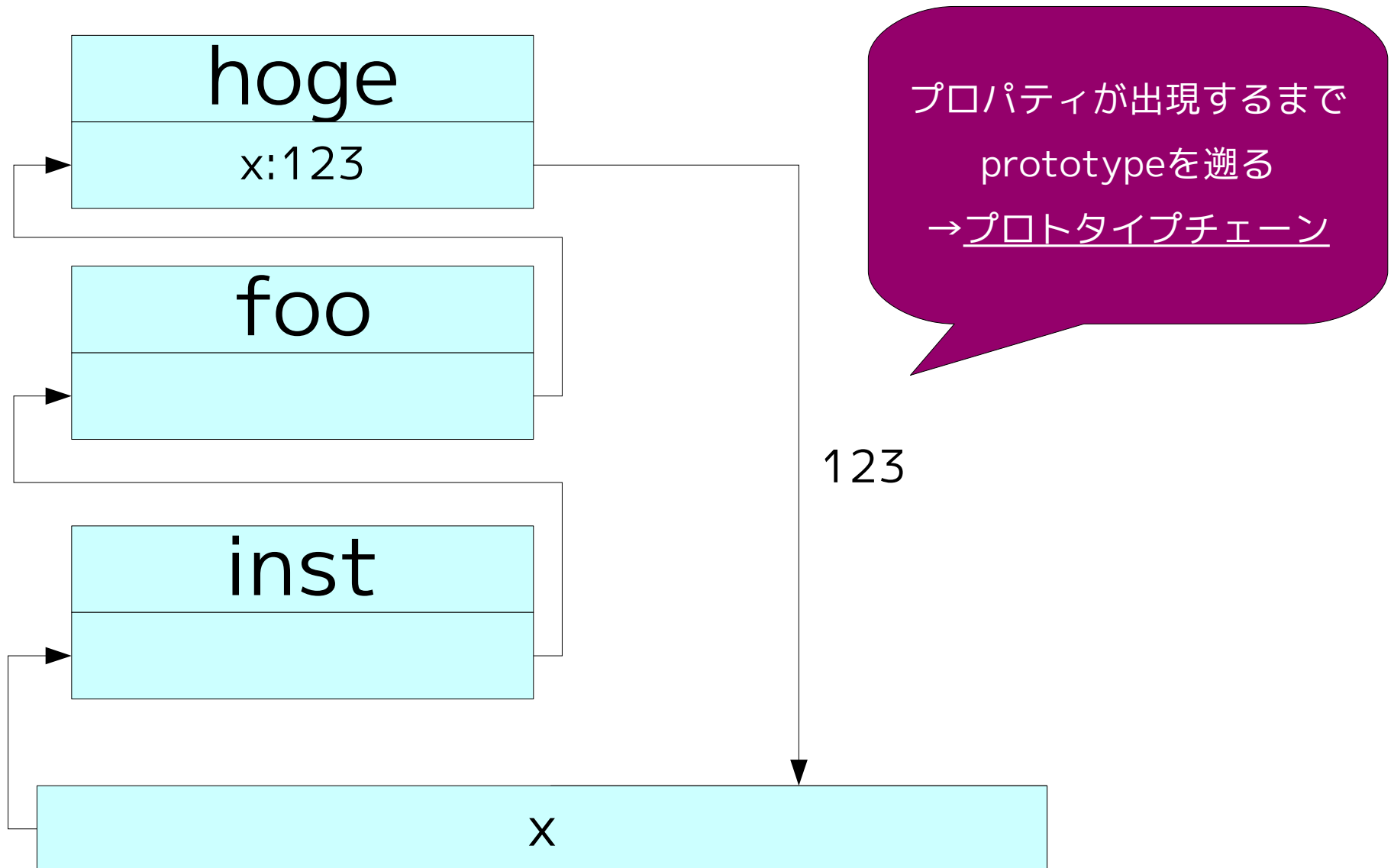
---

- prototypeプロパティにnewで代入

```
function hoge(){}  
hoge.prototype.x = "123";  
function foo(){}  
foo.prototype = new hoge();  
var inst = new foo();  
alert(inst.x); // 123
```

- instはhogeのprototypeとfooのプロトタイプの両方を参照する

# 継承時のprototypeプロパティモデル



オーバーライド

# オーバーライド

---

- 不可

(だって同じメソッド名だと上書きされちゃう)

でも方法はあるよ！

# オーバーライドの実現方法

---

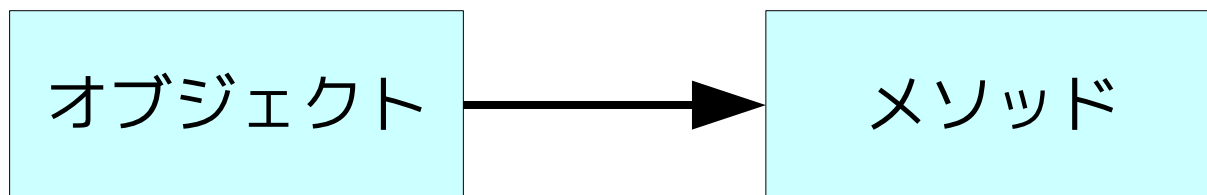
```
function hoge(){
  this.a = "A";
  this.b = "B";
}
hoge.prototype.x =
  function(){alert(this.a);};
var inst = new hoge();
inst.x =
  function(){alert(this.b);};
inst.x(); // B
hoge.prototype.x.apply(inst); // A
```

- applyは呼び出し元オブジェクトを明示的に指定できる

# 補足：applyの説明

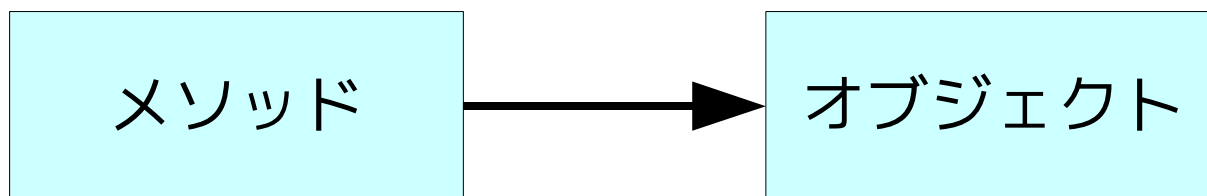
---

- 普通のメソッドの図式



オブジェクトが主体（オブジェクトがメソッドを呼ぶ）

- applyの図式



メソッドが主体（メソッドがオブジェクトを指定する）

オーバーロード

# オーバーロード

---

- 不可

(だって同じメソッド名だと以下略)

でも方法はあるよ！

# オーバーロードの実現方法

---

```
function hoge(){  
hoge.prototype.meth = function(){  
  if(arguments.length == 1){  
    this.meth1(arguments[0]);  
  }else if(arguments.length == 2){  
    this.meth2(arguments[0],arguments[1]);  
  }  
};  
hoge.prototype.meth1 = function (arg1){};  
hoge.prototype.meth2 = function (arg1,arg2){};
```

- arguments配列には引数が入っている

# プライベート 変数

# プライベート変数

---

- 不可

(全部publicなので)

でも方法はあるよ！

(でも参考程度で)

# プライベート変数の実現方法

```
function hoge(){
  var x;
  function setX(_x){
    x = _x;
  }
  function getX(){
    return x;
  }
  return {setX:setX,getX:getX};
}
var obj = hoge();
obj.setX("123");
alert(obj.getX()); //123
alert(obj.x); // undefined
```

- xはプロパティではない（関数の中のローカル変数）ので、obj.xという方法ではアクセスできない

まとめ

# まとめ

---

- メリット
  - ・ prototypeを使用することでメモリ消費量を減らせる。
  - ・ オブジェクトに機能を集約させることで  
スコープを限定できる。
  - ・ (→) 保守しやすい。
- デメリット
  - ・ 難しい

# 質疑応答

---