The Prototype Framework Part III: Better OOP

(Prototype 1.6 Version)

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/ajax.html

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6.
Developed and taught by well-known author and developer. At public venues or onsite at your location.

For live Ajax & GWT training, see training courses at http://courses.coreservlets.com/.

Taught by the author of Core Servlets and JSP, More Servlets and JSP, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
- Java 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF 1.x & 2.0, Ajax, GWT, custom mix of topics
- Ajax courses can concentrate on one library (jQuery, Prototype/Scriptaculous, Ext-JS, Dojo) or survey several
- Courses developed and taught by coreservlets.com experts (edited by Marty)
- Spring, Hibernate/JPA, EJB3, Ruby/Rails

Contact hall@coreservlets.com for details
Topics in This Section

- Constructor and prototype in one place
- Single inheritance
  - (Sort of)
- Merging objects
- Multiple inheritance
  - (Sort of)

"Object-oriented programming is an exceptionally bad idea which could only have originated in California." — Edsger Dijkstra, 1972 Turing Award winner.

Overview

- Compared to Java, JavaScript has
  - Fabulous function support
  - Lousy OOP support (not even real OOP at all)
- Prototype adds better OOP support
  - Still not real OOP, but a definite improvement
- Main methods
  - Class.create
    - Creates a constructor that calls “initialize”
    - You can define everything in prototype instead of half (fields) in constructor and half (methods) in prototype
  - Object.extend
    - Adds new capabilities to existing class
    - Lets you define object hierarchies (almost real inheritance)
    - Object.extend called automatically if first arg of Class.create is a class name
Class.create: Base Classes

- **Makes constructor that calls “initialize”**
  - You supply a *single* object that is automatically attached to MyClass.prototype

```javascript
var MyClass = Class.create({
    initialize: function(args) {
        this.field1 = blah;
        this.field2 = blah;
    },
    method1: function(...) { ... },
    method2: function(...) { ... }
});
```

- **initialize called automatically**

```javascript
var someObject = new MyClass(args);
```

Class.create & Base Classes: Circle Class

```javascript
var Circle = Class.create({
    initialize: function(radius) {
        this.radius = radius;
    },
    getArea: function() {
        return(Math.PI * this.radius * this.radius);
    }
});
```
Class.create & Base Classes: Rectangle Class

```javascript
var Rectangle = Class.create({
    initialize: function(width, height) {
        this.width = width;
        this.height = height;
    },

    getArea: function() {
        return(this.width * this.height);
    }
});
```

Class.create & Base Classes: Helper (Static) Method

```javascript
var ShapeUtils = {};

ShapeUtils.sumArea = function(shapeArray) {
    var sum = 0;
    for(var i=0; i<shapeArray.length; i++) {
        sum = sum + shapeArray[i].getArea();
    }
    return(sum);
}
```
Class.create & Base Classes: Test Case

• Code
  var shapes =
   [ new Circle(10), new Circle(20),
     new Rectangle(5,10), new Rectangle(10,20)];
  ShapeUtils.sumArea(shapes);

• Result
  1820.7963267948967

Class.Create: Single Inheritance

• Idea
  – First argument to Class.create can be a class name
  – Second argument is the class definition

  var Superclass = Class.create({
    initialize: function(...) {...},
    method1: function(...) {...}
  });

  var Subclass = Class.create(Superclass, {
    initialize: function(...) {...},
    method2: function(...) {...}
  })
Class.create: Accessing Overridden Methods

• **Problem**
  – JavaScript has no builtin approach to accessing overridden methods. So what if subclass wants to call superclass’s initialize method?

• **Solution**
  – In subclass, for any overridden method, add $super as first argument
  – $super is now the name of the overridden method
    • Not $super.methodName as in Java

• **Example**
  var Subclass = Class.create(Superclass, {
    initialize: function($super, args...) {
      $super(someOfTheArgs);
      somethingElse(restOfTheArgs);
    },
    ...
  });

Class.create & Single Inheritance: Parallelogram

// Superclass (Base class)

var Parallelogram = Class.create({
  initialize: function(length, width) {
    this.length = length;
    this.width = width;
  }
});
Class.create & Single Inheritance: Rectangle

// Subclass (extended class) of Parallelogram

var Rectangle = Class.create(Parallelogram, {
  initialize: function($super, length, width) {
    $super(length, width);
  },

  getArea: function() {
    return(this.length * this.width);
  }
});

Class.create & Single Inheritance: Test Cases

var shape1 = new Parallelogram(5, 10);
shape1.length; → 5
shape1.width; → 10
var shape2 = new Rectangle(10, 20);
shape2.length; → 10
shape2.width; → 20
shape2.getArea(); → 200
Object.extend and Multiple Inheritance

• Problem
  – Although Class.create lets you define a class, a sub-class, a sub-sub-class, a sub-sub-sub-class, etc., it only lets you specify a single immediate parent
  – So Class.create alone does not support mixin style of programming
    • Main (instantiable) base class provides core functionality
    • Mixin class (usually not instantiable; static methods only in JavaScript) provides additional functionality

• Solution
  – Specify base class in Class.create
  – Use Object.extend(this, MixinClass) in constructor

• Note
  – Java does not support multiple inheritance at all
  – Interfaces are not the same as mixin classes, since interfaces have no real (implemented) methods

Object.extend

• Idea
  – Merges two objects: first now has all properties of second

• Simple usage (object merge)
  – Add properties to a single object
    • var obj1 = {a: 1, b: 2}
    • var obj2 = {c: 3, d: 4};
    • Object.extend(obj1, obj2); // obj1 has a, b, c, d

• Advanced usage (multiple inheritance)
  – In initialize, extend "this" with new class
    • Object.extend(this, MixinClass);

• Note for testing
  – Prototype provides Object.keys that returns array of all of the property names of an object
Object.extend:
Simple Object Merging

```javascript
var obj1 = { a: 1, b: 2};
Object.keys(obj1); ➞ ["a", "b"]
var obj2 = { c: 3, d: 4};
Object.keys(obj2); ➞ ["c", "d"]
Object.extend(obj1, obj2);
Object.keys(obj1); ➞ ["a", "b", "c", "d"]
Object.keys(obj2); ➞ ["c", "d"]
```

Multiple Inheritance: Mixin Class

```javascript
// Mixin class: static methods only.
// Can't call new Printable(...).

var Printable = {};

Printable.printInfo = function() {
    return("Area is " + this.getArea() + "", color is " + this.color);
};
```
Multiple Inheritance: Subclass with Mixin

// Subclass (extended class) of Rectangle, // also supports Printable mixin class

var ColoredRectangle = Class.create(Rectangle, {
    initialize: function($super, length, width, color) {
        $super(length, width);
        this.color = color;
        Object.extend(this, Printable);
    }
});

Multiple Inheritance: Test Cases

var shape3 = new ColoredRectangle(2, 4, "blue");
shape3.length; ➞ 2
shape3.width; ➞ 4
shape3.getArea(); ➞ 8
shape3.color; ➞ "blue"
shape3.printInfo(); ➞ "Area is 8, color is blue"
Summary

- **Base classes**
  - `Class.create({initialize: ..., otherMethod: ...});`

- **Single inheritance**
  - `Class.create(Superclass, {initialize: ..., otherMethod: ...});`
  - Use `$super` to get at overridden methods (esp. initialize)

- **Merging simple objects**
  - `Object.extend(obj1, obj2);`
    - Adds to obj1; leaves obj2 unchanged

- **Multiple inheritance**
  - Call `Object.extend(this, MixinClass)` from constructor

Questions?

Customized Java EE Training: [http://courses.coreservlets.com/](http://courses.coreservlets.com/)
Servlets, JSP, JSF 2.0, Struts, Ajax, GWT 2.0, Spring, Hibernate, SOAP & RESTful Web Services, Java 6. Developed and taught by well-known author and developer. At public venues or onsite at your location.