Telerik Academy

Inheritance with Classical OOP

In JavaScript

Telerik Software Academy

Learning & Development

http://academy.telerik.com

Telerik Academy

Table of Contents

- Implementing classical inheritance
- The prototype chain
- Using parent methods
- OOP frameworks
  - John Resig simple inheritance

Telerik Academy

Inheritance in Classical OOP

Like in C#, Java or C++

Telerik Academy

Inheritance in Classical OOP

- Inheritance is a way to extend the functionality of an object, into another object
  - Like Student inherits Person
  - Person inherits Mammal, etc…
- In JavaScript inheritance is achieved by setting the prototype of the derived type to an instance of the super type

```
function Person(fname, lname) {}
function Student(fname, lname, grade) {}
Student.prototype = new Person();
```

  - Now all instances of type Student are also of type Person and have Person functionality

```
var student = new Student("Kiro", "Troikata", 7);
```

Telerik Academy

Inheritance in Classical OOP

Live Demo

Telerik Academy

The Prototype Chain

The way to search properties in JavaScript

The Prototype Chain

- Objects in JavaScript can have only a single prototype
  - Their prototype also has a prototype, etc…
  - This is called the prototype chain
- When a property is called on an object
  1. This object is searched for the property
  2. If the object does not contain such property, its prototype is checked for the property, etc…
  3. If a null prototype is reached, the result is undefined

Telerik Academy

Calling Parent Methods

Use some of the power of OOP

Telerik Academy

Calling Parent Methods

- JavaScript has no direct way of calling its parent methods
  - Function constructors actually does not who or what is their parent
- Calling parent methods is done using call and apply

Telerik Academy

Calling Parent Methods: Example

- Having Shape:

```
var Shape = (function () {
  function Shape(x, y) {
    //initialize the shape
  }
  Shape.prototype = {
    serialize: function () {
      //serialize the shape
      //return the serialized
    }
  };
  return Shape;
}());
```

- Inheriting it with Rect

```
var Rect = (function () {
 function Rect(x, y,
               width, height) {
   Shape.call(this, x, y);
   //init the Rect
 }
 Rect.prototype = new Shape();
 Rect.prototype.serialize=function (){
   Shape.prototype
        .serialize
        .call(this);
   //add Rect specific serialization
   //return the serialized;
 };
 return Rect;
}());
```

Telerik Academy

Calling Parent Methods: Example

- Having Shape:

```
var Shape = (function () {
  function Shape(x, y) {
    //initialize the shape
  }

  Shape.prototype = {
    serialize: function () {
      //serialize the shape
      //return the serialized
    }
  };

  return Shape;
}());
```

- Inheriting it with Rect

```
var Rect = (function () {
 function Rect(x, y,
               width, height) {
   Shape.call(this, x, y);
   //init the
 }
 Rect.proto                    );
 Rect.prototype.serialize=function (){
   Shape.prototype
        .serialize
        .call(this);
   //add Rect specific serialization
   //return the serialized;
 };
 return Rect;
}());
```

Call parent constructor

Telerik Academy

Calling Parent Methods: Example

- Having Shape:

```
var Shape = (function () {
  function Shape(x, y) {
    //initialize the shape
  }

  Shape.prototype = {
    serialize: function () {
      //serialize the shape
      //return the serialized
    }
  };

  return Shape;
}());
```

- Inheriting it with Rect

```
var Rect = (function () {
 function Rect(x, y,
               width, height) {
   Shape.call(this, x, y);
   //init the
 }
 Rect.proto                    ;
 Rect.prototype.serialize=function (){
   Shape.prototype
        .serialize
        .call(this);
   //add Rect speci
   //return the ser
 };

 return Rect;
}());
```

Call parent constructor

Call parent method

Telerik Academy

Calling Parent Methods

Live Demo

Telerik Academy

OOP Frameworks

Telerik Academy

OOP Frameworks

- OOP is a primary design paradigm in most programming languages
  - Yet, OOP in JavaScript is not that perfect
- And that is why every framework has its own way of doing OOP
  - YUI, Prototype.js, Backbone.js, etc…
- If none of these frameworks is used, a simple implementation by John Resig is intruded

Telerik Academy

John Resig Simple Inheritance

- How to use it?
  - Copy the code from: http://tinyurl.com/simple-inheritance
  - Create "class" with:

```
var Shape = Class.extend({
  init: function(x, y){
    this._x = x;
    this._y = y;
  },
  serialize: function(){
    return {
      x: this._x,
      y: this._y
    };
  }
});
```

  - Inherit it with:

```
var Rect = Shape.extend({
  init: function(x, y, w, h){
    this._super(x, y);
    this._width = w;
    this._height = h;
  },
  serialize: function(){
    var res = this._super();
    res.width = this._width;
    res.height = this._height;
    return res;
  }
});
```

Telerik Academy

John Resig Simple Inheritance

- How to use it?
  - Copy the code from: http://tinyurl.com/simple-inheritance
  - Create "class" with:

```
var Shape = Class.extend({
  init: function(x, y){
    this._x = x;
    this._y = y;
  },
  serialize: function(){
    return {
      x: this._x,
      y: this._y
    };
  }
});
```

Constructor

  - Inherit it with:

```
var Rect = Shape.extend({
  init: function(x, y, w, h){
    this._super(x, y);
    this._width = w;
    this._height = h;
  },
  serialize: function(){
    var res = this._super();
    res.width = this._width;
    res.height = this._height;
    return res;
  }
});
```

Telerik Academy

John Resig Simple Inheritance

- How to use it?
  - Copy the code from: http://tinyurl.com/simple-inheritance
  - Create "class" with:

```
var Shape = Class.extend({
  init: function(x, y){
    this._x = x;
    this._y = y;
  },
  serialize: function(){
    return {
      x: this._x,
      y: this._y
    };
  }
});
```

Constructor

Method

  - Inherit it with:

```
var Rect = Shape.extend({
  init: function(x, y, w, h){
    this._super(x, y);
    this._width = w;
    this._height = h;
  },
  serialize: function(){
    var res = this._super();
    res.width = this._width;
    res.height = this._height;
    return res;
  }
});
```

Telerik Academy

John Resig Simple Inheritance

- How to use it?
  - Copy the code from: http://tinyurl.com/simple-inheritance
  - Create "class" with:

```
var Shape = Class.extend({
  init: function(x, y){
    this._x = x;
    this._y = y;
  },
  serialize: function(){
    return {
      x: this._x,
      y: this._y
    };
  }
});
```

Constructor

Method

  - Inherit it with:

```
var Rect = Shape.extend({
  init: function(x, y, w, h){
    this._super(x, y);
    this._width = w;
    this._height = h;
  },
  serialize: function(){
    var res = this._super();
    res.width = this._width;
    res.height = this._height;
    return res;
  }
});
```

Parent constructor

Telerik Academy

John Resig Simple Inheritance

- How to use it?
  - Copy the code from: http://tinyurl.com/simple-inheritance
  - Create "class" with:

```
var Shape = Class.extend({
  init: function(x, y){
    this._x = x;
    this._y = y;
  },
  serialize: function(){
    return {
      x: this._x,
      y: this._y
    };
  }
});
```

Constructor

Method

  - Inherit it with:

```
var Rect = Shape.extend({
  init: function(x, y, w, h){
    this._super(x, y);
    this._width = w;
    this._height = h;
  },
  serialize: function(){
    var res = this._super();
    res.width = this._width;
    res.height = this._height;
    return res;
  }
});
```

Parent constructor

Parent serialize method

Telerik Academy

John Resig Simple Inheritance

Live Demo

Telerik Academy

Inheritance in Classical OOP