"Object Oriented" JavaScript

© 2010 RoCk SOLiD KnOwledge Ltd.

Objectives

- How to define objects in JavaScript
- Prototypes
- Inheritance
- Instantiation
- Sub-classing

© 2010 RoCk SOLiD KnOwledge Ltd.

Prototypes

- JavaScript supports prototypical inheritance
  - Each JavaScript functions have a 'prototype' property
  - Shared across instances
  - Can use to share functions

© 2010 RoCk SOLiD KnOwledge Ltd.

Constructor functions

- Can create objects many ways in JavaScript
  - Can use the object syntax
    - var k = {name: 'Kevin'};
  - Can use the constructor syntax

```
function User(){
};

var user = new User();
```

© 2010 RoCk SOLiD KnOwledge Ltd.

Issues with instances

- **Creating instances creates multiple copies**
  - **Each instance gets its own copy of functions**

```
function User(){
    this.name = function(){...}
};

var kevin = new User();
kevin.name("Kevin");
var terry = new User();
terry.name("Terry");
```

© 2010 RoCk SOLiD KnOwledge Ltd.

Sharing with prototype

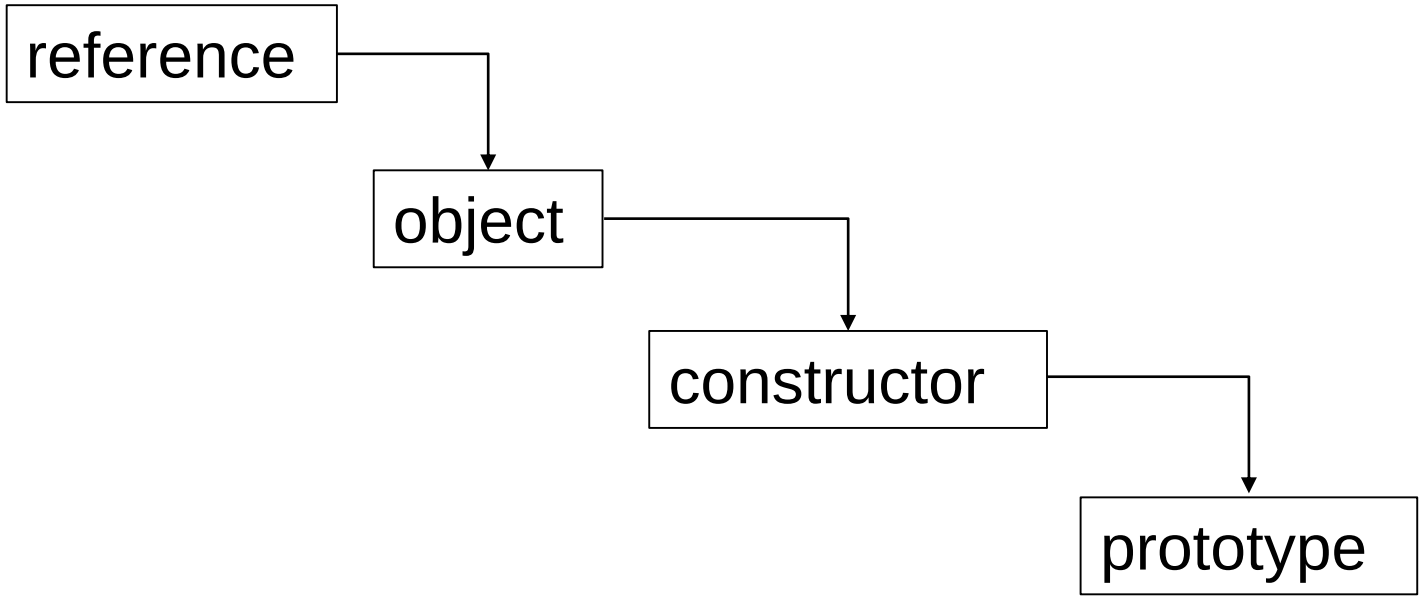
- **Prototype is shared across instances**

```
function User(){
};

User.prototype.name = function(){
};

var kevin = new User();
kevin.name("Kevin");
var terry = new User();
terry.name("Terry");
```

© 2010 RoCk SOLiD KnOwledge Ltd.

Resolving the Prototype

- Prototype properties are not copied into instance
  - Sort of defeats the purpose
  - Each object has a constructor property
  - This references a prototype property



© 2010 RoCk SOLiD KnOwledge Ltd.

Chrome Debugger

Elements Resources Network Sources Timeline Profiles Audits Console

```
> kevin
  ▼User {name: function, fullName: function}
    ▶name: function (name){
    ▼__proto__: User
      ▼constructor: function User(){
          arguments: null
          caller: null
          length: 0
          name: "User"
        ▼prototype: User
          ▶constructor: function User(){
          ▶fullName: function (firstName, lastName){
          ▶__proto__: Object
        ▶__proto__: function Empty() {}
        ▶<function scope>
      ▶fullName: function (firstName, lastName){
      ▶__proto__: Object
>
```

Elements Resources Network Sources Timeline Profiles Audits Console

```
> kevin.constructor
  function User(){
      this.name = function(name){
          return name;
      }
  }
> kevin.constructor.prototype
  ▶User {fullName: function}
> kevin.constructor.prototype.fullName
  function (firstName, lastName){
      return firstName + " " + lastName;
  }
>
```

© 2010 RoCk SOLiD KnOwledge Ltd.

Live updates

- Prototype is 'live'
  - Can attach to the prototype after object construction

© 2010 RoCk SOLiD KnOwledge Ltd.

Order of preference

- **Object instance looked at before prototype**

```
function User(){
};

User.prototype.name = function(){
};

var kevin = new User();
kevin.name = function(){// use this one}
```

© 2010 RoCk SOLiD KnOwledge Ltd.

Prototypes for identity

- Can use constructor to identify type

```
test("identity", function(){
    var kevin = new User();

    assert(kevin instanceof User);
    assert(kevin.constructor === User);
});
```

© 2010 RoCk SOLiD KnOwledge Ltd.

Inheritance

- **Prototypes also enable inheritance**
  - **Via the prototype chain**
  - **Can use instance of one object as prototype of another**

Elements Resources Network Sources Timeline Profiles Audits Console

```
> kevin
  ▼User {name: function, age: function, constructor: function, fullName: function}
    ▶name: function (name) {
    ▼__proto__: Person
      ▶age: function (){
      ▼constructor: function User() {
          arguments: null
          caller: null
          length: 0
          name: "User"
        ▶prototype: Person
        ▶__proto__: function Empty() {}
        ▶<function scope>
      ▶fullName: function (firstName, lastName) {
      ▶__proto__: Person
```

© 2010 RoCk SOLiD KnOwledge Ltd.

Inheritance

- Set 'derived' prototype as 'base class'
  - Sometimes see User.prototype = new Person()
  - Now all derived instances share single base's properties

```
function Dummy () {}
function Person() {
}
Person.prototype.age = function (){...}
function User() {
    this.name = function (name) {...}
}
Dummy.prototype = Person.prototype;
User.prototype = new Dummy();
// otherwise User's ctor == Person
User.prototype.constructor = User;
```

© 2010 RoCk SOLiD KnOwledge Ltd.

Base class constructor

- Call the base class through the constructor function
  - remember constructors are functions

```
function Dummy () {}
function Person(name) {
    this.name = name;
}

function User(name, age) {
    Person.call(this, name);
    this.age = age;
}
```

© 2010 RoCk SOLiD KnOwledge Ltd.

Calling super methods

- Some libraries allow you to set up calling chains
  - e.g. Crockford and John Resig
- Considered not to be particularly worthwhile

© 2010 RoCk SOLiD KnOwledge Ltd.

Summary

- JavaScript supports prototypical 'inheritance'
- Add functions to the prototype
- Set the prototype as another class
- Can take this further, calling superclass etc.
  - See Crockford, Resig and others

© 2010 RoCk SOLiD KnOwledge Ltd.