Prototypal Inheritance

Ambrose Bonnaire-Sergeant
Some things are more convenient with computers...
Your chair...
I want one like that!
...can I’ve a copy?
Your chair...
You want to attach rockers...
You want to attach rockers...
You want to attach rockers…

but *keep the original!*
Your chair...
Your chair’s model is defective.
Your chair’s model is defective
Your chair’s model is defective
Fix each one?
Fix each one?
...or fix them all at once!
Physical objects don’t work like that!
Computer representations of objects are more flexible
This talk: 

Prototypal Objects
What is a prototype?

proto·type
noun

1. a first, typical or preliminary model of something, especially a machine, from which other forms are developed or copied.
...can I’ve a copy?
Copy

Prototype
Inheritance
Notation

`Object.create()`
Notation

= Object.create( )
Notation

\[ \text{Object.create}(\text{<prototype>}) \]

create a new object with given prototype
You want to attach rockers…
but *keep the original!*
Change
Change
<table>
<thead>
<tr>
<th>Commands</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>![Chair Image]</td>
</tr>
<tr>
<td>Commands</td>
<td>Result</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td><code>= Object.create();</code></td>
<td><img src="image" alt="Chair" /></td>
</tr>
</tbody>
</table>
= Object.create();

Commands

Result
= Object.create();

.rockers = ;
Commands

= Object.create();

.rockers = ;

Result
<table>
<thead>
<tr>
<th>Commands</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>= Object.create()</code></td>
<td></td>
</tr>
<tr>
<td><code>.rockers =</code></td>
<td></td>
</tr>
<tr>
<td><code>.redPillow =</code></td>
<td></td>
</tr>
</tbody>
</table>
```javascript
= Object.create();

.rockers = ;

.redPillow = ;
```
...fix them all at once!
Shared State
Share
<table>
<thead>
<tr>
<th>Commands</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image.png" alt="Chair" /></td>
</tr>
</tbody>
</table>
= Object.create();
Commands

= Object.create();

= Object.create();

Result
### Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>= Object.create();</td>
<td></td>
</tr>
<tr>
<td>= Object.create();</td>
<td></td>
</tr>
<tr>
<td>.redPillow =</td>
<td></td>
</tr>
</tbody>
</table>
Commands

= Object.create();

= Object.create();

.redPillow = Pillow;

.rockers = Rocker;
Objects are prototypes
Objects are prototypes
Objects = prototypes
Chain
Chain
Differential Inheritance
<table>
<thead>
<tr>
<th>Commands</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image" alt="Chair" /></td>
</tr>
<tr>
<td>Commands</td>
<td>Result</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td><code>= Object.create();</code></td>
<td><img src="image" alt="Result" /></td>
</tr>
</tbody>
</table>
= Object.create();

.redPillow = ;
= Object.create();

.redPillow = ;

= Object.create();
Commands

= Object.create();

.redPillow = ;

= Object.create();

.yellowPillow = ;
Objects pass messages
To:

Message

Re:
<table>
<thead>
<tr>
<th>Commands</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image-url" alt="Image of a chair and a mailbox" /></td>
</tr>
</tbody>
</table>
.redPillow = pillow;
$.redPillow = ;

$.redPillow
.redPillow =

.redPillow

 Result

.redPillow
Commands

.redPillow = pillow;

.redPillow

Result

.redPillow

.mailbox

.envelope
Objects delegate to prototypes
has proto
Delegation

To:

has proto
Delegation
Delegation

To:

Re:

has proto

Delegation
<table>
<thead>
<tr>
<th>Commands</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="image_url" alt="Image" /></td>
</tr>
</tbody>
</table>
= Object.create();
```javascript
yellowPillow = Object.create({
  redPillow: Object.create(null,
    {
      value: null,
      writable: true,
      enumerable: true,
      configurable: false
    }
  ),
  yellowPillow: null,
});
```
Commands

```
.yellowPillow = Object.create();
.redPillow
.yellowPillow = ;
.redPillow
```

Result
Commands

```javascript
redPillow = Object.create(null);

yellowPillow = Object.create(null);
```

Result
Commands

```
.yellowPillow = Object.create();
.yellowPillow = ;
.redPillow
```
```javascript
.yellowPillow = Object.create({});
.redPillow
.yellowPillow = ;
.redPillow
```
Commands

```javascript
yellowPillow = Object.create();
.redPillow
.yellowPillow = ;
.redPillow
```

Result

```javascript
.redPillow
```
Commands

```javascript
.yellowPillow = Object.create();
.redPillow = Object.create();
.redPillow
```

Result

```javascript
.yellowPillow = Object.create();
.redPillow = Object.create();
.redPillow
```
yellowPillow = Object.create();
.redPillow
.yellowPillow = ;
.redPillow

.redPillow
Objects have behavior
pillowCalc() =
pillowCalc() = Pillow count?
Behavior

To:

Pillow count?

pillowCalc() =

pillow

[Diagram of a chair with pillows and a mailbox, arrows indicating the calculation process]
Behavior

To: Pillow

Re: 1

pillowCalc() = 1

Pillow count?
Behavior

To:
Pillow

pillowCalc() = 1
Prototypes serve descendants
has proto

pillowCalc() =
pillowCalc() = has proto

To:
Pillow count?
pillowCalc() =

To: Pillow

Pillow count?

has proto
has proto

pillowCalc() =

To: Pillow
count?
pillowCalc() =?

To: Pillow count?
pillowCalc() =

Service

has proto

To:

Pillow

count?
The essence of prototypal objects:
Objects are prototypes
Objects are prototypes
$Objects = prototypes$
Objects = prototypes
Prototypes share state with children
Prototypes share state with children
Prototypes share state with children
Objects communicate via messages
Objects communicate via messages
Prototypes serve children
Prototypes serve children
JavaScript is prototypes at the bottom
JavaScript
JavaScript

Prototypes
JavaScript

Prototypes

Object.create()

obj.__proto__

function(){this}
Prototypes

JavaScript

Constructor pattern

uses

Object.create()

obj.__proto__

function(){this}
JavaScript

**Constructor pattern**
- `new C()`
- `instanceof`
- `C.prototype`
- `P.constructor`

**Prototypes**
- `Object.create()`
- `obj.__proto__`
- `function(){this}`
JavaScript

ES6 Classes

Constructor pattern

new C() instanceof C.prototype
P.constructor

Prototypes

Object.create()
obj.__proto__
function(){this}
JavaScript

ES6 Classes

```
class C {
    ...
}
```

Constructor pattern

```
new C() instanceof C.prototype
P.constructor
```

Prototypes

```
Object.create()
obj.__proto__
function(){this}
```
JavaScript

ES6 Classes

```
class C { ... }
```

Constructor pattern

```
new C() instanceof C.prototype
P.constructor
```

Prototypes

```
Object.create() (this)
obj.__proto__
function() {this}
```

This talk

uses

uses

uses
JavaScript

ES6 Classes

Constructor pattern

Prototypes

Thanks!

ambrosebs.com