Node.js ABC's - I is for Inheritance

Joe Pruitt, 2015-26-06

The ability to create a class and then extend it is the basis for object-oriented programming. Inheritance in object-oriented programming is when an object, or class, is derived from another object using the same core implementation.

Inheritance in JavaScript is a little different than other languages as there is no explicit class keyword and a mechanism for specify a derived relationship. In JavaScript classes are all declared as functions as in this example

```
function MyStuff(myThings) {
  this.things = myThings;
}
```

In the above example, the function MyStuff is a way to hold onto a bunch of things. the things property is used to store the value of myThings which is passed in through the constructor.

Prototypes

Since there is no explict "class" keyword to declare class and inheritance, all objects (functions) in JavaScript have a special "prototype" property, which is used to add methods and properties available to instances of that function.

If one wanted to add on a method to print the contents of the MyStuff object, you could do so like this

```
MyStuff.prototype.print = function() {
  console.log(this.things);
}
```

Now you can create a new object and access the print method like this

```
var coolStuff = new MyStuff("Lots of cool things here");
coolStuff.print();
```

The above showed the use of the prototype "property". The second aspect of prototypes in JavaScript is the prototype "attribute". The prototype attribute, also commonly referred to as the "prototype object", points to the objects "parent" that the object inherited it's properties and methods from. The prototype attribute is set automatically when you create a new object.

In the above example, the coolStuff's prototype attribute is MyStuff.prototype. All objects created with the Object constructor, inherit from Object.prototype.

Inheritance

All JavaScript objects inherit the methods and properties of their prototypes. In the following example, I'll create a Pet object and then a Dog object that derives from Pet by using the parent constructor.

```
var Pet = function() {
}
// Add a "feed" method to Pet.prototype
Pet.prototype.feed = function() {
  console.log("The pet has been fed.");
}
// Define a Dog constructor
```

```
function Dog() {
  // Call the parent constructor, making sure that "this" is set correctly
  Pet.call(this);
}
Dog.prototype = Object.create(Pet.prototype);
Dog.prototype.constructor = Pet;
// Add a sleep method
Dog.prototype.sleep = function() {
  console.log("The dog is sleeping...");
}
var pet = new Pet();
var dog = new Dog();
// Call Pet's feed method
pet.feed();
// Call Dog's feed method
dog.feed();
// Call Dog's sleep method
dog.sleep();
console.log(dog instanceof Pet);

$ node inheritance.js
The pet has been fed.
The pet has been fed.
The dog is sleeping...
true
```

Since every object has a prototype, one can chain objects upon objects to create a hierarchy chain.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.
Corporate Headquarters
info@f5.com

F5 Networks
Asia-Pacific
apacinfo@f5.com

F5 Networks Ltd.
Europe/Middle-East/Africa
emeainfo@f5.com

F5 Networks
Japan K.K.
f5j-info@f5.com