ES6 PPT1511241: class 类

Thursday, November 5, 2015 10:15 AM

类 class：

1. 类 class 的基本语法：class 声明的依然是 function 类型，是 function 声明的另一种写法

   写法 function：

```
function Car(color) {
  this.color = color;
}
Car.prototype.maker = function () { return 'function factory'; }
Car.prototype.toString = function () { return 'I am ' + this.color + ' Car created by ' +
this.maker(); }
console.log((new Car('BLUE')).toString()); // -> I am BLUE Car created by function factory
```

   写法 class：

```
class Car { // Car.__proto__ === Function.prototype && Car.prototype.__proto__ ===
Object.prototype
  constructor(color) { // class构造函数
    this.color = color; // class实例属性
  }
  maker() { return 'class factory'; } // class prototype方法
  toString() { return 'I am ' + this.color + ' Car created by ' + this.maker(); }
} // typeof Car === "function"
console.log((new Car('RED')).toString()); // -> I am RED Car created by class factory
```

   注意：constructor()为空方法时可不显式声明，方法声明在prototype上，方法声明不使用;结尾，class声明无变量提升，不能以非new方式直接调用Car()，Car.prototype.constructor === Car

2. 类 class 的继承语法：

   写法 function - prototype：

```
function BMW(color, model) {
  Car.call(this, color);
  this.model = model;
}
BMW.prototype = new Car();
BMW.prototype.getModel = function () { return this.model; }
BMW.prototype.toString = function () {
  return 'I am ' + this.color + ' BMW ' + this.getModel() + ' created by ' + this.maker();
}
console.log((new BMW('BLUE', 'M3')).toString()); // -> I am BLUE BMW M3 created by
function factory
```

   写法 class - extends：

```
class BMW extends Car { // BMW.__proto__ === Car && BMW.prototype.__proto__ ===
Car.prototype
  constructor(color, model) {
    super(color);
    this.model = model; // 子类实例属性
    }
    getModel() { return this.model; } // 子类prototype方法
    toString() { // 覆盖父类prototype方法
      return 'I am ' + this.color + ' BMW ' + this.getModel() + ' created by ' + this.maker();
    }
  }
  console.log((new BMW('RED', 'M5')).toString()); // -> I am RED BMW M5 created by class
factory
```

继承解析：

```
// 1. 继承null：无原型继承
class Base extends null { }
Base.__proto__ === Function.prototype // true
Base.prototype.__proto__ === undefined // true
// 2. 直接定义类：原型继承自Object.prototype
class Parent { }
Parent.__proto__ === Function.prototype // true
Parent.prototype.__proto__ === Object.prototype // true
// 3. 继承类：原型继承自父类prototype
class Child extends Parent { }
Child.__proto__ === Parent // true
Child.prototype.__proto__ === Parent.prototype // true
```

3. 获取父类：Object.getPrototypeOf(BMW) === Car
4. 使用super关键字：super 代表父类实例

```
class BMWUSA extends BMW {
  maker() { return 'USA ' + super.maker(); } // super获取父类实例方法
  toString() {
    return 'I am from ' + this.maker() + ' - a child of ' + super.maker();
  }
}
console.log((new BMWUSA('RED', 'M5')).toString()); // -> I am from USA class factory - a child of
class factory
```

5. 继承原生构造函数：ES6 即将支持继承自 Boolean, Number, String, Array, Date, Function, RegExp, Error, Object
6. 存值 setter 与取值 getter 函数：使用 set / get 关键字前缀于成员方法

```
class BMWChina extends BMW {
  get model() { return '中德合资: ' + this._model; }
  set model(value) { this._model = '华晨宝马-' + value; console.log(this._model); }
}
let myBMWChina = new BMWChina();
myBMWChina.model = '520'; // -> 华晨宝马-520
console.log(myBMWChina.model) // -> 中德合资: 华晨宝马-520
```

7. 静态方法：使用 static 关键字前缀于成员方法，子类可继承之

```
class BMWEuro extends BMW {
  static logo() { return 'BMW Europe';}
}
class BMWGermany extends BMWEuro {
}
```

```
 console.log('Logo for BMWEuro: ' + BMWEuro.logo() + ', Logo for BMWGermany: ' +
BMWGermany.logo());
```

8. 静态属性：ES6 下只能写在 class 结构体外面，ES7 提案可使用 static 前缀于变量声明

```
 class BMWUK extends BMWEuro {
 }
 BMWUK.driveOn = 'LEFT'; // ES6语法定义静态属性
 console.log('Logo for BMWUK: ' + BMWUK.logo() + ', drive on ' + BMWUK.driveOn + ' side');
 class BMWFrance extends BMWEuro {
  static driveOn = 'RIGHT'; // ES7提案语法定义静态属性
 }
 console.log('Logo for BMWFrance: ' + BMWFrance.logo() + ', drive on ' + BMWFrance.driveOn + '
side');
```

9. 使用 new.target 确定构造函数如何调用：ES6 即将支持

```
// 以下 new.target 返回 Person
function Person(name) {
 if (new.target !== undefined) { this.name = name; }
 else { throw new Error('必须使用new生成实例'); }
}
// 以下 new.target 返回 Square
class Rectangle {
 constructor(length, width) { console.log(new.target === Rectangle); }
}
class Square extends Rectangle {
 constructor(length) { super(length, length); }
}
var obj = new Square(3); // false
```

各类应用：

1. 在子类实例中更改父类实例行为：

```
myBMWUSA.__proto__.proto__.toString = function() { 'I have a child BMWUSA' }
myBMW.toString();
```

2. 定义类继承自原生构造函数以其扩充功能：

```
class CustomNumber extends Number { abs() { return this < 0 ? -this : this; } } (new
CustomNumber(-43)).abs();
```

3. 使用 new.target 定义虚类，规定必须有子类继承

附录：JS原型链关系

```
Function.constructor === Function;
Function.__proto__ === Function.prototype;
(function(){}).__proto__ === Function.prototype;
Object.constructor === Function;
Object.__proto__ === Function.prototype;
({}).__proto__ === Object.prototype;
Function.prototype.constructor === Function;
Function.prototype.__proto__ === Object.prototype;
Function.prototype.prototype === undefined;
Object.prototype.constructor === Object;
Object.prototype.__proto__ === null;
```