爱创课堂前端培训

贪吃蛇

班级：爱创课堂十九期

讲师：彭帅伟

日期：2018年12月21日

# 复习：

构造函数与普通函数之间的区别：

定义方式： 没有区别，构造函数的首字母要大写，此要求非语法要求

目的：

普通函数： 实现某一功能

构造函数： 创建对象

使用方式：

普通函数： 直接调用

构造函数： 使用new 关键字来调用

原型：

每一个函数都有一个属性是prototype，它的值是一个对象

特点：原型上面的所有内容都可以被实例化对象所访问

作用：共享方法

继承：

类式继承：

将子类的原型指向父类的实例

构造函数式继承：

在子类的构造函数中， 执行父类的构造函数并使用apply改变this指向，然后将arguments传递

组合式继承:

类式继承+构造函数式继承

寄生式继承：

定义一个寄生函数，接受连个参数，分别是父类和子类， 在寄生函数中定义一个新的构造函数，将该构造函数的原型指向父类的原型，将该构造函数的实例交给子类， 补回子类的constructor属性，返回子类

寄生组合式继承：

在定义类的时候，使用构造函数式继承

安全类：

无论外部如何调用，都是返回该类的实例化对象

# 内置构造函数

## 内置构造函数的分类

ECMAScript核心语法自带的一些内置构造函数：

Function、Object、 Array、 String、 Number、 Boolean、 RegExp、 Date、 Error

## Function

这是一个构造函数，用于创建函数

使用方式：

接受任意个字符串参数， 除了最后一个参数， 其它的都是形参

举例：

|  |
| --- |
| 1. var fun = new Function("a", "b", "return a + b"); 2. var result = fun(1, 2); |

输出：

|  |
| --- |
|  |

需要注意的是， 如果使用构造函数来创建函数，通过函数.name得到的是anonymous，

但是通过函数声明式或者是函数表达式，函数.name得到的就是函数的名称

|  |
| --- |
| 1. 函数.length 的到函数形参的个数 2. arguments.length 是实参的个数 |

## RegExp

这是一个构造函数，作用用于创建正则表达式

使用方式：

接受两个 参数

第一个参数: 字符串，定义正则表达式的表达体

第二个参数：字符串，定义正则表达式的修饰符， i， g , m

举例：

|  |
| --- |
| 1. var reg = /^\s+$/g; 2. var reg1 = new RegExp("\\s+", "g"); 3. console.log(reg); 4. console.log(reg1); |

## String

该构造函数是string值类型的对应包装类型

举例：

|  |
| --- |
| 1. // 是string值类型的对应包装类型 2. var str = "abcdefg"; 3. var str\_obj = new String(str); |

这也就是， 为什么字符串可以调用方法的原因

## Number

该构造函数是number值类型的对应包装类型

举例：

|  |
| --- |
| 1. var num = "3a"; 2. var num\_obj = new Number(num); 3. Number本身不是一个安全类， 当Number自执行的时候，会作为类型转换函数来使用 4. var num1 = Number(num); |

## Boolean

该构造函数是bool值类型的对应包装类型

举例：

|  |
| --- |
| 1. var f = false; 2. var f\_obj = new Boolean(f); 3. // !f true !f\_obj fasle |

# 贪吃蛇游戏:

我们决定使用面向对象的方法来书写游戏

游戏是最适合使用面向对象的方式的

我们把整个游戏看成是一个类

既然是游戏类，可以拥有各种各样的属性和方法

有地图属性

有蛇属性

有食物属性

还有障碍物属性

我们把地图看成一个类

有行属性

有列属性

有宽度属性（总宽）

有高度属性（总高）

有一个数组属性，是一个二维数组， 里面就是存放每一个小方格元素

我们把蛇也看成是一个类

有数组属性，数组中存放的是蛇的每一节身体

有方向属性，

有生成方法

有转向方法

把食物看成是一个类

有x属性

有Y属性

有Img属性

把障碍物看成是一个类

有数组属性

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |