爱创课堂前端培训

# 设计模式第一天

第1天课堂笔记（本课程共3天）

班级：北京前端训练营19期

讲师：彭帅伟

日期：

彭帅伟老师

QQ: 284337908

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

目录

[设计模式第一天 1](#_Toc7600)

[一、 设计模式 3](#_Toc19094)

[1.1 设计模式的分类 3](#_Toc910)

[1.2 设计模式的作用 3](#_Toc18138)

[1.3 设计模式的历史 3](#_Toc27341)

[二、 简单工厂 3](#_Toc19491)

[2.1 寄生增强工厂 4](#_Toc15493)

[2.2 工厂方法 4](#_Toc31327)

[2.3 原型与继承 4](#_Toc28481)

[2.4 安全工厂 6](#_Toc13050)

[下午复习： 7](#_Toc14237)

[2.5 闭包类 8](#_Toc21884)

[2.6 单例模式 8](#_Toc26357)

# 设计模式

设计模式： 是一套编目分明、 广为人知、可复用的代码经验的总结

工具库：一些常用方法的集合体， 比如jquery、underscore，通常这些方法之间是没有联系的

框架：一套半成品代码， 它里面也支持一些方法， 但是这些方法之间是有联系的

架构：一套大型项目的设计思路

## 1.1 设计模式的分类

所有的设计模式基本上可以分为三类：

1 创建型设计模式

2 结构型设计模式

3 行为型设计模式

## 1.2 设计模式的作用

创建型设计模式： 解决了创建对象时候的问题

结构型设计模式： 解决了对象和类组合在一起时候的问题

行为型设计模式： 解决对象和类偶合、职责之间的关系

## 1.3 设计模式的历史

设计模式最初是由国外的一个叫做GOF (Gang of four)设计出来， 设计模式一共有23种， 但是设计模式发展至今，已经远远超过23种

# 简单工厂

|  |
| --- |
| 1. function createObject(name, age, sex, job) { 2. var obj = { 3. name: name, 4. age: age, 5. sex: sex, 6. job: job 7. } 8. // 返回obj 9. return obj; 10. } 11. // 再次调用createObject 并且传递参数 12. var laozhang = createObject("老张", 30, "男", "IT"); 13. var laoli = createObject("老李", 30, "男", "厨师"); |

此时，createObject就是一个“简单工厂”， 每次放入不同的“原料”， 即可得到一个“产品”

## 2.1 寄生增强工厂

|  |
| --- |
| 1. // 定义新的工厂 2. function StrongPeople(name, age, sex) { 3. // 这是一个工厂， 在该工厂内部初始化其它的构造函数，这个就做寄生 4. var p = new People(name, age, sex); 5. // 对p进行改造 6. p.title = "nihao"; 7. // 添加方法 8. p.say = function() { 9. console.log("你好吗"); 10. } 11. // 返回p 12. return p; 13. } |

执行代码：

|  |
| --- |
| 1. var xiaogang = new StrongPeople("小刚", 12, "男"); |

结果：

|  |
| --- |
|  |

## 2.2 工厂方法

定义：在一个工厂内部，可以返回多个实例化对象

|  |
| --- |
| 1. // 定义新的工厂 2. function FactoryMethod(type) { 3. if (type === "dog") { 4. return new Dog(); 5. } else if (type === "cat") { 6. return new Cat(); 7. } else if (type === "pig") { 8. return new Pig(); 9. } 10. } |

## 2.3 原型与继承

原型：

|  |
| --- |
| 1. // 定义构造函数 2. function People(name, age, sex) { 3. this.name = name; 4. this.age = age; 5. this.sex = sex; 6. } 7. // 方法要写在原型上 8. People.prototype.sayHello = function() { 9. console.log("大家好，我的名字叫做" + this.name + "我的年龄是" + this.age + "我是一个" + this.sex + "孩子"); 10. } 11. // 实例化对象 12. var xiaoming = new People("小明", 12, "男"); 13. var xiaohong = new People("小红", 11, "女"); 14. // 调用原型上的方法 15. xiaoming.sayHello(); 16. xiaohong.sayHello(); 17. // 特性: 原型中的所有内容可以被实例化对象所访问 18. // 作用: 共享属性和方法 |

继承：

|  |
| --- |
| 1. // 模拟继承 2. // 定义父类 3. function People(name, age, sex) { 4. this.name = name; 5. this.age = age; 6. this.sex = sex; 7. } 8. // 方法要写在原型中 9. People.prototype.sayHello = function() { 10. console.log("大家好，我的名字叫做" + this.name + "我的年龄是" + this.age + "我是一个" + this.sex + "孩子"); 11. } 12. // 定义子类 13. function Student(name, age, sex, grade) { 14. // 使用构造函数式继承 15. People.apply(this, arguments); 16. this.grade = grade; 17. } 18. // 继承 19. // 使用类式继承 20. // 可以使用ES5提供 Object.create() 21. // Student.prototype = new People(); 22. Student.prototype = Object.create(People.prototype); 23. Student.prototype.constructor = Student; 24. // 这种继承方式会丢失constructor属性 25. // 补回 26. // 实例化Student 27. var s = new Student("小明", 12, "男", 6); |

## 2.4 安全工厂

|  |
| --- |
| 1. function People(name, age, sex) { 2. // 通过判断this指向， 从而决定代码该如何执行 3. if (this instanceof People) { 4. // 说明是使用new 来调用 5. this.name = name; 6. this.age = age; 7. this.sex = sex; 8. } else { 9. // 说明没有使用new 来调用 10. return new People(name, age, sex); 11. } 12. } |

# 下午复习：

设计模式：一套编目分明、广为人知、可复用的代码经验的总结

工具库：一些常用方法的集合体， 比如jquery、underscore，它们里面自带了一些放法，通常这些方法是没有联系的

框架： 一套半成品代码，它里面也支持一些方法， 这些方法之间是有联系的

架构：一套大型项目的设计思路

设计模式的分类与作用：

设计模式可以分为三类：

1 创建型设计模式： 解决了创建对象时候的一些问题

2 结构型设计模式： 解决了对象和类组合在一起时候的问题

3 行为型设计模式： 解决了对象和类之间的偶合、职责之间的关系

简单工厂： 就是创建对象的函数

寄生增强工厂：定义一个新的工厂， 在该工厂内部实例化对象，并对该实例化对象进行该造（增强）

工厂方法：定义一个工厂， 在该工厂内可以返回多个产品

原型：是每一个函数拥有的一个属性， 它的值是一个对象

特点: 原型上的所有内容被实例化对象所访问

作用：实例共享属性与方法

继承：类与类之间的继承

继承有5种方式：

1 类式继承

2 构造函数式继承

3 组合式继承

4 寄生式继承

5 寄生组合式继承

安全工厂： 无论外部如何调用类，都会返回该类的实例化对象

## 2.5 闭包类

|  |
| --- |
| 1. // 闭包类 2. (function() { 3. function People() { 4. } 5. })() |

## 2.6 单例模式

|  |
| --- |
| 1. var single = (function() { 2. function People(name, age, sex) { 3. this.name = name; 4. this.age = age; 5. this.sex = sex; 6. } 7. // 现在将People放入到闭包中，别人就无法访问到， 所以也就无法被实例化 8. // 但是，现在我们也不能实例化对象了 9. // 于是，我们要向外暴露一个接口，通过该接口可以控制对People的访问 10. // 用来保存单例 11. var instance = null; 12. return function(name, age, sex) { 13. // 判断单例是否存在 14. if (!instance) { 15. return instance = new People(name, age, sex); 16. } 17. // 如果存在直接返回 18. return instance; 19. } 20. })() |

执行代码:

|  |
| --- |
| 1. // 实例化对象 2. var p = single("小明", 12, "男"); 3. var p1 = single("小明1", 12, "男"); 4. var p2 = single("小明2", 12, "男"); 5. var p3 = single("小明3", 12, "男"); |

结果：

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |