# 设计模式，MVC，MVVM

第3天课堂笔记（本课程共4天）

[设计模式，MVC，MVVM](#_Toc26297)

[复习](#_Toc16150)

[适配器：](#_Toc24036)

[组合模式](#_Toc7291)

[观察者模式](#_Toc6049)

[策略模式](#_Toc10650)

[命令模式](#_Toc12512)

[命令模式渲染视图](#_Toc19762)

[Canvas绘图](#_Toc25791)

[迭代器模式](#_Toc4028)

[数组迭代器：](#_Toc21578)

[对象迭代器](#_Toc9923)

[面向对象中的迭代器](#_Toc15583)

[迭代器特点](#_Toc18468)

[Canvas滤镜](#_Toc6908)

[同步变量](#_Toc27697)

# 复习

## 适配器：

异步数据的适配

参数的适配

## 组合模式

将对象的组合表示成部分整体的属性结构，使其部分与整体之间的行为一致性

新浪新闻

## 观察者模式

定义了一种依赖关系，解决观察者与被观察者之间的耦合

Jquery callbacks

自定义观察者对象

观察者模式在面向对象中的应用

微博的开发

## 策略模式

将算法封装提取，使其具有一定的独立性，解决算法与客户端之间的耦合

Jquery easing

淘宝商品价格

表单验证

# 命令模式

定义：将请求与实现解耦并封装，实现请求对客户端实现参数化。

## 命令模式渲染视图

需求：页面中有好多数据源，我们要对这些数据源渲染页面，传统方式，我们渲染页面时候，将视图，数据，业务耦合在一起了，不利于我们开发，所以我们要将这些实现东西提取出来封装成一个个命令供我们使用，来实现代码复用，并简化我们的创建操作

是一种行为型设计模式

命令模式解决了命令的发起者与命令的实现者之间的耦合。

命令的发起者（调用命令的时候）我们不必去了解命令是如何实现的以及命令是如何运行的。

所以有的在使用上具有一致性

命令模式在一定成程度上简化了我们操作

## Canvas绘图

我们直接在页面中使用ctx会有一些问题

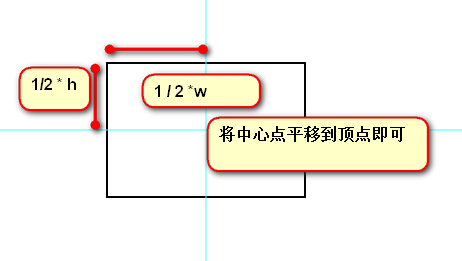
1 在模块外界访问不到ctx

2 ctx很重要，我们不论做什么操作都需要使用这个变量

3 一旦更改这个变量，会造成很严重的后果，造成我们原有功能失效。

fillRectCenter 方法实现原理

只需要将中心点平移到左上定点即可，平移的距离是横坐标w/2,纵坐标h/2



# 迭代器模式

定义：在不暴露对象内部结构的情况下，我们顺序的放聚合对象中内部的元素。

## 数组迭代器：

1 Jquery中each-》回调函数中的第一个参数表示索引值，第二个参数表示值，他的this既不是指向window，也不是指向数组本身

2 高级浏览器中，forEach数组迭代器，他的第一个参数表示是值，第二个表示索引值。他的this指向window

3 我们自定义数组迭代器，希望他this可以指向数组本身，在循环体中执行回调函数时，我们直接传入参数时候，那么实现forEach方法。如果我们将该回调函数在数组的作用域下执行，那么this就指向了原数组，此时，我们就可以在回调函数中通过this来访问元素组了

## 对象迭代器

1 jquery中each，他的回调函数中的第一个参数表示属性名，第二参数表示属性值。他的this既不是指向window，也不是指向对象本身

2 我们自定义的对象迭代器，

1. this指向window，——》在循环体的回调函数中，直接传入参数即可
2. This指向该对象，---》 在循环体内部的回调中，通过call方式在该对象的作用于下执行。

## 面向对象中的迭代器

面向对象中有时我们需要遍历实例化对象中的自身属性，过滤掉原型上的属性，此时我们可以通过hasOwnProperty来实现

## 迭代器特点

1 本身并没有删除循环语句，只是将循环语句转移到迭代器的内部，

2 迭代器模式中，数据对象内部的结构，我们只需要关注处理函数。

3 移出了循环语句，是我们的代码看着更清晰

4 迭代器模式是将数据源于处理函数之间的解耦

## Canvas滤镜



|  |
| --- |
| 1. // switch 改成策略模式 2. var DealStrategy = (function () { 3. var strategy = { 4. red: function (i) { 5. canvasData[i + 1] = 0; 6. canvasData[i + 2] = 0; 7. canvasData[i + 3] = a; 8. } 9. // ... 10. } 11. /\*\* 12. \* @type 算法名称 13. \* @i 第i元素 14. \*/ 15. return function (type) { 16. // strategy[type] 算法函数 17. // strategy[type](i) // 这种方式执行了，因为后面有圆括号，我们不想让他执行，我们只想得到算法的方法 18. return strategy[type] 19. } 20. })() 21. // 将for循环改成迭代器 22. function each (arr, method, step) { 23. for (var i = 0, len = arr.length; i < len; i += step) { 24. method(i) 25. } 26. } 27. // 寻找策略算法 算法名称叫type 28. var method = DealStrategy(type); 29. // 在迭代器中执行算法 30. each(canvasData, method, 4); |
|  |

案例中每次循环每个元素的时候，都要把switch分支走一遍，这样造成不相关的滤镜分支也要走一遍，造成资源的浪费，

Switch分支中每个分支都是一种算法，这样我们将它封装成一个策略对象，这样我们就可以根据策略对象直接获取策略算法，

For循环可以改装成迭代器，这样，我们将策略算法作为回调函数，每次循环直接执行

这样避免了switch判断对资源的浪费

## 同步变量

通常后端会我们变量供我们使用，但是我们不知道变量里面的具体内容。

通常这种情况下，我们贸然直接获取会报错（改变不存在的时候）

我们通过迭代器一级一级的去寻找，如果某一级不存在该属性了，我们就直接返回，这样避免错误的发生，

如果一级一级下去之后，一直存在，我们直接将其返回。