# 前端整体架构

1. 总体思路

基于MVC设计理念，尝试COP面向组件编程方式，宜拓展，松耦合。

1. 环境

开发工具： Flex Builder 4

Flex 4.0 SDK

Parsley框架 用于依赖注射 消息派发和拦截

1. 整体架构



C可以直接调用V和M中对象的方法，V只能通过回调的方式触发控制层的动作，M层不主动与其他层接触，当数据有所更改时触发响应事件，控制层的监听器接收到后进行逻辑处理。

1. 表现层



表现层被动接收控制层的指令，通过回调的方式反馈控制层。主要负责图像渲染、动画播放和元素加载，不涉及任何业务逻辑。表现层中的分成展示由C来控制。

对于复杂对象的图像可以拥有一定逻辑，但只和自身画面相关，过滤用户无效操作，简化控制层逻辑。

1. 控制层



C是程序、模块的起始入口，M和V的桥梁；负责渲染V，同时设置回调函数；负责调度M中的业务逻辑，使游戏玩起来有意思。同时切忌，C的工作是串接流程，具体实施则交给M处理，尽量保持代码简洁，因为多数的BUG都是发生在这个层。

C中为了完成业务流程，里面有众多的管理器，在收到M分发的数据更新事件时，通过Manager找到对应的逻辑单元，执行相应的流程。同时V中的回调函数，也要通过Manager检索出用户当前所操作的实体。

所谓的COP面向组件式编程就是在这一层实现的。COP是一种组织代码的思路，将原先通过对象继承来实现的多样化转为同级拼装的形式，将对象间复杂的继承关系拉伸、扁平化，新增功能不会对既有代码造成影响，更利于拓展。每个Entity皆为一个容器，可随意向其中增添组件addComponent。组件不存储数据，只完成固有逻辑，组件间可以相互调用，彼此的依赖通过parsley注射。同一组件类型，应该实现同一接口类，保持相同接口。

1. 持久层



持久层的作用：

1. 按数据对象格式存放数据
2. 按最小颗粒度封装业务逻辑，对外提供服务。
3. 发送与接收信息

持久层不与其他层主动发生关系，他是被动的，一旦数据发生变化，就要广播响应事件，事件中附带数据对象。

持久层是游戏的灵魂，缺了他游戏就不能玩，但缺了V和C，理论上用户依旧可以通过命令行的方式继续玩。

1. 示例流程

角色P从A区域移动到B区域，获得一物品S。按照上面的设计工作流应该如下：

玩家 V C M

选择角色P click callback -> Set selected

Highlight(p)

点击区域B click B -> Move(A,B)

setPosition(p,B) <- setView

arriveB -> onArrive

setM -> move(A,B)

checkAvaliable

update p

dispatch event “move”

listener onMove

gainStuff S -> checkGain

-> addStuff(p,s)

dispatch event “addstuff”

listener onAddStuff

updatePocket <- setView

获得物品

1. COP设计
2. 场景设计

World 单例，整个游戏的通用设置，包括物理特性等。

Scene 场景，world就是由多个scene组成的。

Layer 层。Scene中多层显示，每个层负责管理本层对象

LayerMgr 层管理器。Scene