电影售票系统-软件设计文档

本项目采用的是系统分析与设计的课程项目

**Web客户端**

技术选型：angular2框架

为什么选择angular2

1. **ALL-IN-ONE**

不管是1还是2，Angular最显著的特征就是其整合性。它是由单一项目组常年开发维护的一体化框架，涵盖了M、V、C/VM等各个层面，不需要组合、评估其它技术就能完成大部分前端开发任务。这样可以有效降低决策成本，提高决策速度，对需要快速起步的团队是非常有帮助的。

1. **模块化的技术**

虽然Angular是一个ALL-IN-ONE的框架，但这并不妨碍它同时是一个灵活的框架.Angular 2很好的践行了OCP原则（开闭原则），即对拓展是开放的，对修改是关闭的

Angular非常有效的分离了渲染与交互逻辑，这就让它可以很好的跟包括React在内的渲染引擎搭配使用，除此之外，它还可以使用内存渲染引擎，以实现服务端渲染；还可以使用Native渲染引擎，以编译出真正的原生程序（NativeScript）

它还分离了数据供应与变更检测逻辑，从而让它可以自由使用包括RxJS以及ImmutableJS在内的第三方数据框架/工具。

不仅如此。

在Angular 1和Angular 2中最具特色的应该算是依赖注入(DI)系统了，它把后端开发中用来解决复杂问题、实现高弹性设计的DI技术引入了前端。Angular 2中更是通过引入TypeScript赋予它更高的灵活性和便利性。

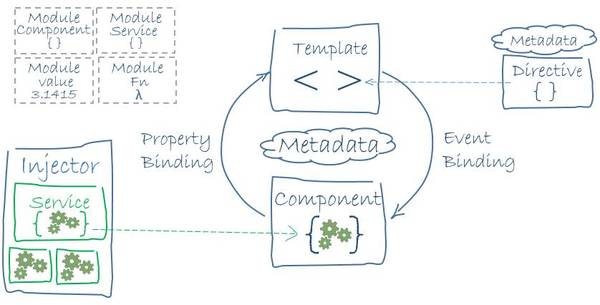
1. **速度**

Angular 2终于摆脱了旧的技术框架束缚，开始了对速度的极致追求。在Angular 1中，虽然绝大多数场景下性能都不是问题，不过还是因为其代码中存在的一个用来实现脏检查的三重循环而饱受抨击 —— 似乎真有人能感受到30毫秒和100毫秒的差异似的。

不过，有软肋总是不太好。于是，在Angular 2中，通过重新设计和引入新技术，从原理上对速度进行了提升，据官方以前提供的一个数据说是把“变更检测”的效率提升了500%。

**架构设计**

Angular2架构图：



这个架构图展示了8个Angular 2应用的构成块：

模块 (module)

组件 (component)

模板 (template)

元数据 (metadata)

数据绑定 (data binding)

指令 (directive)

服务 (service)

依赖注入 (dependency injection)

**模块 （Module）**

首先，根模块是用来启动程序的。没有根模块，再好的程序也跑不起来。所以模块很重要很重要。Angular的应用，都是模块化的。每个feature特性都有自己的子模块，无论是子模块还是根模块定义的结构都差不多。我们可以把模块看成一个行李箱，里面装满了各种东西。包括declaration、imports、providers等。declaration用来放各种小零件，包括组件、指令、管道等。imports是导入其他的模块。providers用来盛放各种service。

**组件 (Component) 元数据 (Metadata) 模板 (Template)**

模板就是HTML，就是你告诉Angular，喂喂喂，这儿你给我加个按钮，那儿你给我来个日历就是这样。除了<h1> <p>等等这些常见的html元素，我们还可以加自定义的元素在模板里面。

组件负责让模板看起来没那么无聊。比如你在模板里面定义了一个按钮，那你按了之后发生了什么啊？或者你定义了一个input box，你输入你的名字后会发生什么啊？这就需要Component来定义。

那每个Component都不一样，我们怎么区分呢？其实应该说Angular怎么识别呢？就是靠这个叫做元数据的东西。元数据会定义一些配置细节，告诉Angular：嘿Angular，如果你想把这个东西给大家看，那你得先去做ABCDEFG这几件事。Angular按照元数据的指示，根据Template的样子最后创建视图。

**数据绑定 (Data Binding)**

Angular2比Angular1更好滴实现了数据绑定。一共有四种吧：

{{someString}}这个叫插值绑定，从Component到DOM，之前在Angular1里面也有

[someThing] 这个叫属性绑定，也是从Component到DOM

（event）这个是事件绑定，从DOM到Component

最后一个就是把上面这两个结合在一起[(ng...)] 双向绑定

**指令 (directive)**

指令跟组件(Component)非常类似，其实组件就相当于一个指令。指令地位崇高，包括结构型指令和属性指令。

结构型我们在Angular1里面也经常用，就是现在语法不太一样了，比如ngfor, ngIf等等，都是结构型指令。这些一般是不会改变任何元素的行为，外观等等。相反，属性型就非要改外观或者行为。

**服务（Service）**

服务是一个广义范畴，包括：值、函数，或应用所需的特性。

几乎任何东西都可以是一个服务。 典型的服务是一个类，具有专注的、明确的用途。它应该做一件特定的事情，并把它做好。

服务可以用来组件通信等。

**依赖注入（Dependency Injection）**

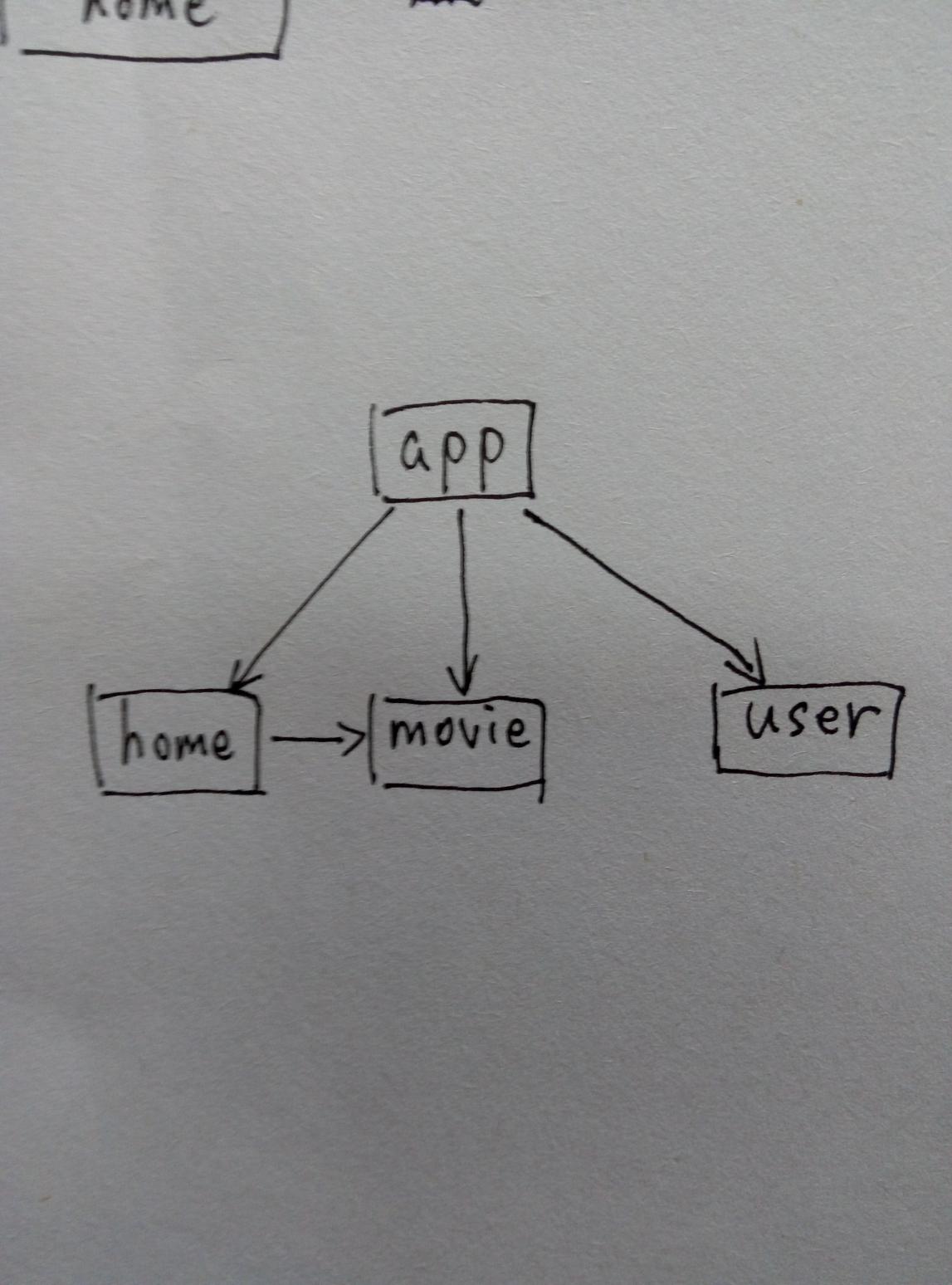
“依赖注入”是提供类的新实例的一种方式，还负责处理好类所需的全部依赖。大多数依赖都是服务。 Angular 使用依赖注入来提供新组件以及组件所需的服务。

Angular 通过查看构造函数的参数类型得知组件需要哪些服务。

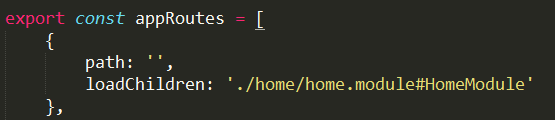
当 Angular 创建组件时，会首先为组件所需的服务请求一个注入器 (injector)。

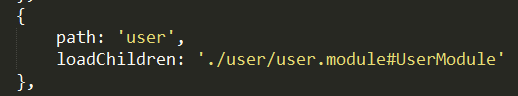
注入器维护了一个服务实例的容器，存放着以前创建的实例。 如果所请求的服务实例不在容器中，注入器就会创建一个服务实例，并且添加到容器中，然后把这个服务返回给 Angular。 当所有请求的服务都被解析完并返回时，Angular 会以这些服务为参数去调用组件的构造函数。 这就是依赖注入 。

**模块划分**

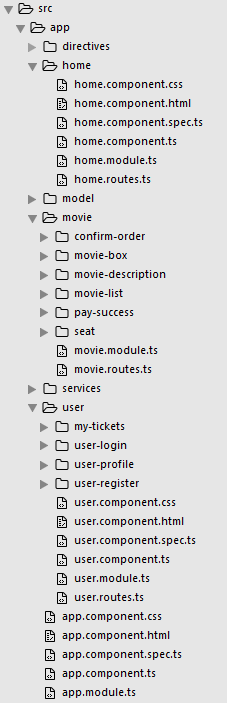
****

其中app是根模块，angular通过引导根模块来启动应用，当需要进入其他模块时，父模块可以调用其子模块：





项目目录：



**项目中使用的软件设计技术：**

1. **SOA（面向服务架构）**

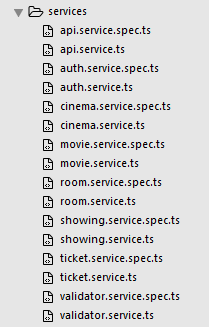
**SOA定义**

　　它是一个组件模型，它将应用程序的不同功能单元(称为服务)通过这些服务之间定义良好的接口和契约联系起来。接口是采用中立的方式进行定义的，它应该独立于实现服务的硬件平台、操作系统和编程语言。这使得构建在各种这样的系统中的服务可以以一种统一和通用的方式进行交互。

　　这种具有中立的接口定义(没有强制绑定到特定的实现上)的特征称为服务之间的松耦合。松耦合系统的好处有两点，一点是它的灵活性，另一点是，当组成整个应用程序的每个服务的内部结构和实现逐渐地发生改变时，它能够继续存在。而另一方面，紧耦合意味着应用程序的不同组件之间的接口与其功能和结构是紧密相连的，因而当需要对部分或整个应用程序进行某种形式的更改时，它们就显得非常脆弱。

对松耦合的系统的需要来源于业务应用程序需要根据业务的需要变得更加灵活，以适应不断变化的环境，比如经常改变的政策、业务级别、业务重点、合作伙伴关系、行业地位以及其他与业务有关的因素，这些因素甚至会影响业务的性质。我们称能够灵活地适应环境变化的业务为按需(On demand)业务，在按需业务中，一旦需要，就可以对完成或执行任务的方式进行必要的更改。

项目代码：



src/app/service/movie.service.ts片段：

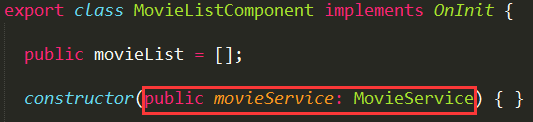


src/app/movie/movie-list/movie-list.component.ts片段：

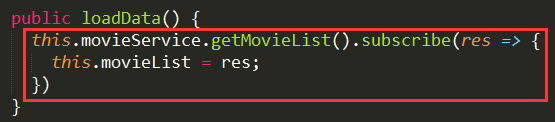
Import service:



依赖注入service：



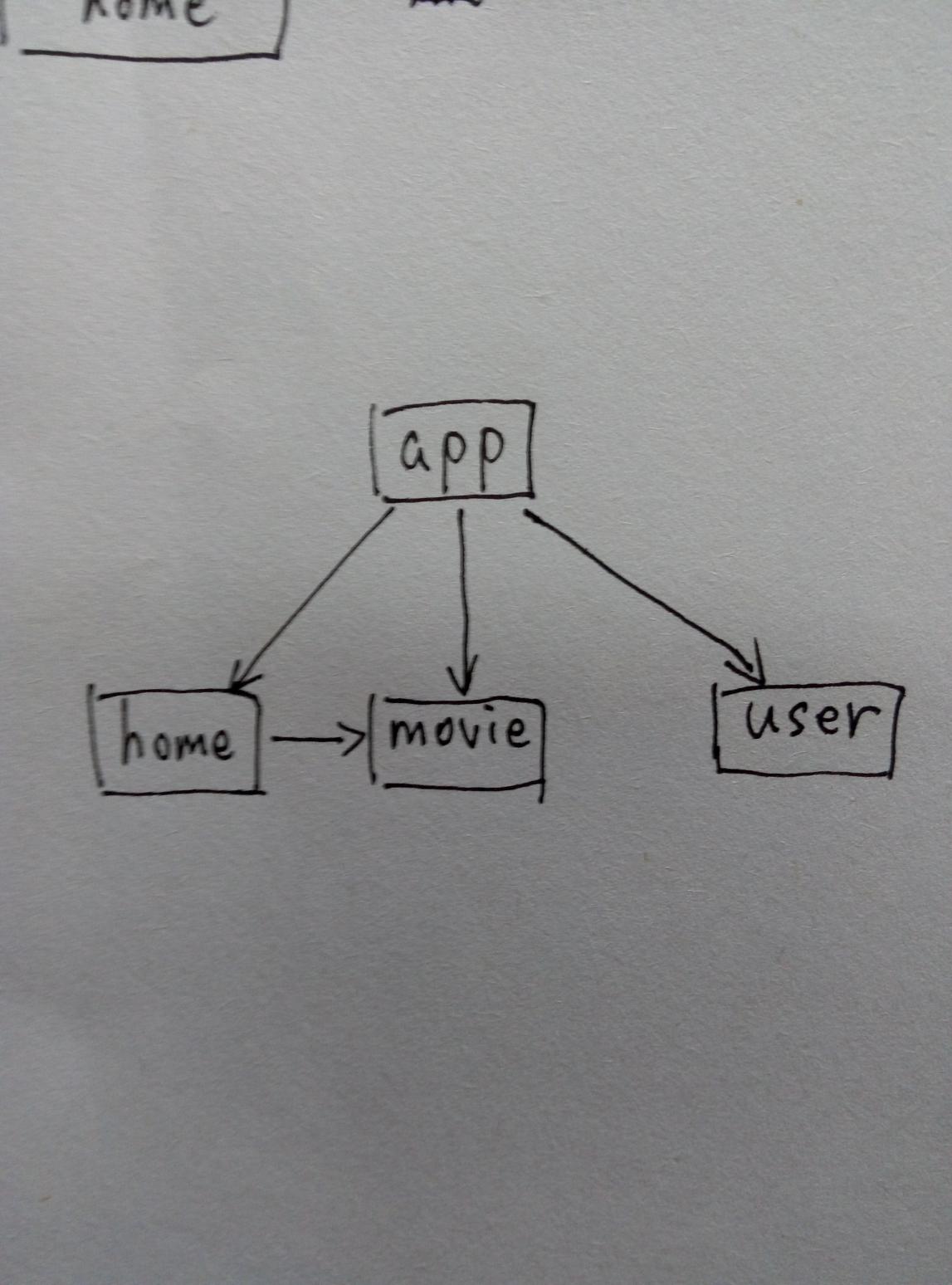
调用服务：



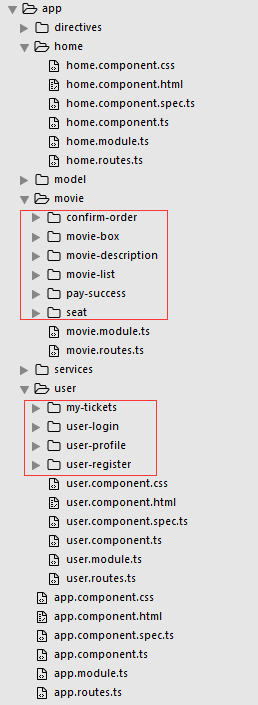
**2.结构化程序设计**

结构化程序设计的思路是：[自顶向下](https://www.baidu.com/s?wd=%E8%87%AA%E9%A1%B6%E5%90%91%E4%B8%8B&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLrHTvnymvPHubmWnkn16Y0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHTkPj0vrHmk)、逐步求精；其程序结构是按功能划分为若干个基本模块；各模块之间的关系尽可能简单，在功能上相对独立；每一模块内部均是由顺序、选择和循环三种基本结构组成；其模块化实现的具体方法是使用子程序。结构化程序设计由于采用了模块分解与功能抽象，[自顶向下](https://www.baidu.com/s?wd=%E8%87%AA%E9%A1%B6%E5%90%91%E4%B8%8B&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1YLrHTvnymvPHubmWnkn16Y0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPHTkPj0vrHmk)、分而治之的方法，从而有效地将一个较复杂的程序系统设计任务分解成许多易于控制和处理的子任务，便于开发和维护。

本项目模块图：

****

本项目按功能划分为若干个基本模块以及组件，本项目的目录：



其中红框内的为本项目的组件（按功能划分）

**服务端**

技术选型

服务端是Java web后端，采用Spring MVC框架，采用Hibernate持久化技术，数据库采用MySQL

理由

一、SpringWeb MVC是什么

spring Web MVC是一种基于Java的实现了Web MVC设计模式的请求驱动类型的轻量级Web框架

使用了MVC架构模式的思想，将web层进行职责解耦，基于请求驱动指的就是使用请求-响应模型

框架的目的就是帮助我们简化开发，Spring Web MVC也是要简化我们日常Web开发的。

二、 SpringWeb MVC能帮我们做什么

让我们能非常简单的设计出干净的Web层和薄薄的Web层

进行更简洁的Web层的开发

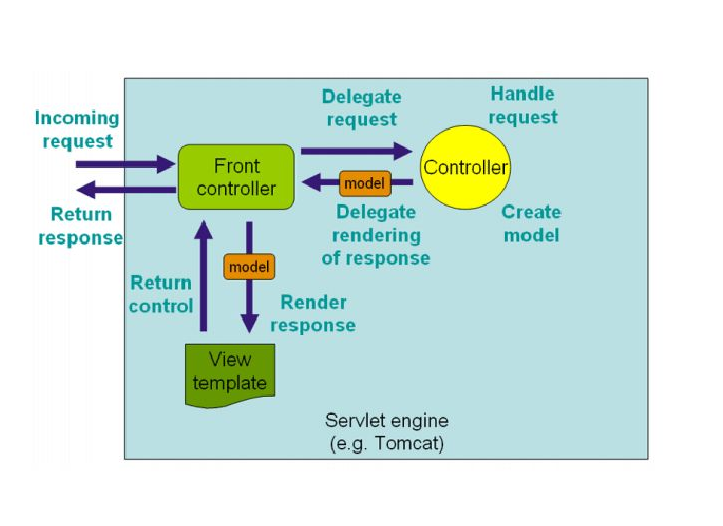
天生与Spring框架集成（如IoC容器、AOP等）

提供强大的约定大于配置的契约式编程支持

非常灵活的数据验证、格式化和数据绑定机制

支持Restful风格

三、 SpringWeb MVC处理请求的流程



1.spring mvc请所有的请求都提交给DispatcherServlet,它会委托应用系统的其他模块负责负责对请求进行真正的处理工作。

2.DispatcherServlet查询一个或多个HandlerMapping,找到处理请求的Controller.

3.DispatcherServlet请请求提交到目标Controller

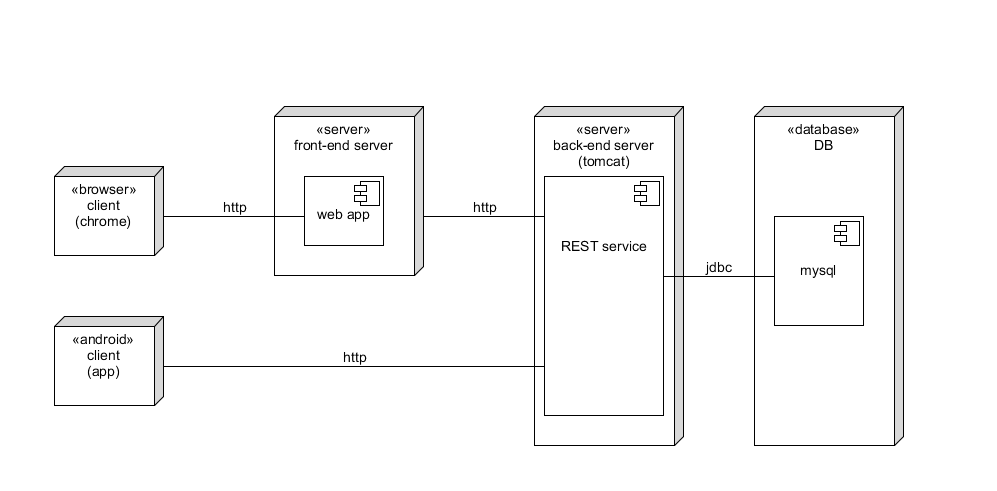
4.Controller进行业务逻辑处理后，会返回一个ModelAndView

5.Dispathcher查询一个或多个ViewResolver视图解析器,找到ModelAndView对象指定的视图对象

6.视图对象负责渲染返回给客户端

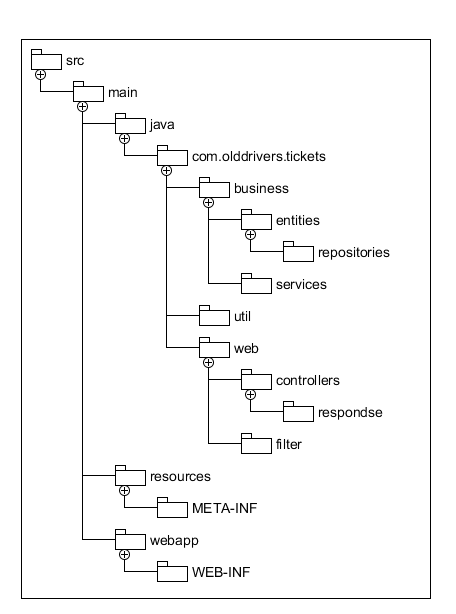
架构设计

总体架构设计

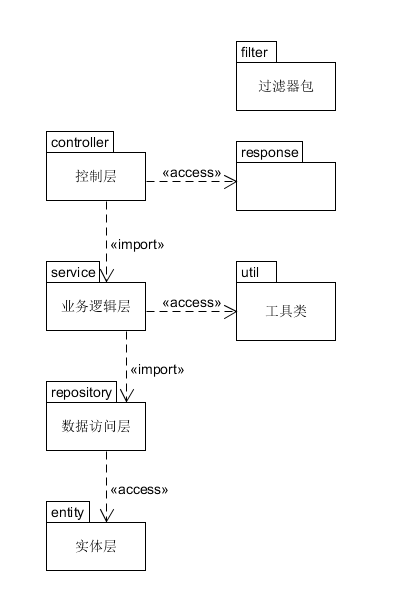


服务端模块划分

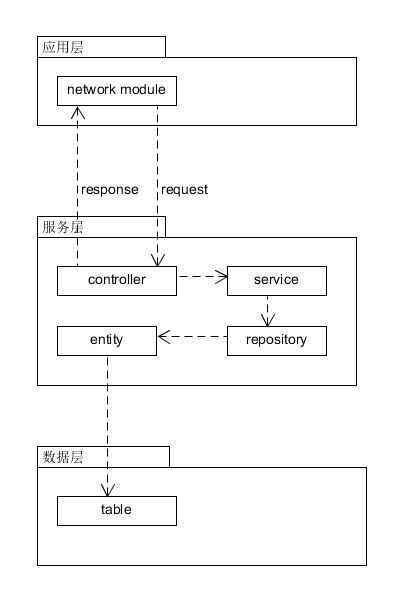
文件结构图



包图

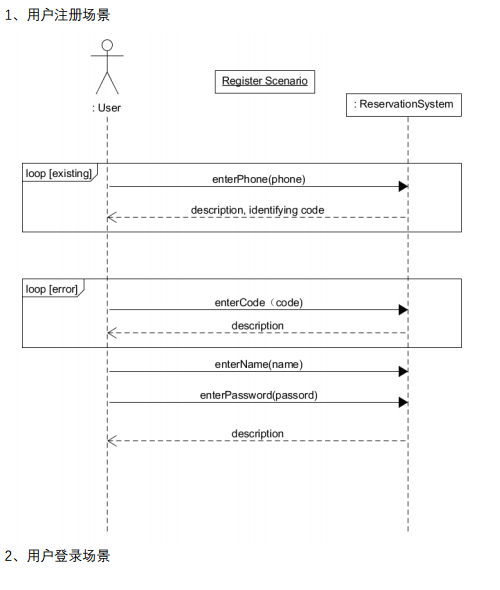


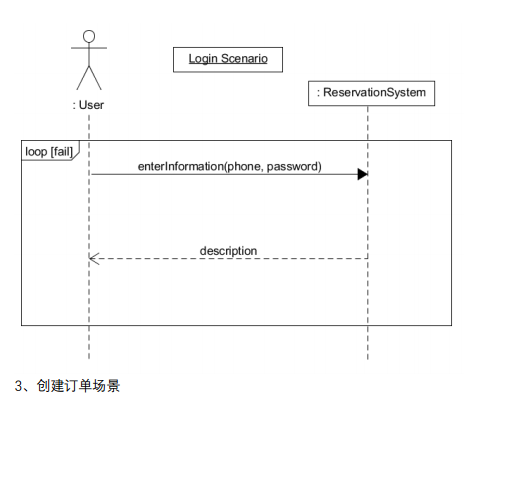
系统处理视图

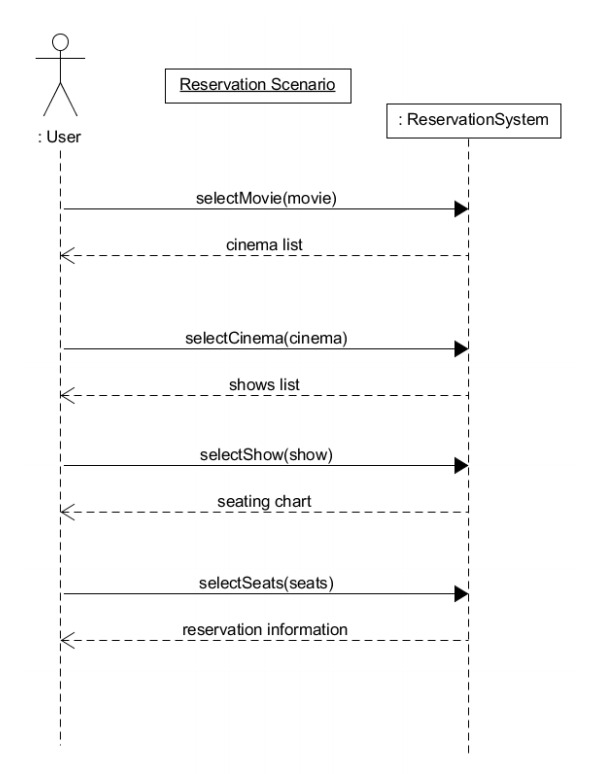


系统工作的基本流程：应用层发送请求到服务层，服务层中controller接受后，向下调用service，service通过repository对entity进行对应的操作，entity映射为数据层中对应的表，转为数据库进行操作后，controller整理返回response给应用层

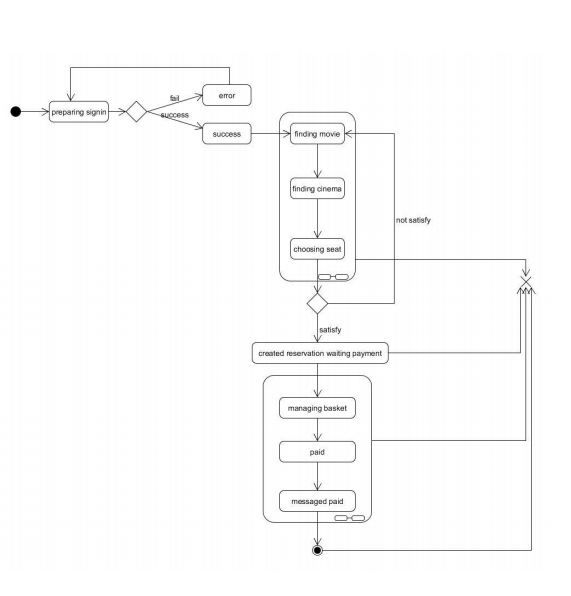
场景视图







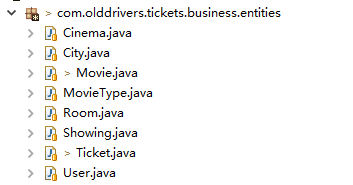
状态图



软件设计技术

Object-Oriented Programming

以下八个POJO类为服务端的基础数据类，采用ORM对象映射技术映射成数据库中对应的表，对这些类的操作即可作用到底层的数据库数据。



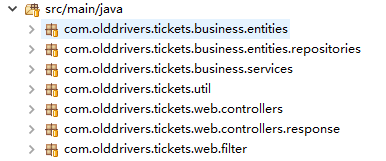
MVC模式

MVC即Model-View-Controller，把一个应用的输入、处理、输出流程按照Model、View、Controller的方式进行分离，这样一个应用被分成三个层——模型层、视图层、控制层。

视图(View)代表用户交互界面，对于Web应用来说，可以是HTML，也可能是jsp、XML和Applet等。一个应用可能有很多不同的视图，MVC设计模式对于视图的处理仅限于视图上数据的采集和处理，以及用户的请求，而不包括在视图上的业务流程的处理。业务流程的处理交予模型(Model)处理。

模型(Model)：是业务的处理以及业务规则的制定。模型接受视图请求的数据，并返回最终的处理结果。业务模型的设计是MVC最主要的核心。MVC设计模式告诉我们，把应用的模型按一定的规则抽取出来，抽取的层次很重要，抽象与具体不能隔得太远，也不能太近。MVC并没有提供模型的设计方法，而只是组织管理这些模型，以便于模型的重构和提高重用性。

控制(Controller)可以理解为从用户接收请求, 将模型与视图匹配在一起，共同完成用户的请求。划分控制层的作用也很明显，它清楚地告诉你，它就是一个分发器，选择什么样的模型，选择什么样的视图，可以完成什么样的用户请求。控制层并不做任何的数据处理。



注：本系统中服务端实现的是RESTful接口，返回给客户端的是JSON数据，而不是视图

**Android客户端**

技术选型：

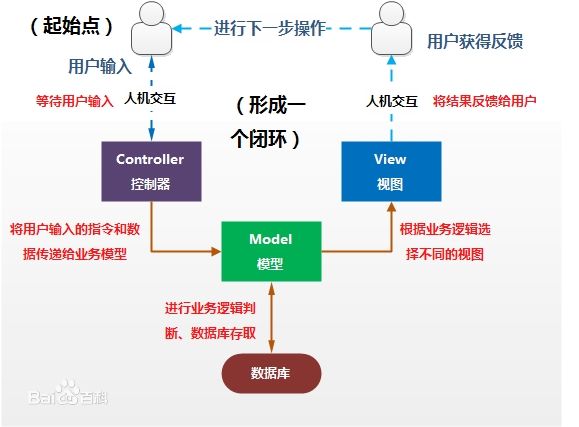
原生的Android应用使用Java编程语言开发，并且很容易的移植到其他移动平台上，可以在Windows、Mac还有Linux上都能完成Android的开发工作。谷歌为Android开发者营造了一个极为良好的开发环境。Android Studio开发工具可以极大的方便和有利于Android平台应用开发，其开发环境和模式更加的丰富和便捷，能够支持多种语音，还可以为开发者提供测试工具和各种数据分析。

此外Android Studio还支持基于Gradle的灵活的构建系统；支持构建变种版本和生成多个APK文件；支持通过代码模板来快速建立通用的app功能；支持丰富的布局编辑器，可拖放主题编辑；支持lint工具来捕获性能、可用性、版本兼容等问题；支持代码混淆和应用签名内置的Google云平台支持。因此很容易整合Google云消息和app引擎内置了ADB工具，可以方便的查看log内置了性能检测工具,可以实时的检测内存、CPU、流量、帧率。

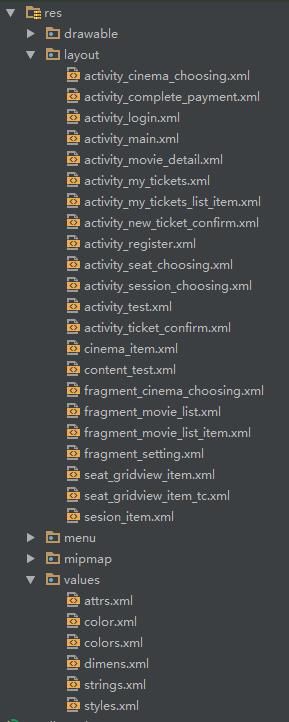
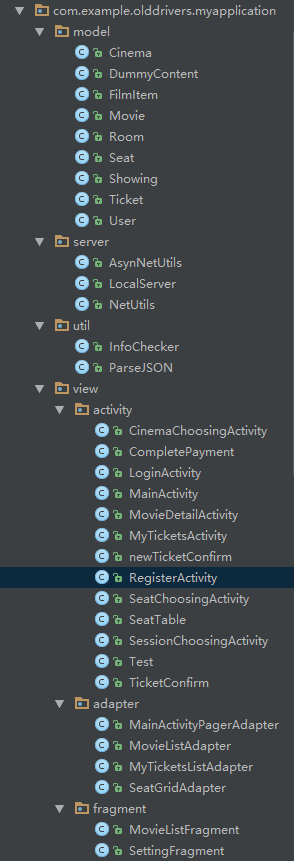
架构设计

本项目使用了MVC框架，因为代码量不算太多，很多数据处理的操作已经封装起来，所以activity并不会很臃肿，所以没有再使用MVP框架。

MVC业务流程

、

项目文件结构



M层：业务逻辑处理，网络请求，json文件处理，输入信息检查，本地信息储存，在项目util和server文件夹中

V层：应用层中处理数据显示的部分，XML布局，显示Model层的数据结果。

C层：在Android中，Activity处理用户交互问题，因此可以认为Activity是控制器，Activity从V视图层读取数据（eg.读取服务器数据），显示到界面（eg.EditText.setText），并向Model发送数据请求（eg.发起网络请求等）。

MVC的优点：

(1)耦合性低。所谓耦合性就是模块代码之间的关联程度。利用MVC框架使得View（视图）层和Model（模型）层可以很好的分离，这样就达到了解耦的目的，所以耦合性低，减少模块代码之间的相互影响。但是在Activity中还是有关于视图UI的显示代码，因此View视图和Activity控制器并不是完全分离的，也就是说一部分View视图和Contronller控制器Activity是绑定在一个类中的。

(2)可扩展性好。由于耦合性低，添加需求，扩展代码就可以减少修改之前的代码，当需要改变UI显示的时候，无需修改Contronller（控制器）Activity的代码和Model（模型）模型中的业务逻辑代码，很好的将业务逻辑和界面显示分离，降低bug的出现率。

(3)模块职责划分明确。主要划分层M,V,C三个模块，利于代码的维护。

Android软件技术：

（1）网络请求的封装:  
1)实现并发的网络请求，在主线程进行网络请求,因为网络请求耗时较长，会报错。  
2)不同环境对应不同的网络请求方式(post/get)。

3)网络请求是异步的，在另外一个线程里面执行

4)网络请求中，把变化的、动态的抽取出来，比如请求参数、请求地址，回调函数；把静态的，不可变的封装起来，不对外暴露，比如相同的请求头、公共的请求参数，公共请求参数的处理、统一错误处理、公共的返回参数。

5)因为HttpClient早就已经不在更新，只适合低版本Android，所以网络请求框架的底层都是HttpURLConnection  
6)没有使用OkHttp等网络库

7)没有实现发送网络请求之前，判断网络连接状况

（2）java单例模式以及Android中单例模式的问题

项目中有一些变量需要储存在全局，比如说登录的用户信息，在登录后到应用结束之前这个信息应该一直被保存下来。

所以我第一个想到的就是使用单例模式，但之前都是c++中使用单例模式，于是上网学习java的单例模式

单例模式是指在一个类使用单例模式，在这个类被创建后，只可能产生一个实例供外部访问，并且提供一个全局的访问点

要求：

(1) 将采用单例设计模式的类的构造方法私有化（采用private修饰），不允许使用者调用构造方法。

(2) 在类其内部产生该类的实例化对象，并将其封装成private static类型，因为只有一个实例。

(3) 定义一个静态方法返回该类的实例。

方法一

单例模式的实现：饿汉式,线程安全 但效率比较低

public class SingletonTest {

// 定义一个私有的构造方法

private SingletonTest() {

}

// 将自身的实例对象设置为一个属性,并加上Static和final修饰符

private static final SingletonTest instance = new SingletonTest();

// 静态方法返回该类的实例

public static SingletonTest getInstancei() {

return instance;

}

}

优点是：写起来比较简单，当类SingletonTest被加载的时候，会初始化static的instanc，不存在多线程同步问题，避免了synchronized所造成的性能问题；  
缺点是：当类SingletonTest被加载的时候，会初始化static的instance，静态变量被创建并分配内存空间，从这以后，这个static的instance对象便一直占着这段内存（即便你还没有用到这个实例），当类被卸载时，静态变量被摧毁，并释放所占有的内存，因此在某些特定条件下会耗费内存。

方法二

饱汉模式  
优点是：写起来比较简单，当类SingletonTest被加载的时候，静态变量static的instance未被创建并分配内存空间，当getInstance方法第一次被调用时，初始化instance变量，并分配内存，因此在某些特定条件下会节约了内存；  
缺点是：并发环境下很可能出现多个SingletonTest实例。

代码因篇幅问题省略了

最佳实现为《Android 源码设计模式》中的方式

public class SingletonTest {  
  
private SingletonTest() {   
}   
  
public static synchronized SingletonTest getInstance() {   
return SingletonHolder.instance;   
}   
  
private static class SingletonHolder{  
private static SingletonTest instance = new SingletonTest();  
}  
}

在第一次使用时创建，效率高，并且线程安全。

但是！在Android中只是这样使用单例模式是会出问题的

如果你在接收数据的Activity中，按下Home键返回桌面，长时间的没有返回你的App。系统有可能会在系统内存不足的时候杀掉进程。当你再从最近程序运行列表进入你的App，系统会默认恢复刚刚离开的状态，直接进入接收数据的Activity中，然后调用各个生命周期方法回调，其中只要运行到从单例模式类取数据行，程序就会弹出空指针NullPointerException异常导致崩溃，或者取到的数据为空。

因为上一次的单例模式的数据已经在系统内存不足，并且程序在后台的时候被清除掉了

查找解决方法如下

还是直接将数据通过intent传递给 Activity 。

使用官方推荐的几种方式将数据持久化到磁盘上，再取数据。

在使用数据的时候总是要对变量的值进行非空检查，这样还是取不到数据

使用EventBus传递数据时采用onSaveInstanceState(Bundle outState)方法保存数据，使用onCreate(Bundle savedInstanceState)等待恢复取值。

这里因为项目比较简单，直接将信息存在sharedpreferences中，这样存在磁盘就不会被清除丢失了，不过这样的缺点是存在安全和隐私泄露的问题。

2）Android中的消息处理

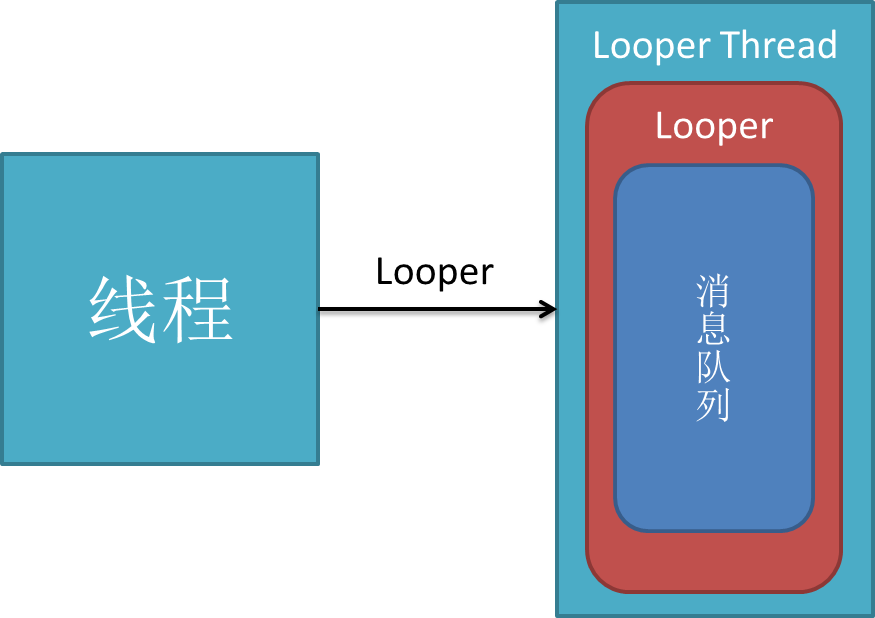
首先是Looper类

Looper可以使一个普通线程变成Looper线程，所谓Looper线程就是循环工作的线程。

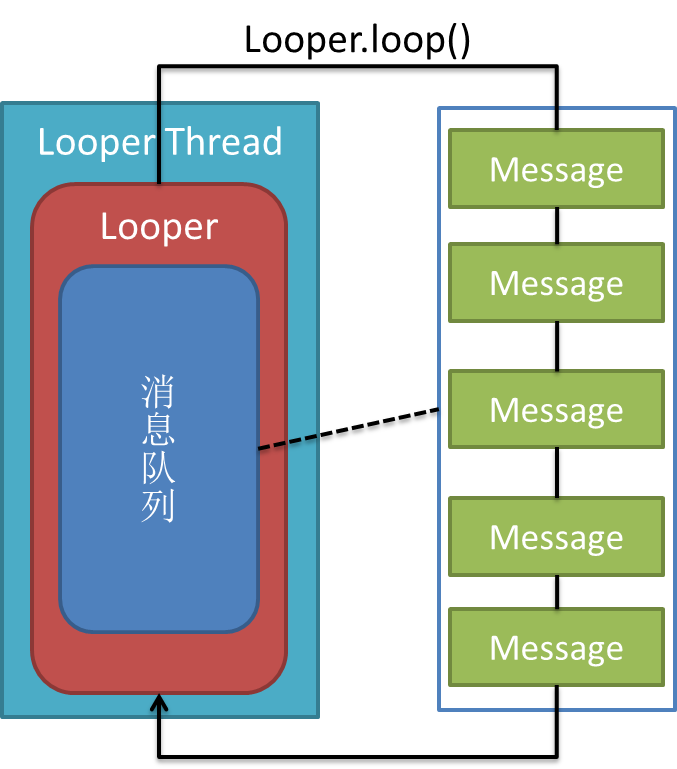
public class LooperThread extends Thread {  
 @Override  
 public void run() {  
 // 将当前线程初始化为Looper线程  
 Looper.prepare();  
   
 // ...其他处理，如实例化handler  
   
 // 开始循环处理消息队列  
 Looper.loop();  
 }  
}

通过上面两行核心代码，你的线程就升级为Looper线程了

Looper.prepare()



Looper.loop()



调用loop方法后，Looper线程就开始真正工作了，它不断从自己的MQ中取出队头的消息(也叫任务)执行。

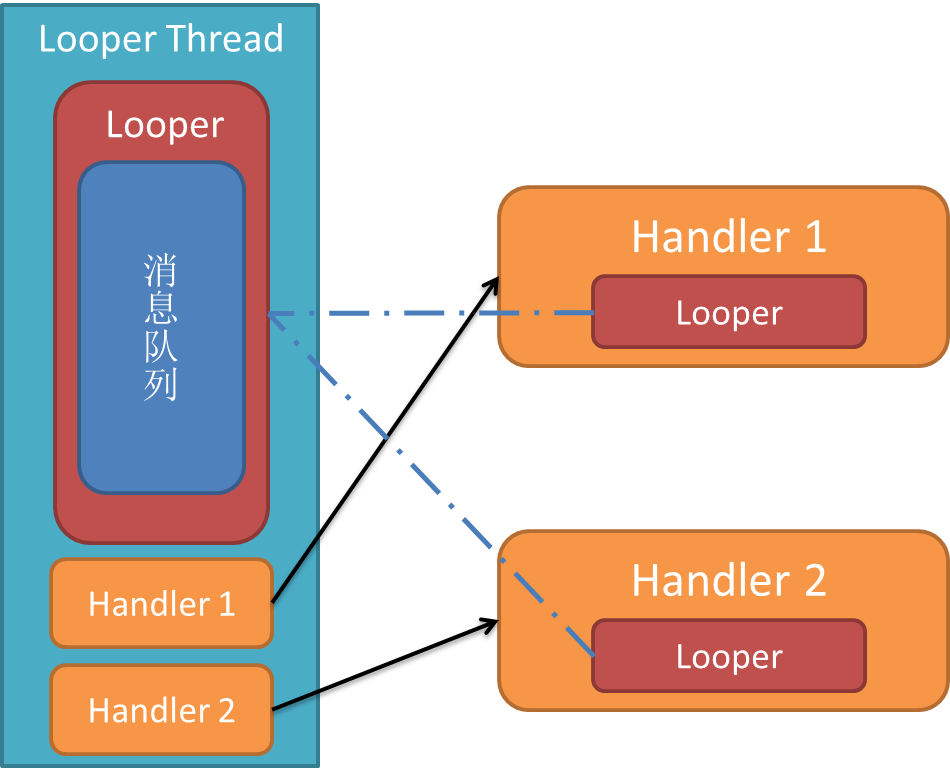
1.每个线程有且最多只能有一个Looper对象，它是一个ThreadLocal

2.Looper内部有一个消息队列，loop()方法调用后线程开始不断从队列中取出消息执行

3.Looper使一个线程变成Looper线程。

如何在消息队列中添加消息呢，就要使用Handler了

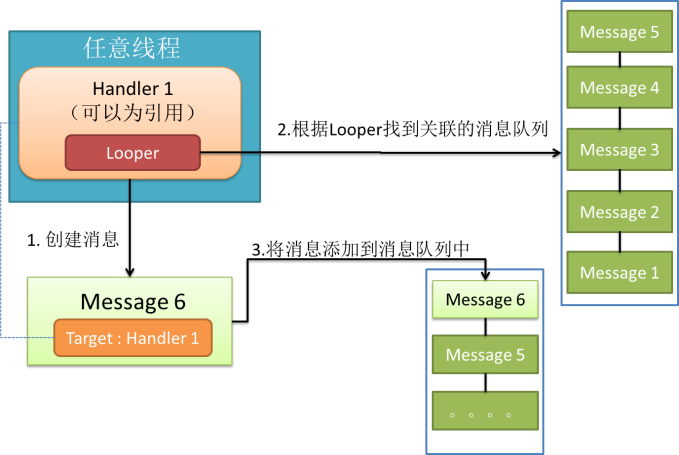
handler扮演了往MQ上添加消息和处理消息的角色（只处理由自己发出的消息），通知MQ它要执行一个任务(sendMessage)，并在loop到自己的时候执行该任务(handleMessage)，整个过程是异步的



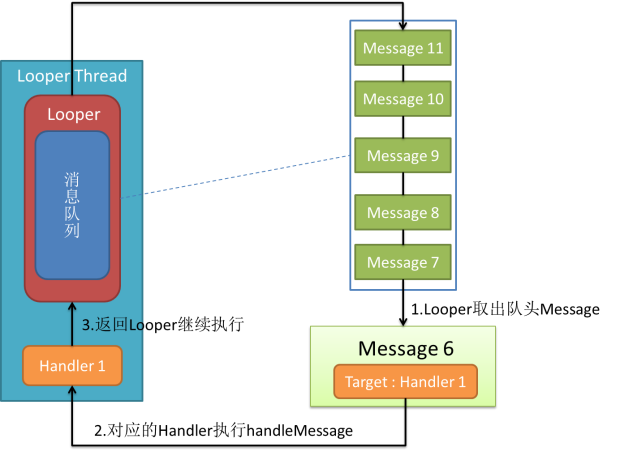
一个线程可以有多个Handler，但是只能有一个Looper！

有了handler之后，我们就可以使用 [post(Runnable)](http://developer.android.com/reference/android/os/Handler.html#post(java.lang.Runnable)), [postAtTime(Runnable, long)](http://developer.android.com/reference/android/os/Handler.html#postAtTime(java.lang.Runnable, long)),[postDelayed(Runnable,long)](http://developer.android.com/reference/android/os/Handler.html#postDelayed(java.lang.Runnable, long)), [sendEmptyMessage(int)](http://developer.android.com/reference/android/os/Handler.html#sendEmptyMessage(int)), [sendMessage(Message)](http://developer.android.com/reference/android/os/Handler.html#sendMessage(android.os.Message)), [sendMessageAtTime(Message, long)](http://developer.android.com/reference/android/os/Handler.html#sendMessageAtTime(android.os.Message, long))和[sendMessageDelayed(Message, long)](http://developer.android.com/reference/android/os/Handler.html#sendMessageDelayed(android.os.Message, long))这些方法向MQ上发送消息了

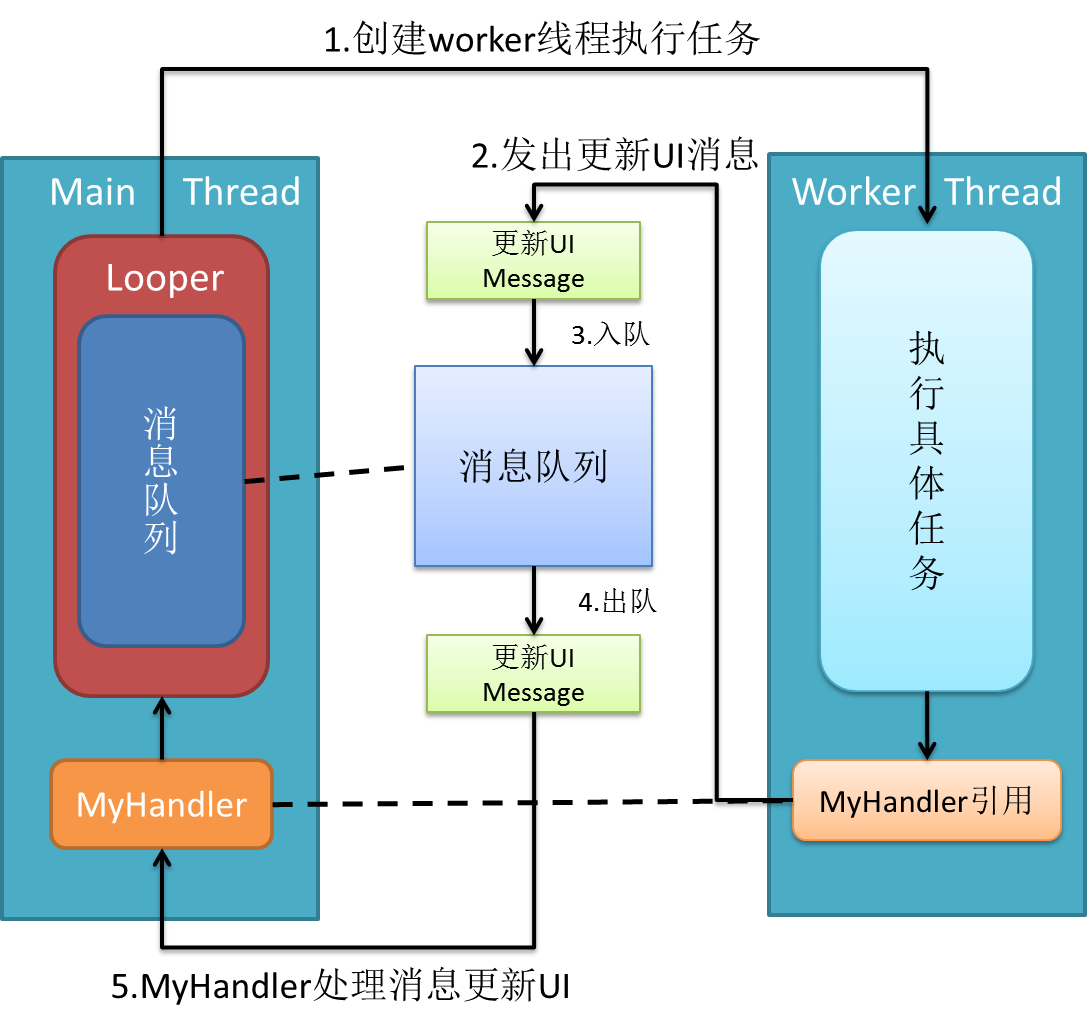
1.handler可以在任意线程发送消息，这些消息会被添加到关联的MQ上。



2.handler是在它关联的looper线程中处理消息的。



这就解决了android最经典的不能在其他非主线程中更新UI的问题。android的主线程也是一个looper线程



利用handler的一个solution就是在activity中创建handler并将其引用传递给worker thread，worker thread执行完任务后使用handler发送消息通知activity更新UI。

这次项目中的网络访问就是这样实现异步操作和更新UI的。