社交网络系统

软件构架文档

版本 <1.0>

修订历史记录

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **版本** | **说明** | **作者** |
| 2015.3.26 | 1.0 | 迭代一 | 张子辰 |
| 2015.4.22 | 2.0 | 迭代二 | 张子辰 |
| 2015.5.22 | 2.1 | 补充设计模式 | 张子辰 |
|  |  |  |  |

目录

1. 简介 4

1.1 目的 4

1.2 功能 4

1.3 开发平台和环境 4

2. 架构风格 4

3. 逻辑视图 5

3.1 类图 5

3.2 包图 6

4. 进程视图 7

5. 部署视图 8

6. 开发视图 8

7. 监控与恢复 8

8. 可靠性 9

9. 安全 9

10. 可修改性 9

11. 易用性 9

12. 可测试性 9

13. 设计模式 10

13.1 工厂模式 10

13.2 装饰器模式 10

13.3 构造器模式 10

13.4 桥接模式 10

13.5 迭代器模式 10

13.6 适配器模式 10

13.7 抽象工厂模式、单例模式 10

软件构架文档

# 简介

## 目的

本文档将从构架方面对社交网络系统（即SNS）进行综合概述，其中会使用4+1视图来描述系统的各个方面。接着，它会从安全、故障监控与备份、易用性、可靠性来详细描述该系统的设计。

## 功能

该社交网络系统具有如下功能：发微博、写博客和即时聊天。

## 1.3 开发平台和环境

开发语言：Java。

组件或框架：servlet+jsp。

建模工具：PowerDesign。

数据库：mysql。

# 架构风格

该网站共使用了以下架构：

（1）浏览器和服务端间用了B/S架构。

（2）服务端的程序整体上用了MVC三层架构。

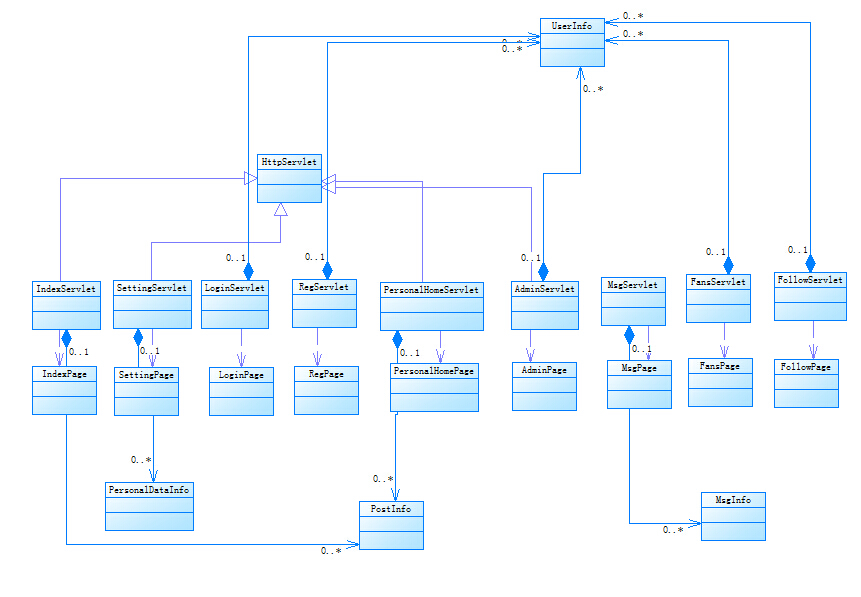
（3）控制层和数据层间用了黑板报模型。

（4）控制层的输入校检处用了管道过滤器模型。

在web开发方面，基本都是套现有的框架或者中间件，框架是哪种模型就用哪种模型，没有太多选择的余地。

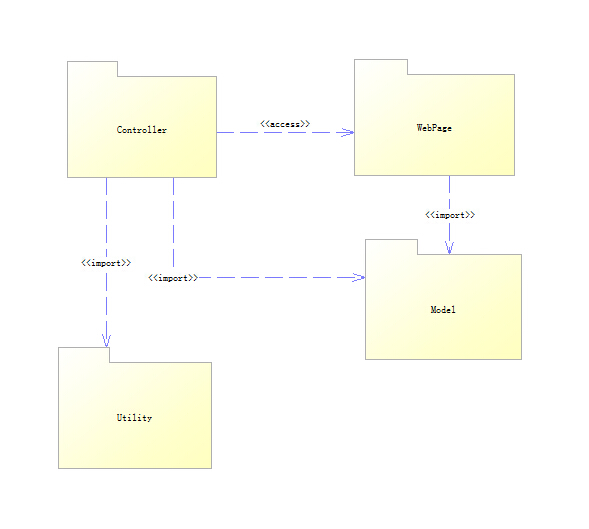
# 逻辑视图

## 类图



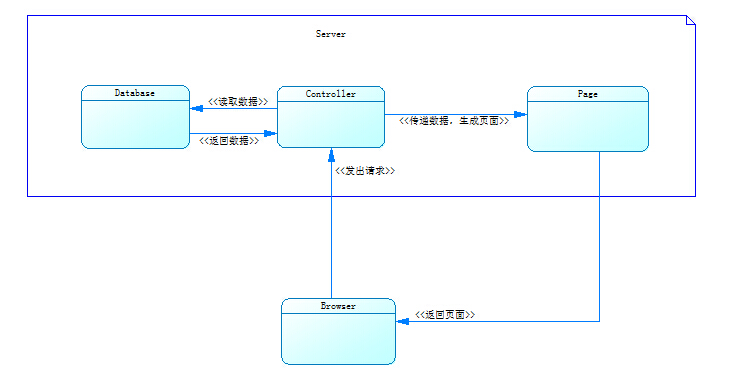
首先，按照功能划分为如下几个页面：主页、登录、注册、个人页、设置、粉丝、关注、消息、后台管理。每个页面配有一个Servelet类和一个Page。Info后缀的是存放数据实体的类。

## 包图

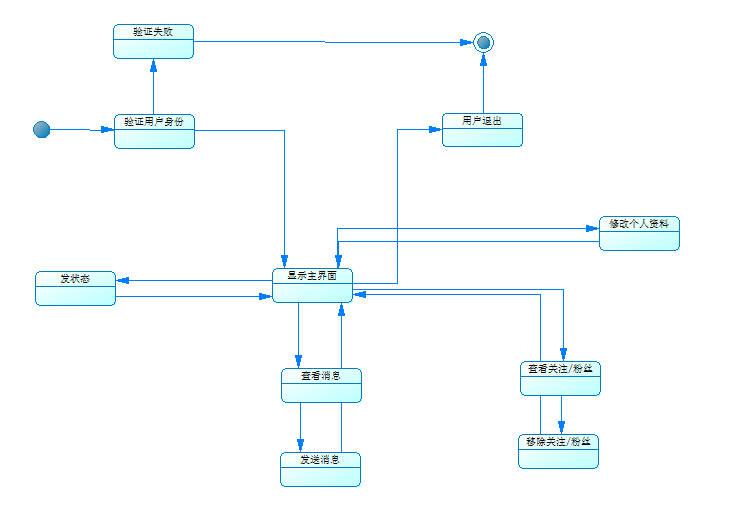


其中，Sevlet存放于Controller中，Page存放于WebPage中，Info存放于Model中。Utility中是一些功能类，比如编码解码。

# 进程视图

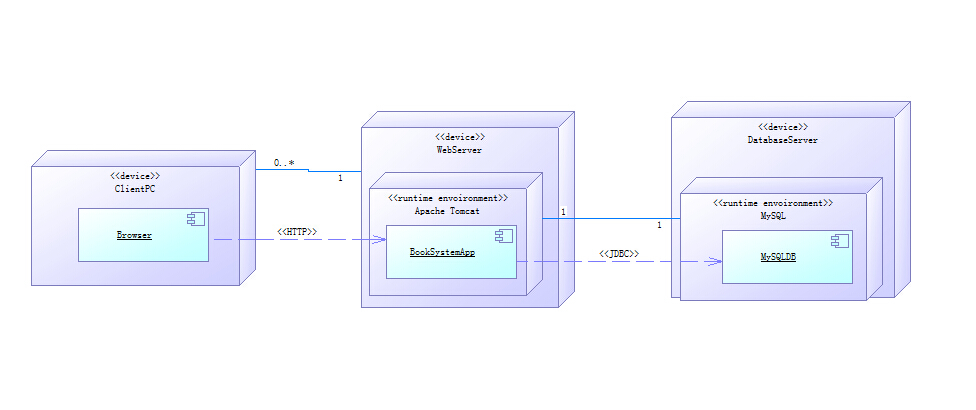


这是在一次请求之内的视图。当用户访问一个页面时，服务器会创建相应Controller类的对象，这个对象从数据库读取数据后，封装为一些数据结构，通过request传递给相应的Page。Page生成页面后返回给浏览器。



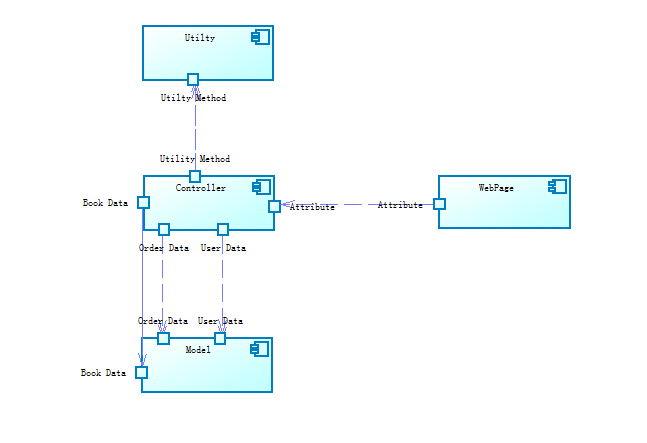
这是用户在线上的操作轨迹。由于http请求不是长连接，其中每一个操作包含一次完整的请求。

# 部署视图



其中，WebServer和DatabaseServe可以是一台服务器，也可以是一个集群，视用户数量而定。

# 开发视图



各个名字的含义和包图中相同。

# 监控与恢复

对于集群中的每台主机，采用心跳的方式进行监控。即被监测的服务器每隔一段时间，向监控服务器发心跳包，内容带上时间即可。如果没有收到某个服务器的消息，可以推算出大致是什么时候挂的。

监控一共有三种方式，除了心跳之外，有一种方案是被监控的主机在发生故障时向监控服务器发消息，但是故障可能很严重而导致消息并不能够发出。还有一种方案叫做ping或者echo，即监控用的主机定时向被监控者发请求，若收到请求则说明正常。但这种方案需要正在工作的服务器监听额外的请求，无疑增加了它的负载。综合各种因素，最后选择了心跳的方式。

该系统的数据采用被动冗余的方式进行备份。被动冗余指的是，服务器各节点之间不主动进行主备切换，而是由客户端来试，直到找到一个发送成功的节点，这个节点就自动成为主要节点。由于SNS从本质上来说可靠性要求不是很强，故采用这种策略。

# 可靠性

我们采用如下方式来提升它的可靠性：

（1）集群之间共享资源。

（2）限制单位时间请求次数，限制单位时间网络流量，限制单位时间内数据库连接次数，限制单位时间并发连接数。

（3）设置优先级。优先级相同的任务先创建先处理。

# 安全

经过我们在web开发方面的经验，采用如下方式来提升安全性：

（1）对所有具有格式的输入参数进行检验。

（2）对于所有可能带有html实体的输出参数，在page层进行转义。

（3）使用orm框架来防止sql注入。

（4）关键操作，比如添加删除等，设置并检验token来避免csrf。

（5）在所有页面的controller层，首先检验用户的权限。

（6）用户的密码用sha1或sha256加密后在数据库中存储。

（7）限制单ip24小时内的最大注册用户数。

# 可修改性

一般情况下，我们只是改动接口里面的逻辑，如果一定要改动接口，会采用增加适配器的方式来实现可修改性。适配器方式，实际上是在一个类上面实现两个接口，一新一旧，新的接口函数的实现去调用旧的。我们在开放新的接口同时，并不关闭旧的接口，以保证使用旧接口的客户端正常运行，当然旧接口只是没新接口强大罢了。

# 易用性

在易用性方面，我们会从一下几点来入手：

（1）页面使用合理的布局和颜色搭配。

（2）常使用的链接放置在突出位置，其他链接也不会放置过深。设置一个区域，用户可以自定义他们常使用的链接。

（3）顺应用户的使用习惯，通过参考其他网站，使用页面交互上的一些惯例。

此外，网站的易用性不仅仅是架构方面的事情。用户界面，包括布局和配色和交互方式，这种设计上的东西也影响者易用性。

# 可测试性

我们会考虑为测试人员开放特殊访问接口来增加可测试性。比如，重要操作前后都要记录日志，然后开放接口让测试人员可以按需访问日志的某个部分。也可以开放数据库的读权限给测试人员，便于检测数据的增删改查情况。

# 设计模式

## 工厂模式

工厂模式提供了一个接口，可以根据传进去的参数返回相应的对象。

假设我的社交系统打算面向世界，就要有一个国际化的策略。我的策略是这样，所有要进行语言转换的词，都有一个id编号，读取用户的语言设置后，加载相应的字典，用id在字典中取到相应的词后，写到页面上。获取字典的接口可以设计成工厂模式，传进去表示语言的字符串，返回相应的字典。

## 装饰器模式

装饰器模式相当于给被装饰的对象做外包装，同一个对象可以表现出不同的效果。

我的社交系统可能会根据访问者权限的不同，展示用户资料的完整度不同，比如，管理员可以看到全部资料，包括邮箱和手机等敏感信息，好友可以看到大部分资料，陌生人只能看到必要的信息。再比如，博客的文章，需要根据情景的不同（手机或者PC端，首页或者内页）加载不同详细程度的正文内容。这种情况会用到装饰器模式。

## 构造器模式

构造器模式用于简化对象的构造操作，当一个对象的构造器中，参数特别多的时候，就应该改为构造器模式进行构造。

我的社交系统带有按条件搜索指定用户的功能。搜索条件包括昵称、年龄、城市、工作教育经历，等等一系列用户设置的信息。搜索的时候需要构造一个SearchCondition对象，如果直接在构造器中传入所有参数，会显得很繁琐，所以这里采用构造器模式。

## 桥接模式

桥接模式将类的继承关系简化为组合关系。这样的好处是，当类有多个属性时，所需要创建类的数目由相乘的数量，减少为相加的数量。比如，手机有品牌和功能两个属性，假设有n个品牌，m中功能，若是按照继承关系来构建类，需要有mn个类；但采用桥接模式后，将品牌和功能变为手机的属性，只需要有（m + n）个类，编码量减少，代码复用程度提高，而且每个属性的更换和组合会更加灵活。

在社交系统中，用户的属性之一是权限，当然它还有很多属性。权限这个属性将封装成一个类，和用户类组合，而不是由用户类派生出管理员和版主类。这个地方应用了桥接模式。

## 迭代器模式

迭代器模式将各种奇奇怪怪的数据结构（比如说树和图）统一为线性操作。这样调用者可以在不知道具体数据结构及其相关知识的情况下，遍历整个结构。

为了便于操作，数据库要做or映射，返回值是一个装有查询结果的容器，这个容器将采用迭代器模式，可以用迭代器进行访问。

## 适配器模式

适配器模式可以将一组接口转化为另一组接口，常常用于不同版本的接口间的适配。

我的社交系统为了方便各种客户端（PC、移动端）的访问，设置了不同协议的HTTP接口。而在控制层内部，为了代码复用会把业务逻辑与控制逻辑分离，单独写在一些类里面。这些业务逻辑类，会提供一些接口，比如传入用户信息、标题和内容调用AddArticle来发布博文。假设我的系统开始没有移动端，后来新加上去，然而移动端需要上传一个值来表示它的种类（ios、安卓、wp），我就需要开一个新的接口，但是我应用了适配器模式之后，就不需要完全重写业务逻辑。我只需要把新的接口适配到旧接口上面，之后额外处理用户端种类就好。

## 抽象工厂模式、单例模式

在一个web系统里面，经常要连接数据库。连接数据库的语句不只一步，而且连接的时候要传很多参数，比如用户名、密码、主机。这些参数是从配置文件里面获得的，每次都要把get方法写一遍很麻烦。这些东西可以包装在一个工厂类里面，工厂类提供一个getConn的静态方法用来获取数据库连接。还可以应用单例模式，在每次调用的时候，检查数据库连接的状态，如果当前进程已有一个连接，就直接把内置的连接返回，不额外新建连接。

## 模板方法模式

在我的社交系统中，专门有一个类，根据加载进来的参数，渲染html页面。但是用户常常有更换不同皮肤的需求，不同皮肤实际上只是css和一些装饰用的部件的不同，大体上还是一致的。于是我把整个页面切成若干个部分，分别由若干个函数进行渲染。如果要更换皮肤，一个新的皮肤对应一个派生类，这个类可以覆盖掉原始皮肤类的方法，实现自己的逻辑；也可以新增加一些方法，渲染出新的部件，最后按自己的方式组合。

## 解释器模式

解释器模式用于解释某一种脚本语言。解释器模式的难点不在于创建类本身，而在于词法解析的算法部分。由于技术难度太大，而且我觉得没有必要为这个社交网站单独设计一门语言，所以自己编写的类里面没有用到解释器模式。不过，社交网站还是大量调用了带有解释器模式的库，比如正则表达式。在控制层的最开始，我们读进参数后要验证参数的合法性，这个位置大量用到了正则表达式。

## 中介者模式

中介者模式常常用于GUI程序的绑定回调中。比如，一个按钮的回调函数是设置一个文本框的内容，我们通常的做法是在窗体类里面写回调，然后在这个按钮的点击事件上面绑定个带有窗体对象的闭包，闭包里面调用回调函数。而不是直接把文本框这个对象传给按钮，然后按钮继承出一个类来改写回调内容。这样减少了控件间的耦合程度。

由于web应用并不处理控件，中介者模式在这个web应用本身不适用。但是如果我为我的社交系统编写一个移动端的app，里面绑定回调的操作会大量应用中介者模式。

## 备忘录模式

备忘录模式是指创建一个类来保存历史操作。这个模式在游戏的存档读档撤销功能里非常常见。

现在很多网站，比如知乎、wordpress都拥有查看历史记录的功能，我的社交网站也不例外。在用户请求查看历史记录的时候，从数据库加载历史记录，并存到备忘录的类里面，刷到页面上给用户展示。之后，用户可以根据历史记录的id请求恢复到某个版本。