**武汉大学计算机学院**

**本科生实验报告**

**软件设计与体系结构大作业**

专 业 名 称 ：计算机科学与技术

课 程 名 称 ：软件设计与体系结构

指 导 教 师 ：王翀

学 生 学 号 ：2018302110269

2018302110024

2018302110083

学 生 姓 名 ：徐翔宇

叶博辰

李昊洋

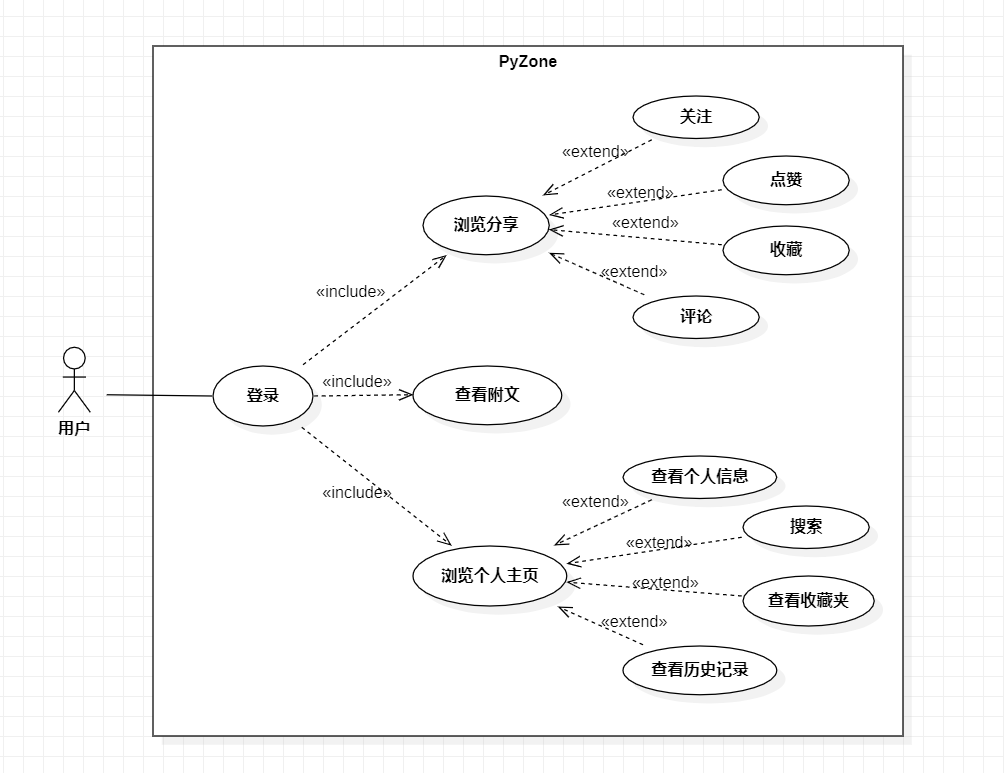
二○二○年十二月

# 1 实验一

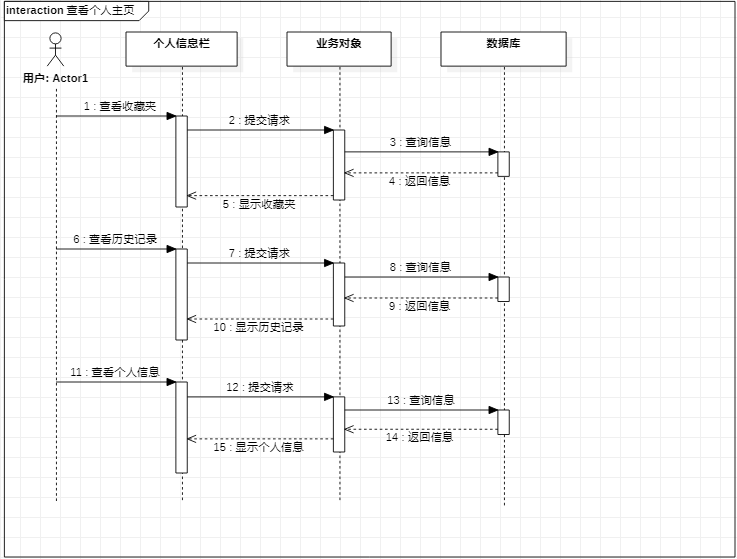
## 1.1 需求分析

派哟编程拼图是一个儿童编程教学系统，包括拼插式的实体积木和配套的微信小程序。 小程序内设有社区。用户可以在社区内发布自己的作品，浏览其他用户的作品，并进行收藏、点赞和评论。用户可以在个人中心查看自己的作品、收藏夹和点赞记录。

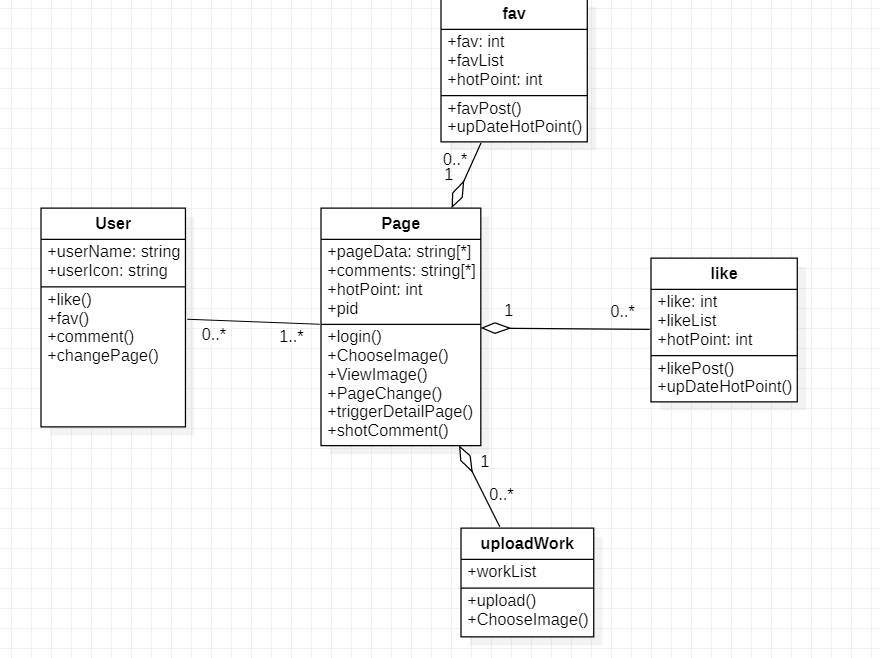
## 1.2 用例图



## 1.3 顺序图



## 1.4 类图



# 2 实验二

## 2.1 应用观察者模式

在前端我们使用了观察者模式。因为中间的数据是有关联的（以热度为例），当与页面相关的参数发生变化的时候（例如点赞，收藏）的时候，就要对这些数据进行监听，然后回调更新函数，来更新热度的计算结果。

#### 2.1.1 应用场景

每一个涉及到收藏，点赞数更新的函数后面都跟有一个“upDateHotPoint()”，这个函数会读取收藏数和点赞数并且计算相应的热门值，进行更新。但是如果出现忘记打这一行代码的情况，就会导致数值更新了，热度值没有更新。

启动的时候给数据中的pageData增加监听，当里面发生变化的时候，调用watch中间的函数进行更新，核心代码如下：

onLoad: function () {

this.loadPages();//加载新页面

getApp().setWatcher(this); // 设置监听器

},

watch: {

pageData: function (newPage) {//当pageData变化的时候，调用监听函数

newPage.forEach(e => {

e.hotPoint = 0;

if (e.fav > 5) {

e.hotPoint += e.fav \* 6

} else {

e.hotPoint += e.fav \* 12

}

e.hotPoint += e.like \* 3;

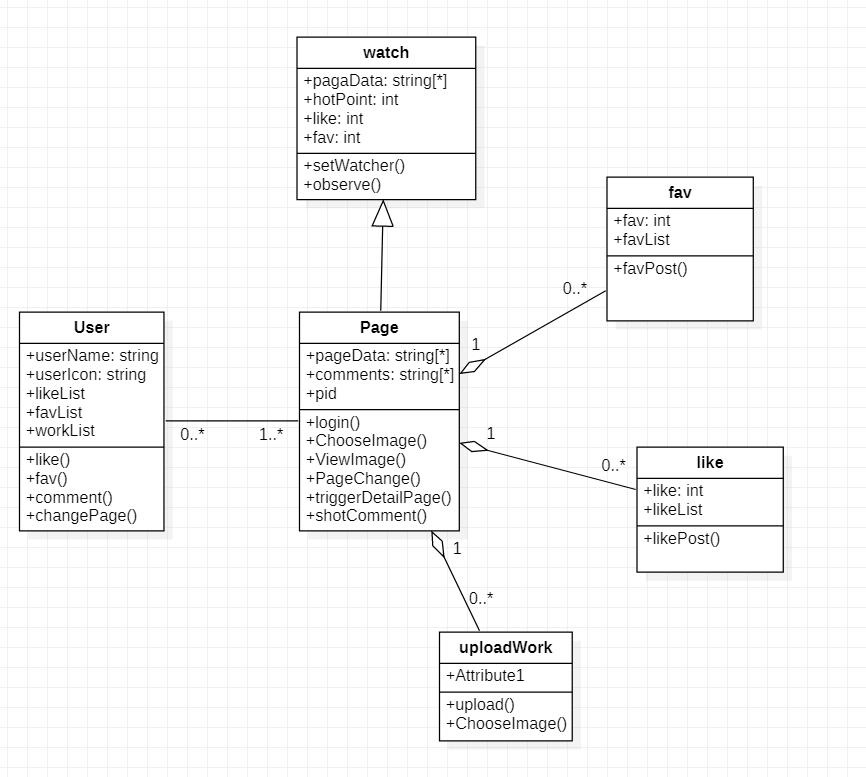
})

this.setData({

pageData: newPage//更新热度值

});

}

},

2.1 应用观察者模式后前端类图

#### 2.1.2 结果分析

使用了观察者模式后大大减少了代码的冗余，让各个地方的更新函数都可以回调监听器函数，并且只在值变化的时候进行更新，提高了运行效率。

## 2.2 应用工厂模式

#### 2.2.1 应用场景

在后端中，使用了基于SpringBoot的框架与前端进行对接。在建立连接/执行sql语句的时候，我们需要建立一个链接类，对于sql语句也是，我们要获得一个语句对象。

因此这里使用了工厂模式，用springIoc以及Mybatis的mapping自动的获得实例，进行注入，这样，就不用进行复杂的新建和初始化，可以减少代码的复杂量：

使用之前，每次需要建立数据库连接，都需要：

Connection dbConn=DriverManager.getConnection(dbURL,userName,userPwd);

来获得连接实例

使用之后

@Autowired

public UserController(postMapper postMapper){

this.postMapper = postMapper;

}

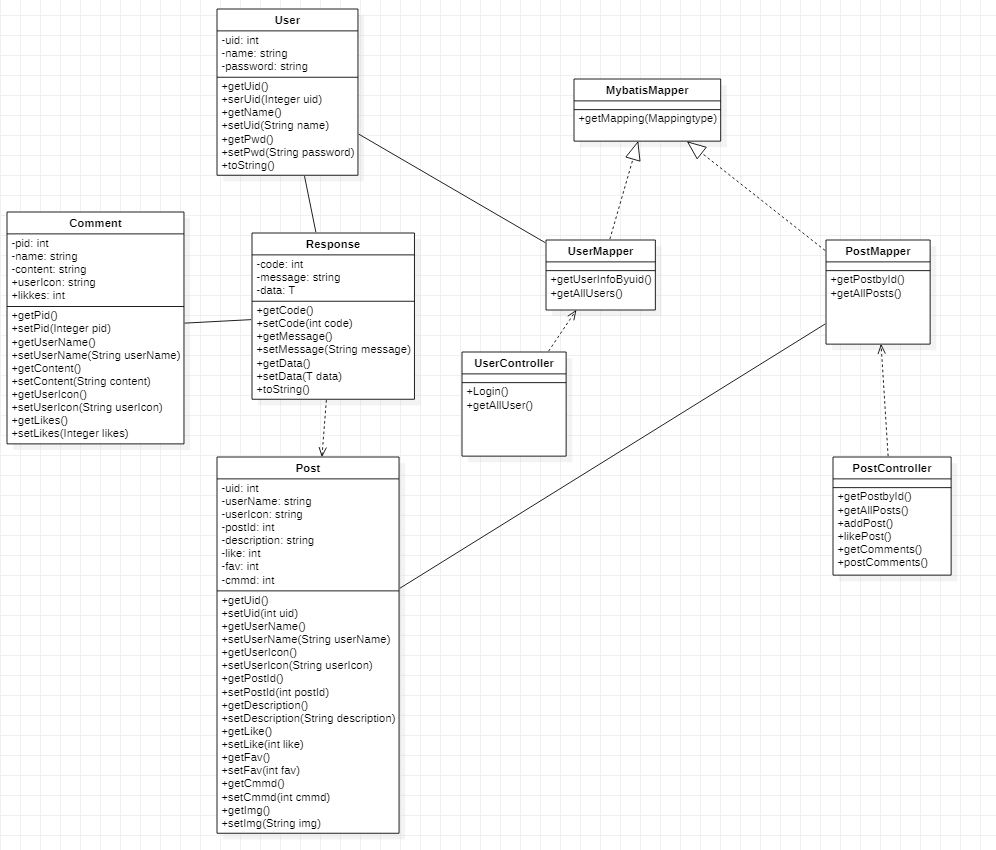
private postMapper postMapper;

在这里，我通过java注解的方式，用Mybatis创建的简单工厂获得一个数据库链接实例，并且在接下来的使用中，只需要调用里面的函数就可以执行sql语句，例如：

List<Post> posts = postMapper.getAllPosts();对应着

<select id="getAllPosts" resultType="com.migu.tntsb.tntsb.domain.Post">

select \* from posts

</select>

2.2 应用工厂模式后后端类图

#### 2.2.2 结果分析

工厂模式是spring的核心优势，善用这种设计模式可以减少重复的代码，让实例的获取和管理更加简单方便。

# 3 实验三

## 3.1 功能

使得其他开发者可以通过调用本API，获取教主小宇宙的博客上的博文、评论，以及相关的时间戳等信息。

## 3.2 接口

#### 3.2.1访问

通过HTTP GET进行数据获取，通过URL传递需要获取的数据类型。

#### 3.2.2返回

回传JSON，包含访问状态（成功或失败），访问的对象类型，以及访问的对象body。body通过UTF-8编码。 当访问成功时，body为博文的正文。在访问失败时，body为错误原因。

###### 3.2.2.1JSON格式

{

"status": "success",

"requested\_object": "post",

"body": "hello world"

}

#### 3.2.3访问地址

|  |  |
| --- | --- |
| 功能 | 地址 |
| 博文 | <https://eture.tech/apocal/api/post/> |
| 评论 | <https://eture.tech/apocal/api/emotion/> |

## 3.3实现

提取URL当中的requested\_object

# urls.py

from django.urls import path

# ......

from . import views

urlpatterns = [

# ......

path('api/<str:requested\_object>/', views.api, name='api'),

]

对requested\_object进行合法性判断

# views.py

# ......

@csrf\_exempt

def api(request, requested\_object):

# ......

if requested\_object == 'post':

# ......

if requested\_object == 'emotion':

# ......

# ......

对合法地request从数据库中查询相对应的object

Emotion.objects.filter(happy=True, message\_\_isnull=False))

进行序列化

response['body'] = serializers.serialize('xml', Emotion.objects.filter(happy=True, message\_\_isnull=False))

封装成JSON

response = {'status': 'failed', 'requested\_object': requested\_object, 'body': ''}

# ......

json.dumps(response)

HTTP返回

return HttpResponse(json.dumps(response))