

“易聊”软件设计说明书

“易聊”软件设计说明书

- 1 引言
 - 1.1 背景
 - 1.2 参考资料
- 2 设计概述
 - 2.1 任务和目标
 - 2.1.1 需求概述
 - 2.1.2 运行环境概述
 - 2.1.3 条件与限制
 - 2.1.4 详细设计方法和工具
- 3 模块设计
- 4 项目结构
- 5 流程逻辑
 - 整体用例
 - 各用例活动图
- 4 接口定义
 - 4.1 通用枚举类说明
 - 4.1 接口会使用的类说明
 - 4.1.1 接口使用的实体类说明
 - 4.1.2 接口使用的统一响应类
 - 4.2接口说明
 - 4.3 netty的handler说明
- 5 框架说明和代码设计
 - 框架的设计
 - java反射机制
 - Spring
 - Bean
 - 控制反转(IoC)和依赖注入(DI)
 - Netty
 - 责任链模式
 - Mybatis
 - 对象关系映射(ORM)
 - 自己的设计
 - 1.
 - 2.
- 6 用户界面设计
 - 5.3 系统界面详细设计
 - 5.3.1 外部界面设计
 - 5.3.2 内部界面设计
 - 5.3.3 用户界面设计
- 6、数据库系统设计
 - 6.1设计要求
 - 6.2 信息模型设计
 - 6.3 数据库设计
 - 6.3.1 设计依据
 - 6.3.2 数据库种类及特点
 - 6.3.3 数据库逻辑结构
 - 6.3.4 物理结构设计
 - 6.3.5 数据库安全
 - 6.3.6 数据字典

1 引言

1.1 背景

课程作业要求完成一个聊天软件，平台不限，至少要求最基础的聊天功能。本项目内部开发代号为“易聊”。

1.2 参考资料

netty源码

<https://github.com/netty/netty>

openjdk11源码

<https://github.com/AdoptOpenJDK/openjdk-jdk11u>

tiny-spring源码

<https://github.com/code4craft/tiny-spring>

2 设计概述

2.1 任务和目标

预计完成好友功能，文字图片聊天，视频聊天功能。

2.1.1 需求概述

业务需求见架构设计说明书，本系统的性能需求要求能够承担每秒1万次请求（注：由于没有实际项目经验，参考了csdn对netty的一篇测试），可以看到阿里云8核16G的服务器能承担三万左右并发，由于本系统的部署机为2核4G，预估性能瓶颈为每秒1万次请求。

2.1.2 运行环境概述

本系统运行在2核4G的腾讯云服务器上，硬盘大小为50G。操作系统采用ubuntu18.04，数据库使用mysql，使用redis缓存，使用fastDFS文件服务器管理文件。

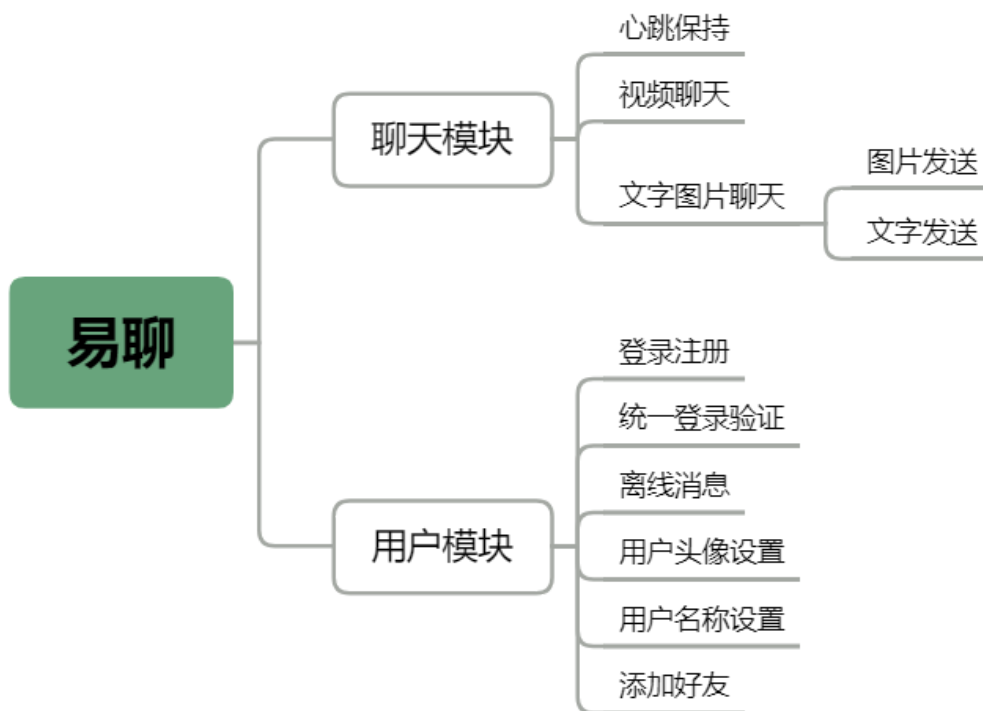
2.1.3 条件与限制

详细描述系统所受的内部和外部条件的约束和限制说明。包括业务和技术方面的条件与限制以及进度、管理等方面的限制。

2.1.4 详细设计方法和工具

简要说明详细设计所采用的方法和使用的工具。如HIPO图方法、IDEF (I2DEF) 方法、E - R图，数据流程图、业务流程图、选用的CASE工具等，尽量采用标准规范和辅助工具。

3 模块设计

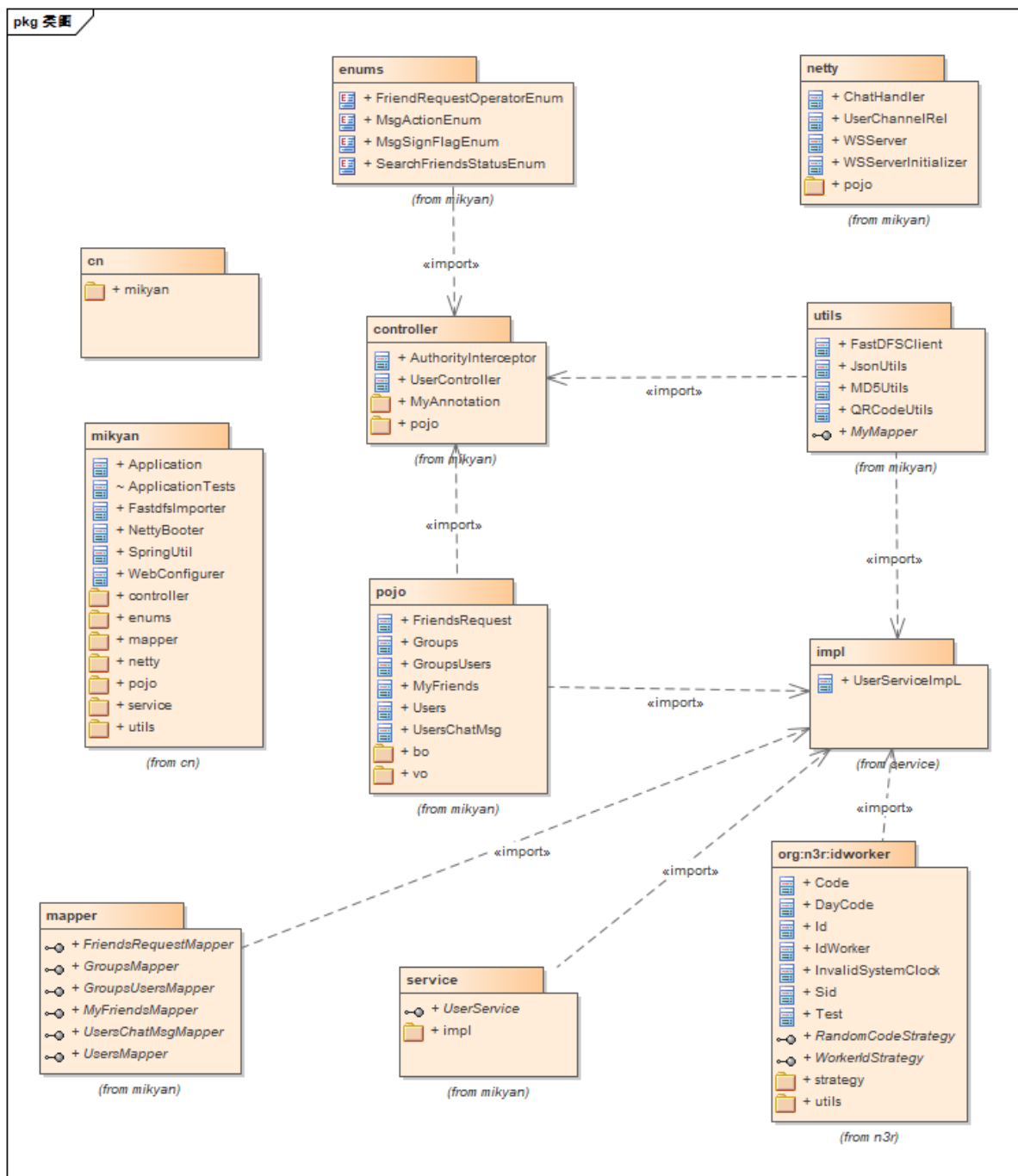


如图，系统划分为聊天模块和用户管理模块，聊天模块与用户管理模块功能重叠比较小，使用的技术也不同，开发时也很方便并行开发。

聊天模块中，文字图片聊天，视频发送的ws链接部分，心跳保持模块，分别是netty的管道模式（责任链模式）上的3个valve（责任链的处理单元），可以分开编写，并行开发。视频聊天模块中使用websocket的部分也可以作为一个valve来实现，webRTC部分则需要另外实现。

用户模块中，离线消息作为用户模块的一部分似乎并不合理，但是由于离线消息是采用拉取的方式，而不是推送的方式，所以在编码是可以放在用户的controller里的。统一登录验证采用基于注解的拦截器模式，通过在springboot框架下自定义注解，然后在需要登录的接口前添加@LoginRequired注解即可。其他均为用户路由的接口。














4 项目结构



注意，本图中没有画出部分不涉及到引用关系的子图

pkg 类图

在包cn.mikyan中的元素列表

	Name	Scope	Status	Author
	Application	Public	提议的	MIKYAN
	ApplicationTe...	Packa...	提议的	16776
	controller	Public	提议的	16776
	enums	Public	提议的	16776
	FastdfsImporter	Public	提议的	tobato
	mapper	Public	提议的	16776
	netty	Public	提议的	16776
	NettyBooter	Public	提议的	16776
	pojo	Public	提议的	16776
	service	Public	提议的	16776
	SpringUtil	Public	提议的	16776
	utils	Public	提议的	16776
	WebConfigurer	Public	提议的	BIANP

显示 1 - 13 of 13 项

项目的根目录为cn.mikyan，同时我们在外部引入了包org:n3r这个包，用于生成字符串型的id。下面主要介绍cn.mikyan包下的子包功能

cn.mikyan包下的类包括启动类和配置类，用于项目的启动和配置。utils包是全局工具类，需要的通用工具都包含在这个包里。service包定义了后端的主要服务，所有的后端服务都在这个包里实现。controller包利用service包内提供的服务，编写接口，负责与前端的交互。enums包下编写全局枚举量，主要用于controller类。pojo下编写全局实体类，主要保存交互对象，例如消息对象，用户对象等。mapper是ORM(对象关系模型)中的对象所在包。netty包负责所有websocket相关的处理，主要聊天功能都在这里实现。

5 流程逻辑

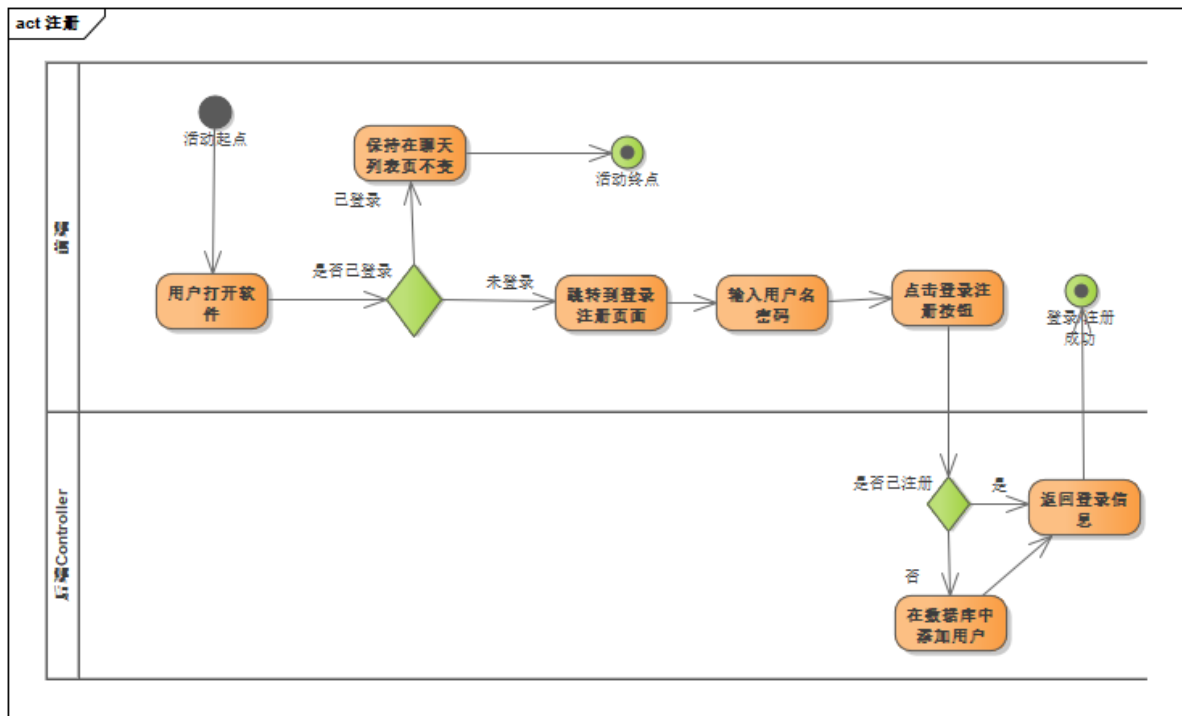
整体用例



请注意我们本次的消息设计说明书细化了用例，将图片聊天这一用例从文字聊天内独立了出来，方便说明，同时增加了修改个人信息的用例。

各用例活动图

注册登录



在前端，登录注册并做一个按钮，后端也是写在一起，所以这里只画一个活动图，首先，用户打开软件，然后软件判断用户是否已登录，如果已经登录则保持打开的chatlist，否则跳转到登陆注册页面，然后用户输入账户密码，点击登录注册按钮，数据被传回后端，后端直接判断是否已注册，未注册则直接注册，已注册则登录。

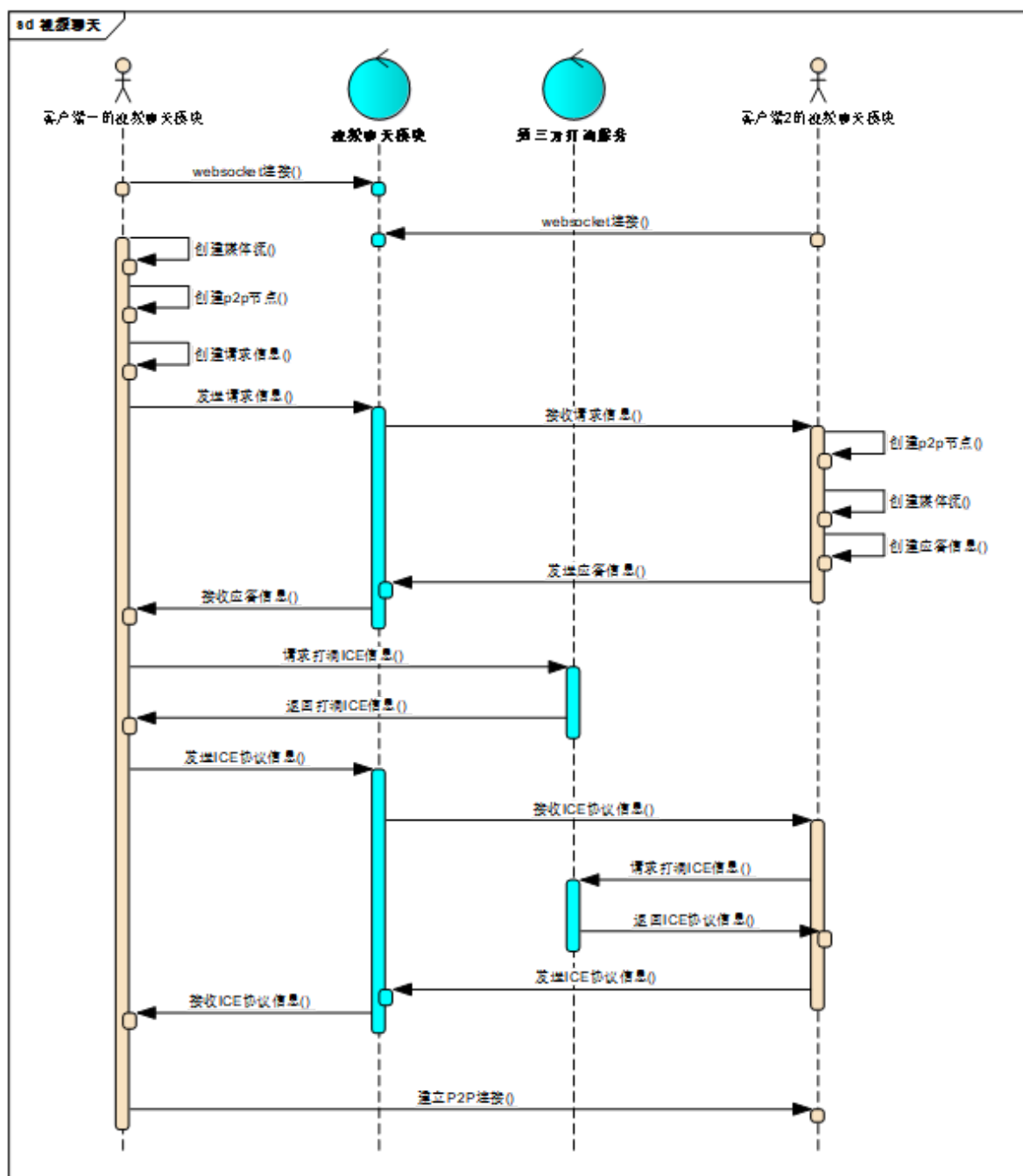
添加好友

与好友文字聊天

与好友图片聊天

与好友视频聊天

本流程图与架构设计中的协作图一章中相同



首先，客户端一二要保证与服务端建立了ws连接，然后由客户端一（发起视频聊天的一方）建立媒体流和本地的p2p节点，然后把描述信息发送给后端，后端再发送给客户端2，客户端2接收到后，进行与客户端1相同的操作，然后客户端一向打洞服务器，请求自己的IP地址，这里使用的是ICE协议，打洞服务器把相应的信息返回，客户端一再把这个信息发送给客户端2，客户端2也如此操作，然后双方就建立了p2p连接，可以进行视频通话。

修改个人信息

4 接口定义

代码框架已经搭建好，请开发人员结合代码框架和本部分，编写程序

4.1 通用枚举类说明

只保留枚举量的定义，未写出方法，方法请参照具体的代码

```
1  /**
2   * @Description:忽略或通过好友请求的枚举
3   */
4  public enum FriendRequestOperatorEnum {
5      IGNORE(0, "忽略"),
6      PASS(1, "通过");
7  }
8
9
10 /**
11  *
12  * @Description: 发送消息的动作 枚举
13  */
14
15 public enum MsgActionEnum {
16
17     CONNECT(1, "第一次(或重连)初始化连接"),
18     CHAT(2, "聊天消息"),
19     SIGNED(3, "消息签收"),
20     KEEPALIVE(4, "客户端保持心跳"),
21     PULL_FRIEND(5, "拉取好友"),
22     Friend_Request(6, "发送好友请求"),
23     CHAT_IMAGE(7, "消息是图片");
24 }
25
26 package cn.mikyan.enums;
27
28 /**
29  * @Description: 消息签收状态的枚举
30  */
31 public enum MsgSignFlagEnum {
32
33     UNSIGN(0, "未签收"),
34     SIGNED(1, "已签收");
35
36 }
37
38 package cn.mikyan.enums;
39
40
41 /**
42  * @Description:添加好友的状态
43  */
44 public enum SearchFriendsStatusEnum {
45     SUCCESS(0, "OK"),
46     NOT_EXIST(1, "无此用户"),
47     NOT_YOURSELF(2, "不能添加自己"),
48     ALREADY_FRIENDS(3, "不能重复添加好友");
49 }
```

4.1 接口会使用的类说明

4.1.1 接口使用的实体类说明

实体类只有属性和get, set方法, 所以下面只列出属性, 省去get, set方法

```
1  HttpServletResponse response; //http的响应体, 仅在登录注册中会用到
2
3  public class UsersBO {
4      private String userId;
5      private String faceData;
6      private String nickname;
7  } // 前端回传用户昵称等信息时使用的实体类
8
9  public class UsersVO {
10     private String id;
11     private String username;
12     private String faceImage;
13     private String faceImageBig;
14     private String nickname;
15     private String qrcode;
16 } //后端发送用户信息时用的实体类
17
18 package cn.mikyan.netty.pojo;
19 public class ChatMsg implements Serializable {
20
21     private static final long serialVersionUID = 1L;
22
23     private String senderId; // 发送者的用户id
24     private String receiverId; // 接受者的用户id
25     private String msg; // 聊天内容
26     private String msgId; // 用于消息的签收
27     private Integer type; // 用于判断消息类型
28
29 } //用于前后端传输消息的实体类
30
31 package cn.mikyan.pojo;
32 public class UsersChatMsg {
33
34     private String id;
35     private String sendUserId;
36     private String acceptUserId;
37     private Integer type;
38     private String msg;
39     private Boolean signFlag;
40     private Date createTime;
41 } //用于后端储存消息到数据库的实体类
42
43 package cn.mikyan.pojo;
44
45 import javax.persistence.*;
46
47 public class Users {
48     private String id;
49     private String username;
50     private String password;
51     private String faceimage;
52     private String faceimageBig;
53     private String nickName;
54     private String qrcode;
55     private String cid;
```

```

56     private String appid;
57     private String appkey;
58 }//主要用于登录注册时前端返回用户信息，并用于用户信息的储存
59
60
61
62 public class FriendRequestVO {
63
64     private String sendUserId;
65     private String sendUsername;
66     private String sendFaceImage;
67     private String sendNickname;
68 }//后端发送到前端的好友请求信息

```

4.1.2 接口使用的统一响应类

未标出getter和setter方法

```

1
2  /**
3   * @Description: 自定义响应数据结构
4   *              本类的作用是后端接口响应的统一结构题，如果需要返回信息，则存在该实体类
   的data字段里
5   *              前端接受此类数据后需要使用本类的方法转换成对于的数据类型格式（类，或者
   list）
6   *              200: 表示成功
7   *              500: 表示错误，错误信息在msg字段中
8   *              501: bean验证错误，不管多少个错误都以map形式返回
9   *              502: 拦截器拦截到用户token出错
10  *              555: 异常抛出信息
11  */
12
13 public class ResponseJSON {
14     // 响应业务状态
15     private Integer status;
16
17     // 响应消息
18     private String msg;
19
20     // 响应中的数据
21     private Object data;
22
23     public static ResponseJSON build(Integer status, String msg, Object
   data) {
24         return new ResponseJSON(status, msg, data);
25     }
26
27     //返回正确，并携带数据体，如返回好友信息等
28     public static ResponseJSON ok(Object data) {
29         return new ResponseJSON(data);
30     }
31
32     //返回正确，不携带数据体，如返回拒绝好友请求
33     public static ResponseJSON ok() {
34         return new ResponseJSON(null);
35     }
36

```

```

37 //返回错误和错误信息
38 public static ResponseJSON errorMsg(String msg) {
39     return new ResponseJSON(500, msg, null);
40 }
41
42 //返回bean验证错误
43 public static ResponseJSON errorMap(Object data) {
44     return new ResponseJSON(501, "error", data);
45 }
46
47 //返回token错误
48 public static ResponseJSON errorTokenMsg(String msg) {
49     return new ResponseJSON(502, msg, null);
50 }
51
52 //返回异常
53 public static ResponseJSON exception(String msg) {
54     return new ResponseJSON(555, msg, null);
55 }
56
57 public ResponseJSON() {
58
59 }
60
61 public ResponseJSON(Integer status, String msg, Object data) {
62     this.status = status;
63     this.msg = msg;
64     this.data = data;
65 }
66
67 public ResponseJSON(Object data) {
68     this.status = 200;
69     this.msg = "OK";
70     this.data = data;
71 }
72
73 public Boolean isOK() {
74     return this.status == 200;
75 }
76 }

```

4.2接口说明

api遵守RESTful规则，第一版api统一使用v1/作为地址第一层，用户路由使用u做为地址第二层

所以用户接口的api命名规范为 v1/u/"api名称"。下面说明各api的功能，输入，输出。

由于api由统一前缀，所以下面的表格只写api名称

由于api统一返回ResponseJSON，所以输出部分只写返回的data字段,以下均使用POST协议

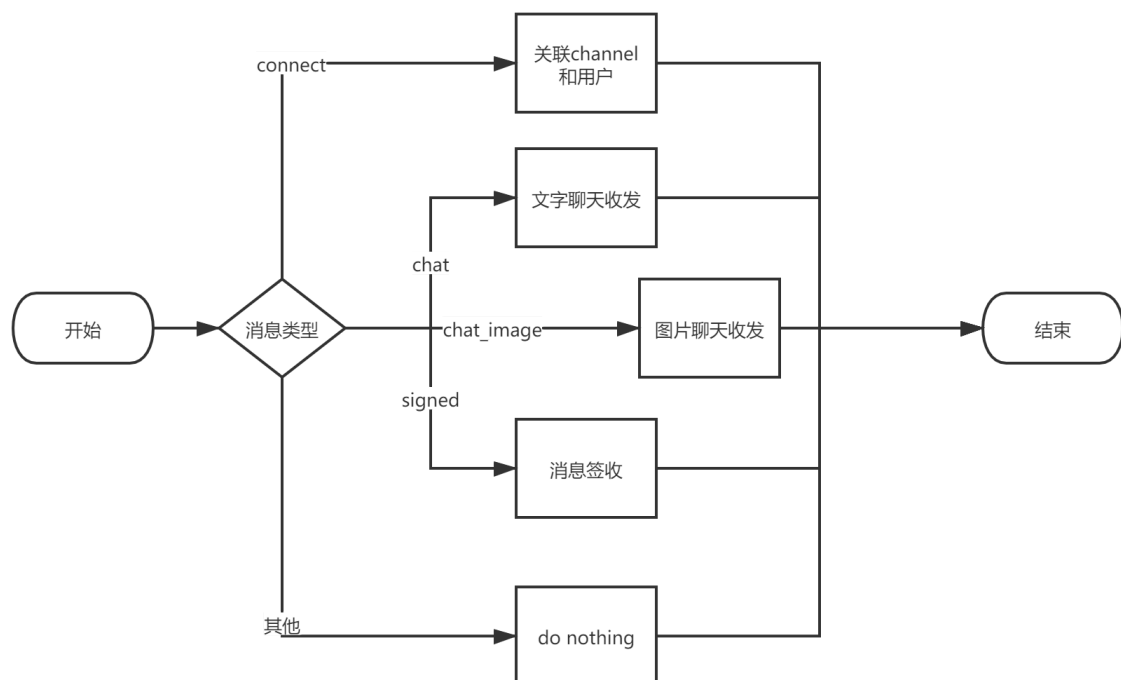
@LoginRequired 表示需要登录

名称	输入	输出	功能	前置条件
registerOrLogin	HttpServletResponse, Users	UserVO (把token 携带在cookie里)	进行登录或注册	
uploadFaceBase64	UsersBO	UserVO	通过base64方法, 上传头像	@LoginRequired
setNickname	UsersBO	UserVO	设置昵称	@LoginRequired
search	String myUserId, String friendUsername	UserVO	搜索其他用户	@LoginRequired
addFriendRequest	String myUserId, String friendUsername	ResponseJSON.ok()	发送添加好友的请求	@LoginRequired
queryFriendRequests	String userId	List	查询好友请求	@LoginRequired
operatorFriendRequest	String acceptUserId, String sendUserId,Integer operType	ResponseJSON.ok()	接受或忽略好友请求	@LoginRequired
myFriends	String usersid	List	查询好友列表	@LoginRequired
getUnReadMsgList	String acceptUserId	List	拉取离线消息	@LoginRequired

4.3 netty的handler说明

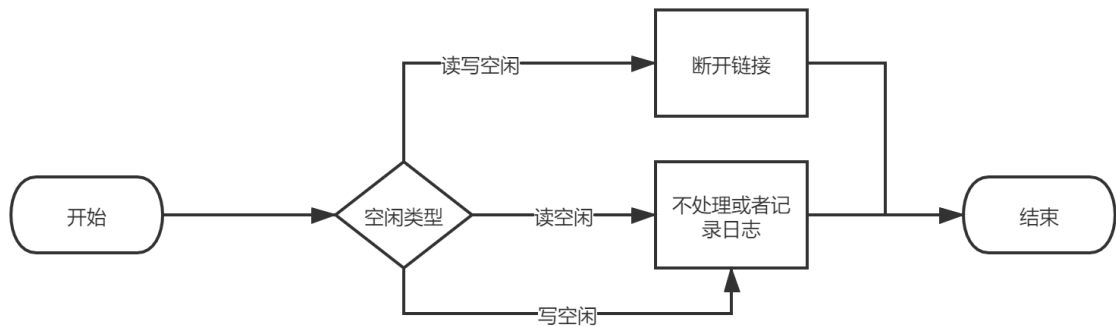
1.ChatHandler

本类，负责聊天的处理，图片和文字均在此handler里处理，同时负责处理建立userid和channel的对应



2. HeartBeatHandler

这个handler逻辑很简单，主要是处理websocket链接的保持和关闭



5 框架说明和代码设计

框架的设计

由于使用类spring+netty+mybatis这样的后端组合，所以很多东西用了框架的模式，是在框架留下的接口上做扩展。所以有必要对用到的设计概念在此进行说明。

java反射机制

java反射机制我们的代码没有直接用到，但是用到的Springboot和Mybatis都依赖于java反射机制，在笔者写后面的时候，越发觉得不讲清楚java反射机制，就很难对Mybatis中ORM的实现有一个好的理解，所以在这里首先写一下，读者可以跳过不看，遇到问题，再返回阅读。

Java反射机制可以让我们在编译期（Compile Time）之外的运行期（Runtime）获得任何一个类的字节码。包括接口、变量、方法等信息。还可以让我们在运行期实例化对象，通过调用get/set方法获取变量的值。

作者：总是擦破皮

链接：<https://www.jianshu.com/p/2315dda64ad2>

来源：简书

此外，你需要了解一些常用的api，方便之后再Mybatis一节中，对源码的理解

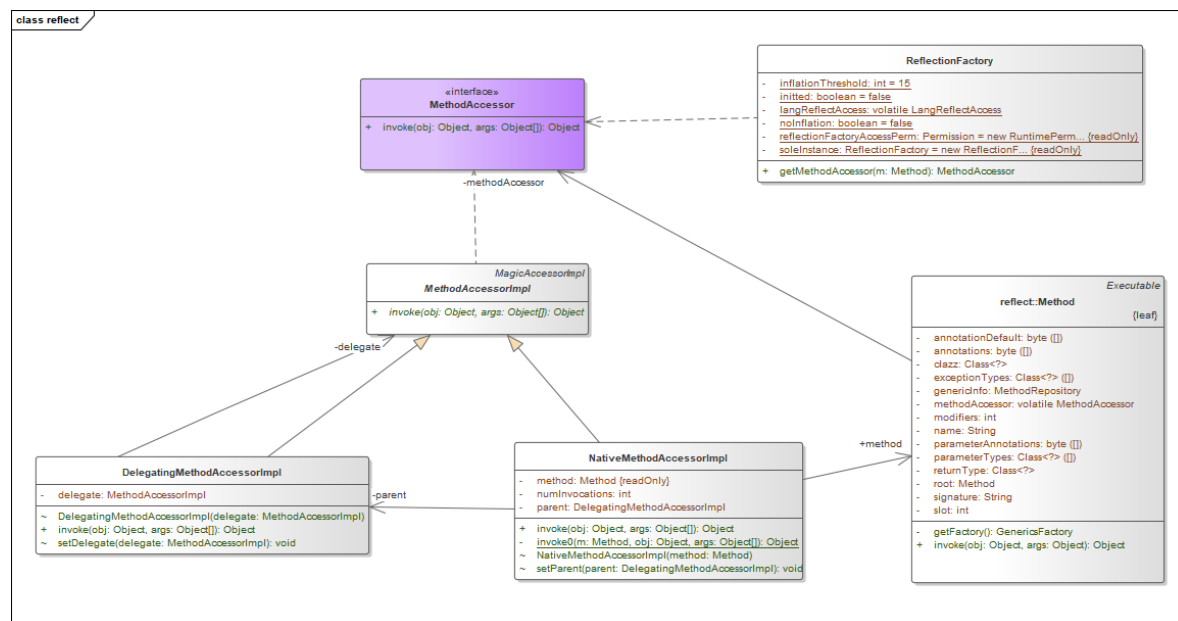
```
1 class clz=Class.forName("cn.mikyan.apple") //可以通过类名来获取一个类的Class实例
2 Constructor appleConstructor = clz.getConstructor();//根据 class 对象实例获取
  Constructor 对象
3 Object appleObj = appleConstructor.newInstance();//通过newInstance可以获得需要的
  类的对象。
4 Apple apple = (Apple)appleConstructor.newInstance();//或者你可以直接做一次强制类
  型转换
5 Apple apple = (Apple)clz.newInstance();//也可以从Class类实例直接获取对象
6 Field[] fields = clz.getFields();//你可以通过getFields获取类的属性
7 Field[] fields = clz.getDeclaredFields();//或是通过这个接口获得所有属性（包括私有
  属性）
8 Method setPriceMethod = clz.getMethod("setPrice", int.class);//通过getMethod方
  法来获取类的方法的Method类
9 setPriceMethod.invoke(appleObj, 14);//通过invoke方法，调用要用的方法
```

反射的实现

首先说，在此我不会把反射的实现从头到尾讲一边，要是这样可以另外专门写一篇教程了，在此我只会讲解反射中的invoke方法的流程。这里面涉及到一个设计模式有趣的应用。

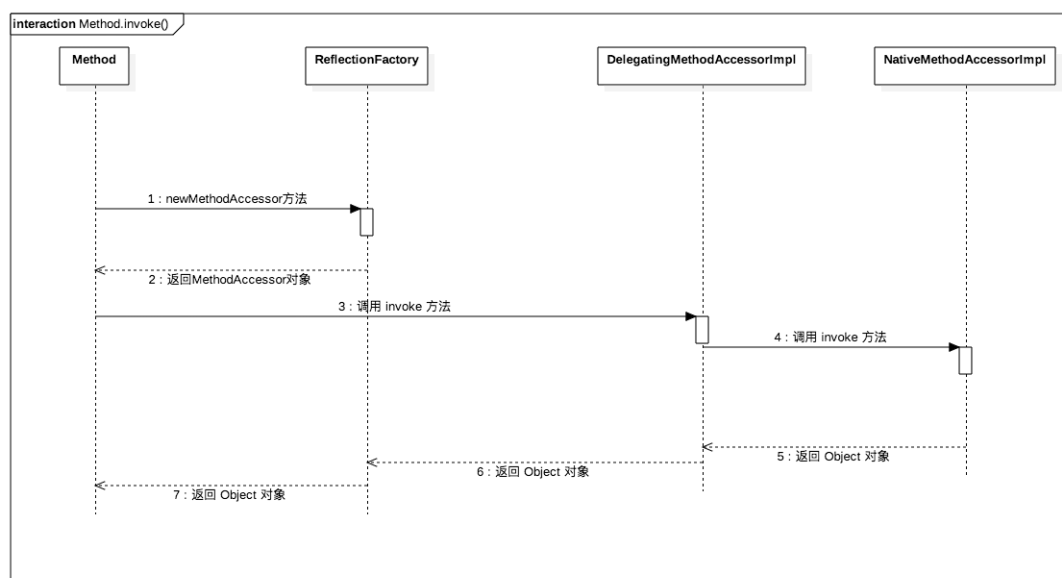
反射的实现用到了代理模式

使用上文中的invoke方法，最终是调用了MethodAccessor对象的invoke方法，而MethodAccessor对象是一个接口，他相关的几个类类图如下



获取MethodAccessor的方法是ReflectionFactory的newMethodAccessor方法，方法会首先盘那段是否存在对象，如果存在则直接使用之前的对象。如果不存在，则会先创建一个NativeMethodAccessorImpl对象，然后用这个对象去生成一个DelegatingMethodAccessorImpl对象，NativeMethodAccessorImpl对象被存在了DelegatingMethodAccessorImpl对象的delegate属性里（名字很长，可以参考类图，看着就很清晰了），最后返回的是DelegatingMethodAccessorImpl对象，而该对象内的invoke是调用了delegate属性的invoke方法（也就是NativeMethodAccessorImpl对象的invoke方法）。所以最后执行的是NativeMethodAccessorImpl的invoke方法。可以参考下面的时序图

(来自https://www.cnblogs.com/chanshuyi/p/head_first_of_reflection.html)



该方法会在invoke调用次数小于15次时，使用本类的invoke0方法。而调用高于15次时，会生成一个新的MethodAccessor对象，并将该对象赋值给DelegatingMethodAccessorImpl对象的，delegate属性。这是因为NativeMethodAccessorImpl的invoke0方法，在运行初期会比较快，但随着调用的增加，速度会变慢，而另一个MethodAccessor对象的invoke方法，启动比较慢，一开是就调用会影响最初的性能，但是随着程序的运行它的速度却会变快，所以会在调用多次之后替换到这个新的MethodAccessor对象。

这很有趣，一般代理模式在使用时，场景是这样的

在直接访问对象时带来的问题，比如说：要访问的对象在远程的机器上。在面向对象系统中，有些对象由于某些原因（比如对象创建开销很大，或者某些操作需要安全控制，或者需要进程外的访问），直接访问会给使用者或者系统结构带来很多麻烦，我们可以在访问此对象时加上一个对此对象的访问层。

但是此处的代理模式，是为了实现类似**策略模式**的效果，通过调换DelegatingMethodAccessorImpl对象内的delegate属性，实现了对invoke方法的替换，至于为什么没有直接使用策略模式，源码太多，暂时还弄不明白，如有兴趣，可自行研究。

Spring

作为开发的核心，有必要对其进行最基本的理解，所以我在这里会对几个核心概念做出解释，并为开发成员提供一些链接，作为扩展阅读。

Bean

为了理解后面的IoC和DI我们需要先了解Bean

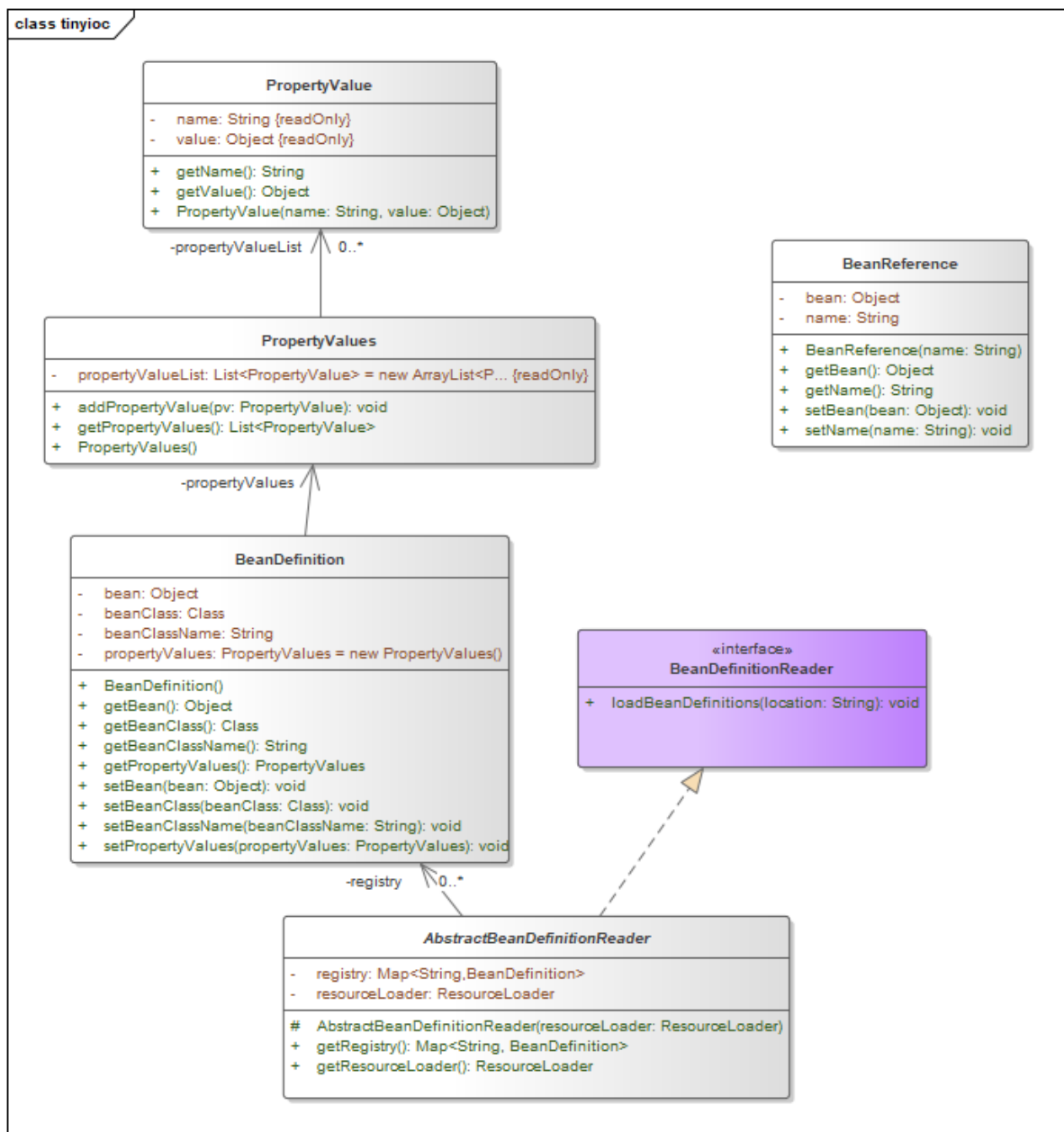
官方给的定义是

在 Spring 中，构成应用程序**主干**并由**Spring IoC容器**管理的**对象**称为**bean**。bean是一个由 Spring IoC容器实例化、组装和管理的对象。

听起来可能有点拗口，从实现的角度来说，IoC容器内管理的对象，是由一个BeanDefinition类来定义的，这个类包装了Object类（所有类的父类），属性信息等字段。通过包装Object类，可以让它保存任何类，保证了IoC容器对客户类型的管理。

可以参考github上的[tiny-spring](#)项目，这是一个开发者自己实现的项目，类名结构等都照搬的Spring，但做了简化，非常方便理解。

下图是通过对tiny-spring(版本四)制作的类图，其中，BeanDefinition就是bean的类，而PropertyValues是对类的属性做管理，下面的BeanDefinitionReader是用来实现通过xml文件自动装配的。独立的BeanReference是为了解决Bean引用Bean而设计的。

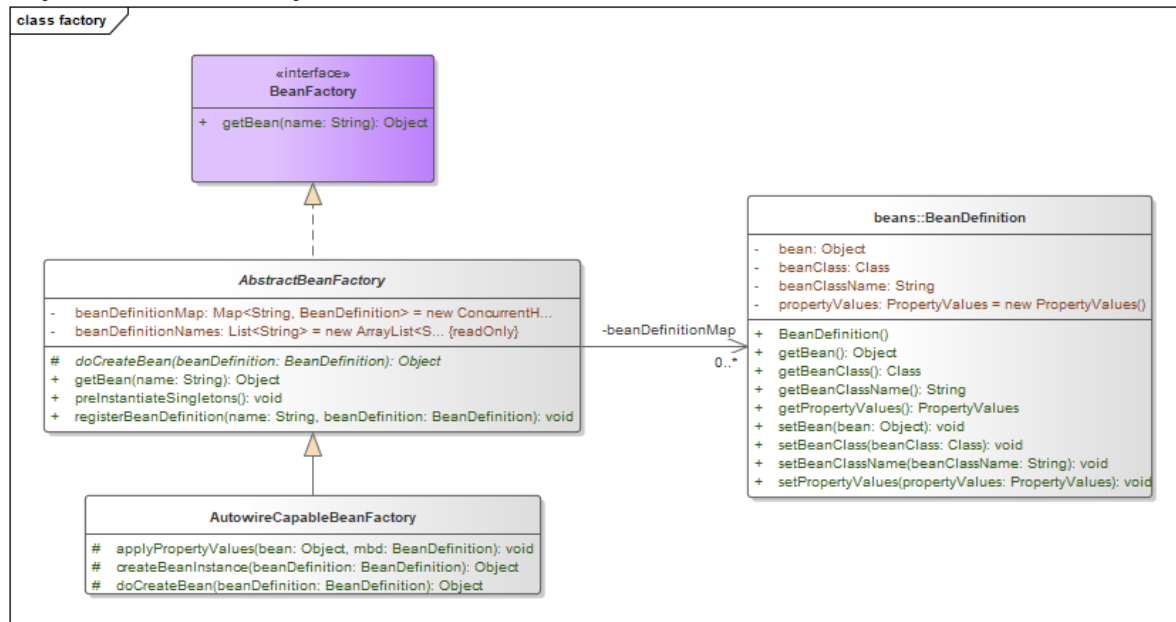


控制反转(IoC)和依赖注入(DI)

控制反转的实现，设计上有些类似于**工厂方法模式**。它将工厂方法模式中的抽象建造者类，添加了一个Bean容器，用来实现对于Bean的管理。在工厂方法模式中，创建具体对象的方法，会被交给具体工厂来实现，但是这样的问题就是，每当有一个新的产品，就需要编写新的具体工厂类，这给编程带来了麻烦。而SpringIoC通过Java的反射机制，来实现对与具体类的创建。反射可以在运行时获得任意一个类的所有信息，在运行时动态的注入，实现对象间的松耦合。通过这样的方式，IoC实现了将对象的管理，将对对象的管理从具体类里抽取出来，并且适合于所有类。

下面是通过tiny-spring制作的类图（做了精简），可以看到，这样的结构非常类似于一个工厂方法模式的结构，首先有一个BeanFactory的接口，然后，定义了一个抽象工厂（和工厂方法模式中的抽象工厂类类似），然后下面是一个具体工厂（和具体的工厂类类似，但是这个工厂配合BeanDefinition能实现更强大的功能），抽象工厂是对BeanDefinition操作，想要创建其他类时，其他需要被创建的类不需要像工厂方法模式一样去继承某个抽象类（因为所有类都是Object的子类），而BeanDefinition则时对

Object类做了管理（基于java反射机制）。



另外注意，在实际开发中，我们使用的是Springboot来做开发，它基于注解来配置IoC容器，而不是xml文件，在需要交由Spring管理的类前，添加相应的注解，才能让Spring对其进行管理。在我们的项目中，netty的handler并不属于Springboot架构中的某个组成部分，无法在其中直接使用注解来进行自动注入。一般有两种方法解决，第一种是实现ApplicationContextAware接口，定义getbean等方法，在运行时使用getbean方法来获取；另一种是在类前加入@Component注解，首先将该类交给Spring管理，然后在@PostConstruct方法，初始化一个初始化器，在初始化器内对Bean进行注入。

Netty

责任链模式

//TODO

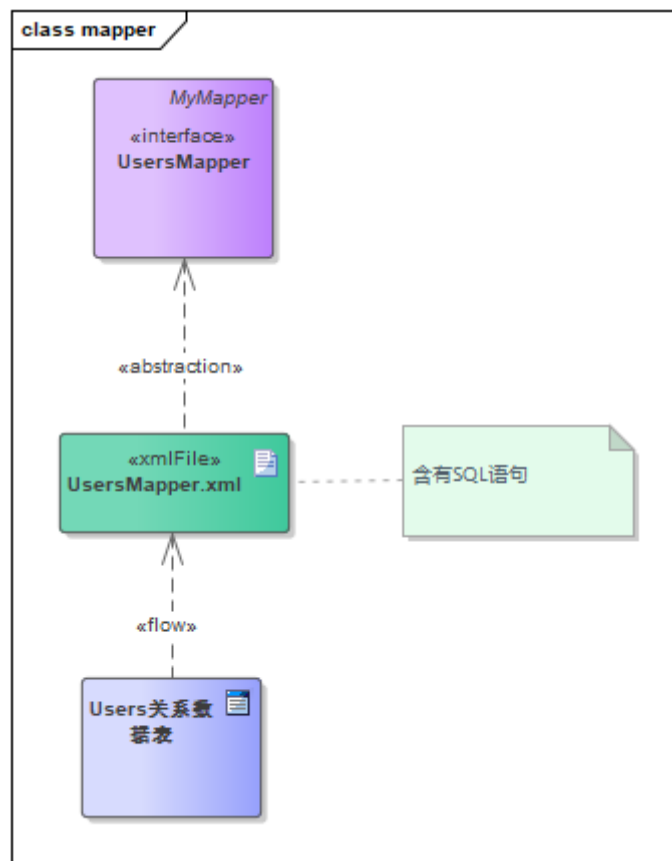
Mybatis

对象关系映射(ORM)

本项目采用了Mybatis的通用生成器生成了pojo类的映射类和resources内的xml映射文件，同时继承了通用Mapper来简化Sql语句的编写。

ORM持久层常用的设计方案，通过将关系数据库的表和面向对象的类关联起来，避免了直接编写SQL语句带来的类与关系数据库的耦合。ORM属于MVC结构中的Model，可以方便MVC结构的使用。

以Users为例，下图（箭头只代表有关联，上面的文字不准确）展示了一组映射关系，xml包含SQL语句，而UsersMap类则与对应xml文件关联，因此通过UsersMapper可以方便的以对象的方式访问数据库。



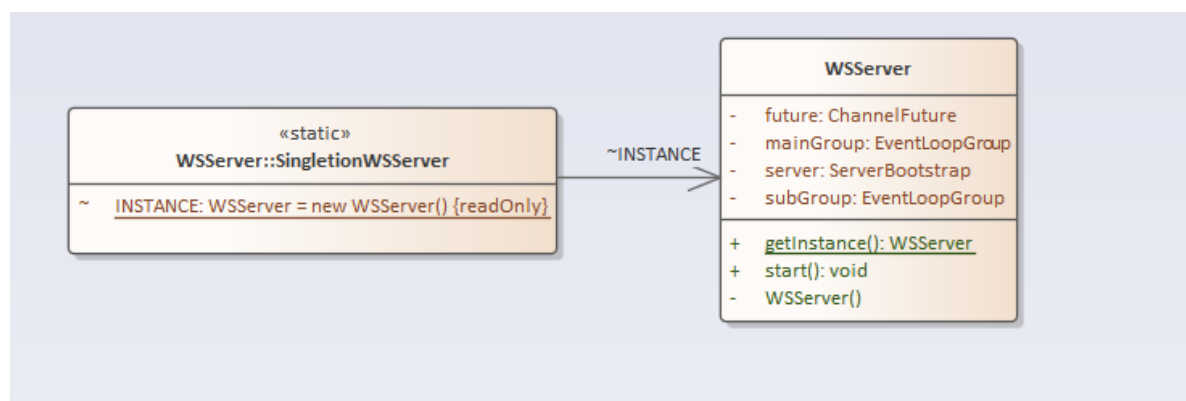
在这里简单讲解下Mybatis是如何实现ORM的，此外开发者还可以参考[这篇文章](#)，为避免文章失效，[网页打印的PDF](#)也会一并下发。

过程很复杂，这里希望读者返回去阅读java反射一章。实现上，Mybatis利用反射，实现了一个**动态代理工厂**，可以注意到，我们定义的是一个UsersMapper接口，却并没有实现它，这是因为，使用在启动时，首先Mybatis会加载xml和配置文件，获得Sql语句和mysql连接信息，通过Sql语句创建我们平常用JDBC连接数据库的SqlSession等，然后把我们的UsersMapper接口传入到一个 Configuration 对象的 getMapper方法里，在这里，通过反射的方式，在运行时生成了UsersMapper的实现类，通过这样的方式，来实现了我们的接口和xml文件内Sql语句的映射。具体还请下发的PDF或者可以阅读源码。

自己的设计

1.

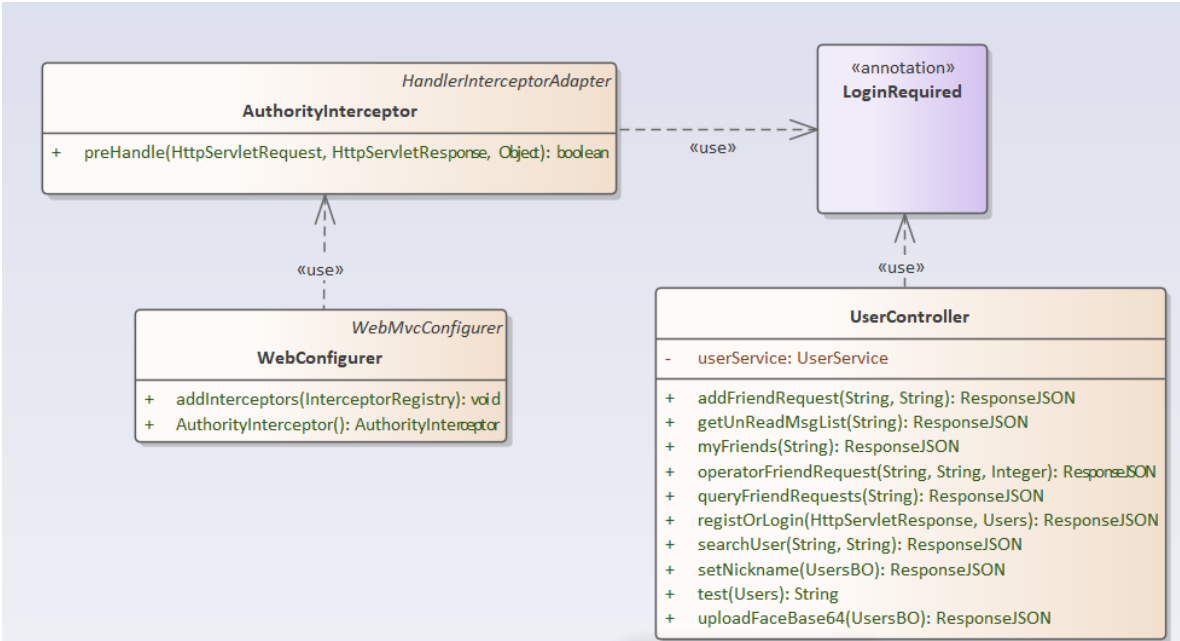
为保证只有一个WebSocket服务实例对WSServer类采用单例模式



2.

基于spring注解驱动过滤器

这和传统的拦截过滤器在编写的时候有所不同，由于spring已经帮我们定义好了拦截器的接口，和父类，我们只需要集成HandlerInterceptorAdapter这个类，并重写他的preHandle方法，就可以实现我们的拦截器，同时通过自定义注解，我们在使用时，只需要在接口前添加@LoginRequired注解即可(),注意，以下类图只是我们编码时会使用的类，但基于spring的注解拦截方式会涉及到更多的类。



6 用户界面设计

5.3 系统界面详细设计

系统界面说明应用系统软件的各种接口。整个系统的其他接口（如系统硬件接口、通讯接口等）在相应的部分说明。

5.3.1 外部界面设计

根据系统界面划分进行系统外部界面设计，对系统的所有外部接口（包括功能和数据接口）进行设计。

5.3.2 内部界面设计

设计系统内部各功能模块间的调用关系和数据接口。

5.3.3 用户界面设计

规定人机界面的内容、界面风格、调用方式等，包括所谓的表单设计、报表设计和用户需要的打印输出等设计。此部分内容可能比较多。

6、数据库系统设计

此数据库设计可单独成册，尤其对大型的数据库应用系统，即有一个单独的《数据库设计说明书》。

6.1设计要求

6.2 信息模型设计

确定系统信息的类型（实体或视图），确定系统信息实体的属性、关键字及实体之间的联系，详细描述数据库和结构设计，数据元素及属性定义，数据关系模式，数据约束和限制。

6.3 数据库设计

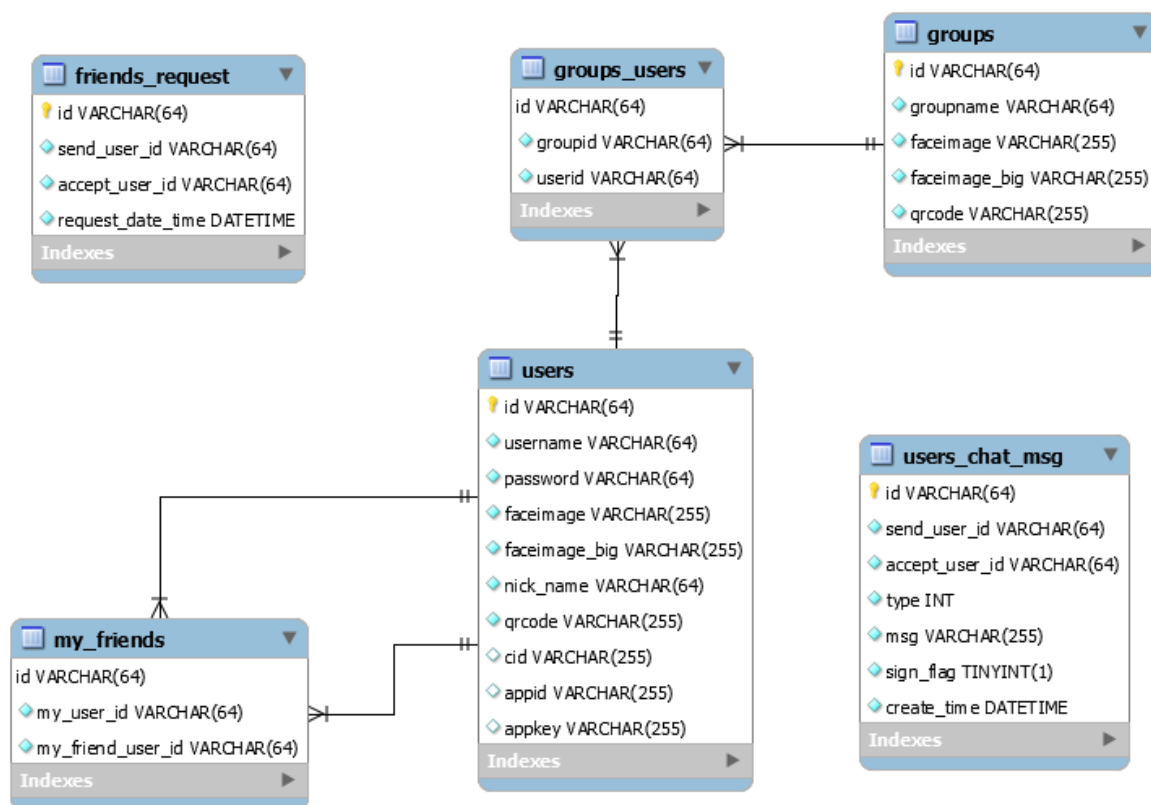
6.3.1 设计依据

由于用户的登录信息需要被频繁访问到，所以使用redis存在内存里，用户的消息需要做离线消息，说明数据被访问的频度和流量，最大数据存储量，数据增长量，存储时间等数据库设计依据。

6.3.2 数据库种类及特点

说明系统内应用的数据库种类、各自的特点、数量及如何实现互联，数据如何传递。

6.3.3 数据库逻辑结构



如图，my_friends表记录

6.3.4 物理结构设计

列出所使用的数据结构中每个数据项的存储要求、访问方法、存取单位和存取物理关系等。建立系统程序员视图，包括：

- 数据在内存中的安排，包括对索引区、缓冲区的设计；
- 所使用的外存设备及外存空间的组织，包括索引区、数据块的组织与划分；
- 访问数据的方式方法。

6.3.5 数据库安全

说明数据的共享方式，如何保证数据的安全性及保密性。

6.3.6 数据字典

编写详细的数据字典。对数据库设计中涉及到的各种项目，如数据项、记录、系、文卷模式、子模式等一般要建立起数据字典，以说明它们的标识符、同义名及有关信息。