WIO从零开始部署

### 第一步，打开码云网站，下载项目

<https://gitee.com/uwolf1233/w-io>

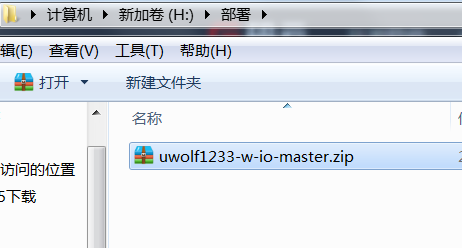


点击右边的克隆/下载按钮

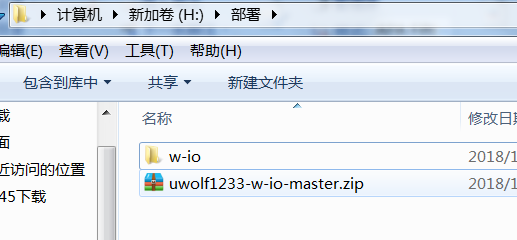


点击下载ZIP后，如果需要先登录，则先去登录

再次点击下载后需要输入验证码，然后打包下载，码云后台帮你打包了



然后找个地方放置压缩包

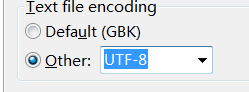


解压后会得到w-io文件夹，资源就下载好了

#### 第二步，引入日志项目

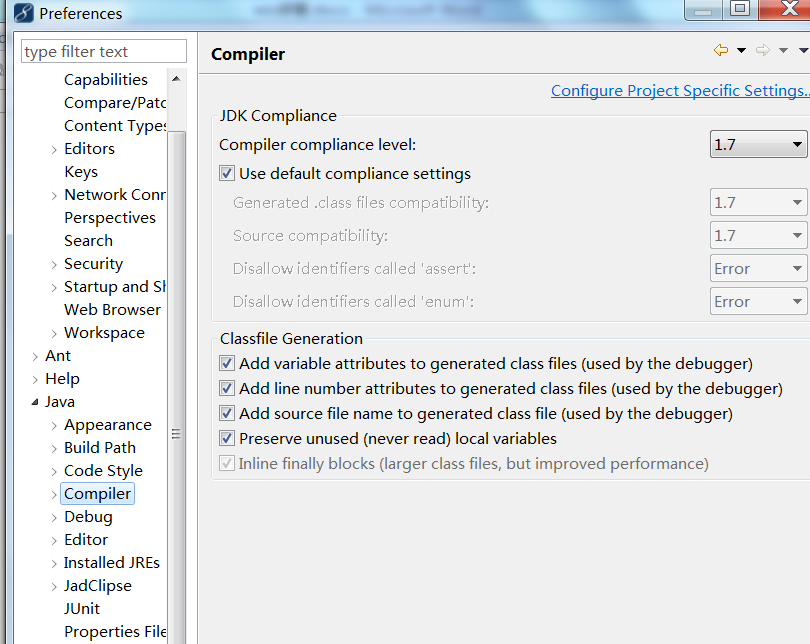


我这用的是myeclipse10，目前本项目在java7下开发，所以me10足够了



注意，编码一定要设置UTF-8，否则出现乱码

Jdk设置就不多说了，用jdk7



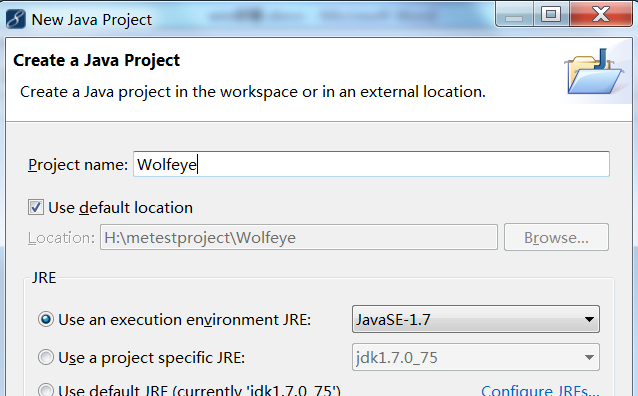
这个记得要设置为1.7

这些设置好后，就可以开始引入项目了

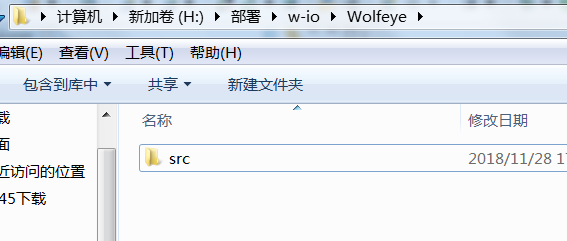


第一个必须引入的项目是Wolfeye，这是个日志，虽然没有完成，但这是必须引的

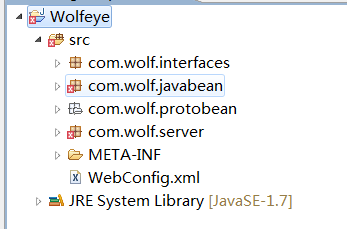
在myeclispe新建java project



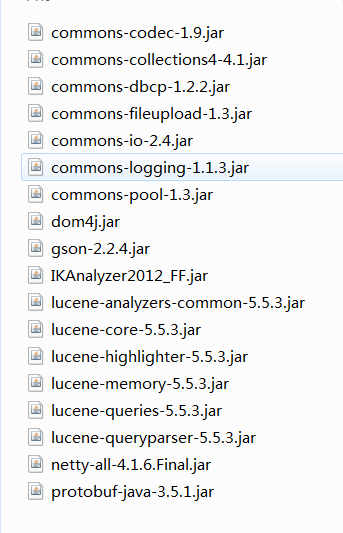
命名为Wolfeye，点击finish

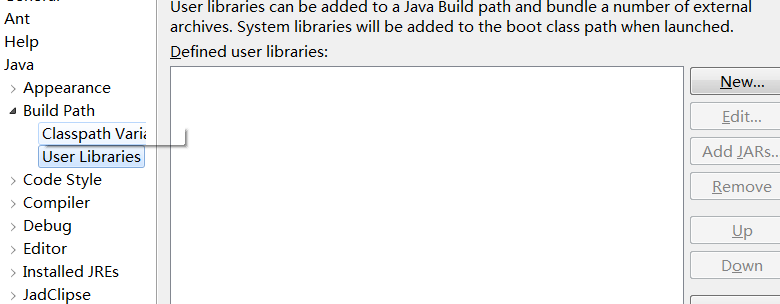


将Wolfeye下的src文件夹拷贝过去

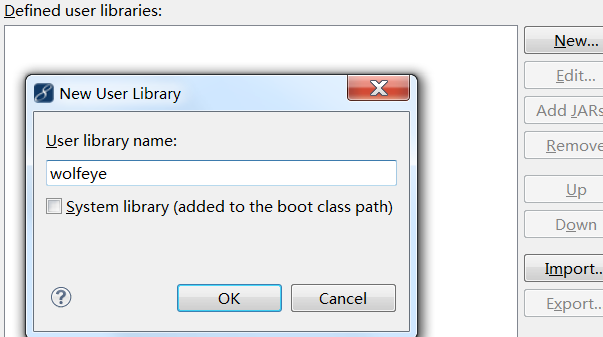
拷贝完成后就是这样子

然后就是导入jar包，需要的第三方jar如下

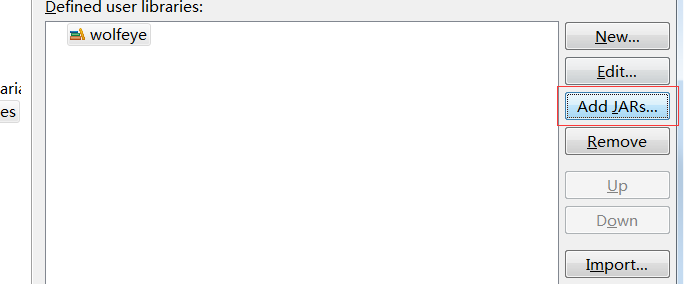




可以专门做个user Linraries

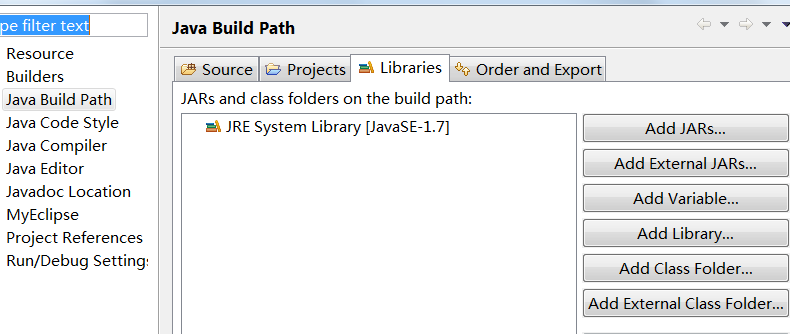


点击新建，输入名称



点击add JARs，把所有eyelib下的jar加入

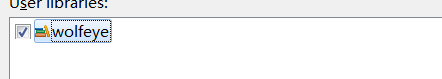
然后右键Wolfeye，进入这个界面



点击 Add Library

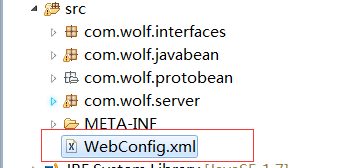


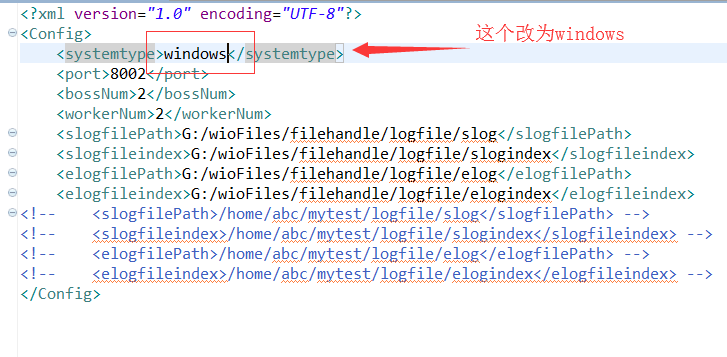
选择User Library

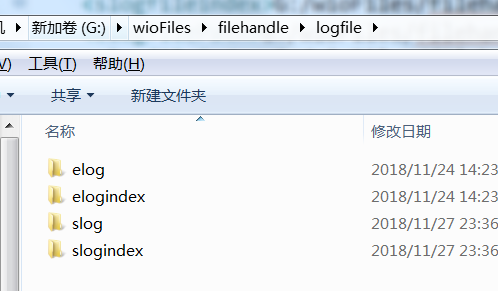


选择刚刚加好的jar包

将jar包加入项目后，点击WebConfig.xml,打开

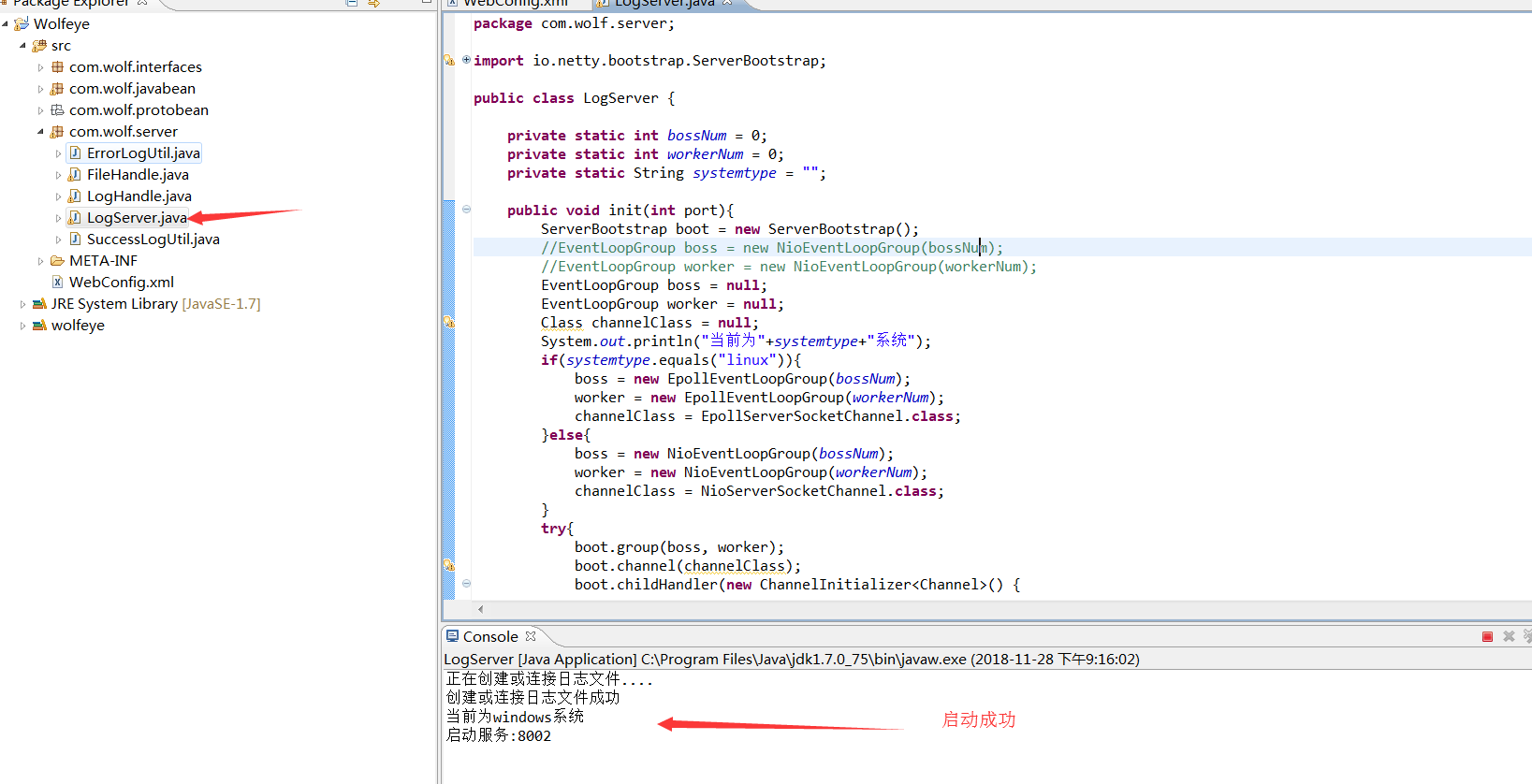






如果是windows系统，Systemtype改为windows，然后根据上面的配置建立这几个目录

这些设置完后，就可以启动项目了，启动方式看下图所示



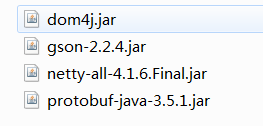
打开LogServer.java，然后右键Run，执行main函数，这些都是基础就不多说了

看到没有报错并且有下面的打印语句，证明启动成功了

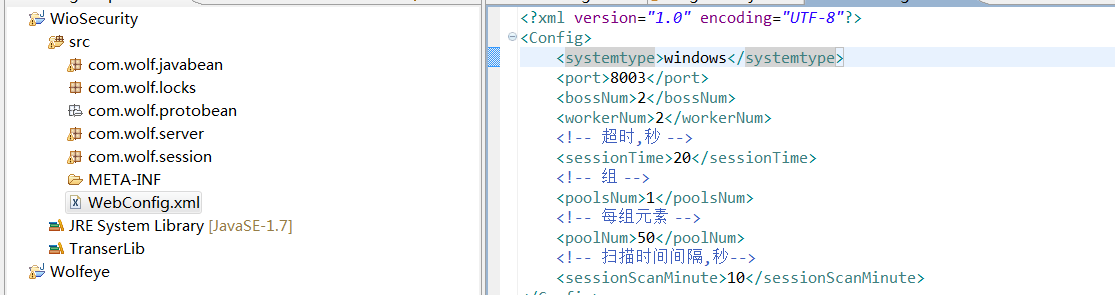
#### 第三步，引入会话项目



会话项目是这个，为了后面扩展安全性，各种验证等，才起的这个名字，先不多说，先部署，和之前的模块基本类似

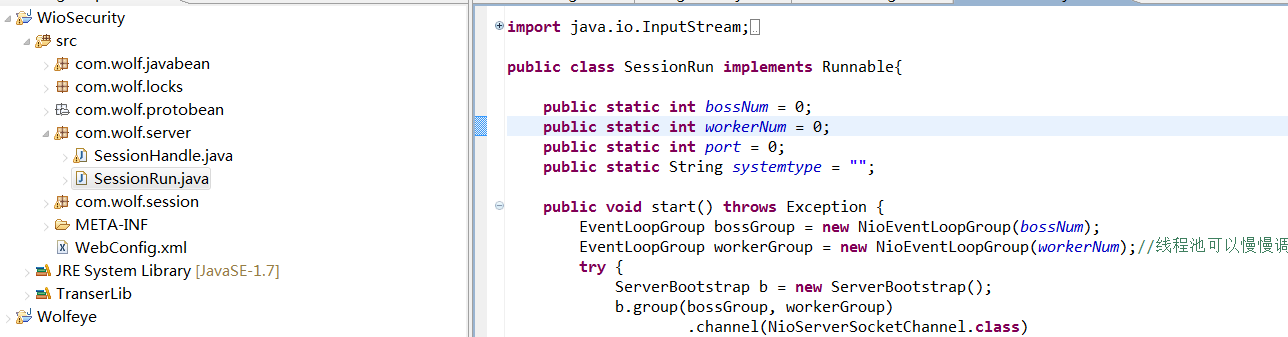


这个项目的jar就这四个，刚刚wolfeye项目的jar里面包含了这四个



然后点开WebConfig.xml根据需要修改相关配置，bossNum和workerNum每个项目都有，学过netty的都知道，没接触过也没关系，不用管，超时时间按需求设置，组，如果需要session多，可以多设置几个组，每组元素，可以稍微设置少一些，占内存也相对少一些。

这个实现我大概说一下，这其实是一个二维数组，每个组都有自己的定时执行，循环1000个session还是比较快的，所以每组元素设置1000也行，前提是内存够，因为后续把数据存入session也占内存，session支持集群扩展，目前这块没测试，但单个session服务没问题，session是和服务一起的，意思是只要服务启动着，里面的session不会被jvm回收，每个服务的session都是固定数量的，由自己设置，session的id包含了组流水和jvm流水，组流水是指当前session属于哪个组，组比较多的情况下也能快速找到自己所在的组。



设置完成后找到sessionRun右键Run启动

这样就表示session服务启动成功

Session相关方法

isOvertime() 是否请求超时

getSessionId() 获取本地sessionid

setSessionId(String sessionId) 设置本地sessionid，一般情况框架会获取到

getPath() 获取本地路径

setPath(String path) 设置本地路径

getDatas() 获取本地数据

setDatas(ByteString datas) 设置本地数据

create(Map<String,Object> map) 创建session,凡是超时，都将到下一个session服务中，map为需要存入session的数据

hasSession(GetResult getResult) 异步判断session是否存在

hasSessionRun(WioThreadResult wioThreadResult) 同步判断session是否存在，开发中，参数为null，当前参数是提供给异步调用，后面的方法也一样

vaildPath(String path,GetResult getResult) 异步验证路径

vaildPathRun(String path,WioThreadResult wioThreadResult) 同步验证路径

setPath(List<String> paths) 设置路径组

setAttr(String key,Object o,GetResult getResult) 异步设置属性

public boolean setAttrRun(String key,Object o,WioThreadResult wioThreadResult) 同步设置属性，o为可序列化对象

getAttr(String key,GetResult getResult) 异步获取属性

getAttrRun(String key,WioThreadResult wioThreadResult) 同步获取属性

removeAttr(String key,GetResult getResult) 异步删除属性

removeAttrRun(String key,WioThreadResult wioThreadResult) 同步删除属性

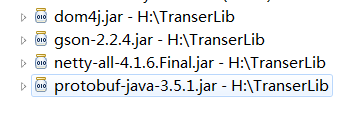
reset() 重置session

clearAttr(GetResult getResult) 异步清空session所有属性

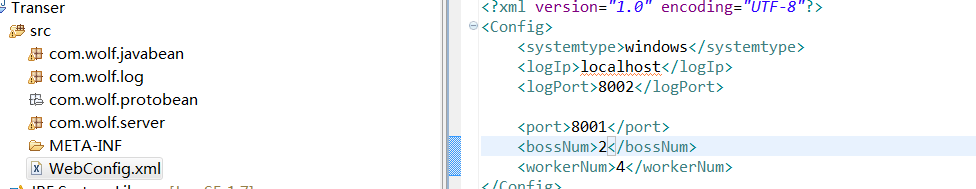
clearAttrRun(WioThreadResult wioThreadResult) 同步情况session所有属性

#### 第四步，引入事务控制项目

事务控制项目是Transer

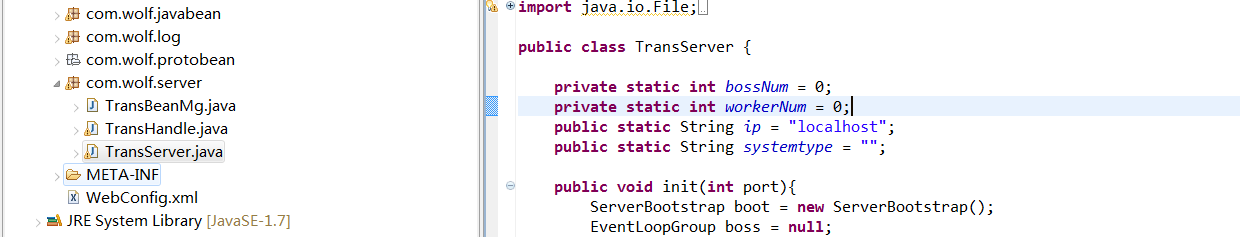


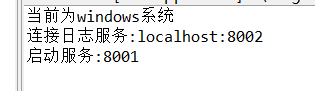
这个项目也是这四个jar包，只需要引入就好了



根据需要进行设置，建议各种端口配置不要变，不然连自己都记不住，有冲突除外

事务控制服务是基于三阶段提交的，建议worker线程池稍微设置大一些，避免因为池耗尽导致的死链





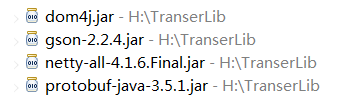
找到TransServer进行Run启动

#### 第五步，引入入口项目

入口项目是整个框架的核心之一，里面包含了http解析，websocket解析，内部接口调用，负载均衡，分布式锁控制，注册中心，前置数据缓存，静态资源缓存等，目前在不断的完善中。



有两个，一定要用NoSpring版本的，因为第一个已经不再维护了，当然，起名字还是可以按第一个起



Jar包还是框架的四大金刚，这里没啥可说的，打开配置文件



注释掉的不用理会了

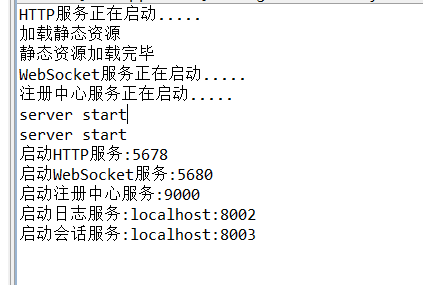
Path指的是磁盘上的任意路径，只要下面有WebAPP这个目录，这个目录的所有文件都会载入堆外内存中，一定要注意，是堆外内存，数据不在jvm内存中，所以不要什么乱七八糟的文件都放进来，避免堆外溢出，堆外内存就算快满了也不会导致FullGC，这点要慎重。W-io的前端目录结构和tomcat的不太一样，这个是固定式的，不能通过../模式找上层文件夹，直接/就是WebAPP下面的一层，不管你的静态资源文件是第几层。

Viewtype，如果访问的是这些后缀，将直接请求，不受注册路径不存在的拦截，一般情况下，业务访问路径要注册而且需要http模式前端才能访问

负载均衡策略，第一种是满了到下一个，第二种是轮询式

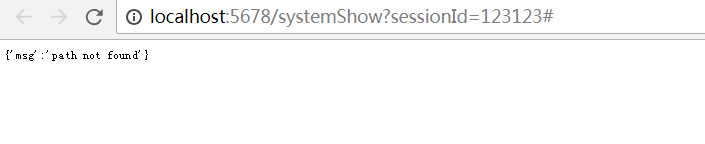
其他可以不用管，配置好后找到spring目录下的StartInit.java，然后Run启动项目





#### 第六步，业务项目

目前W-io框架有点特殊，没有个业务服务进不去任何html页面



现在加个业务服务去控制



业务项目除数据库连接除外，需要这五个jar，除了四大金刚外，还有个WIO-Server.jar,这个jar在下载的项目里面有



也可以通过RegServer项目进行打包。



Xml配置可以根据WioUser项目的xml来做，直接复制过来，基本不用改什么



然后新建一个controller，可以先是一个空java类

然后新建一个Main文件，把启动代码加入进去

**public** **static** **void** main(String[] args) {

InputStream in = TestsMain.**class**.getResourceAsStream("/WebConfig.xml");

*regController*();//运行注册器，注册器必须在配置读取之前执行

OtherServer.*mains*(in);//运行

OtherServer.*execServer*();

}

**public** **static** **void** regController(){//注册

RefController.*INSTANCE*.putoList(**new** TestController());

}

启动项目示例代码

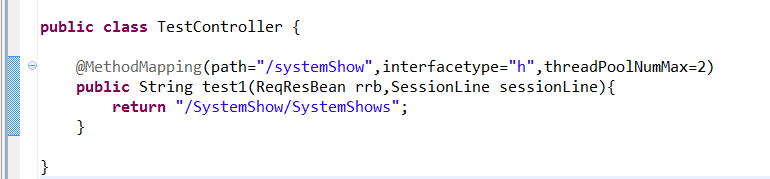
在controller加入

@MethodMapping(path="/systemShow",interfacetype="h",threadPoolNumMax=2)

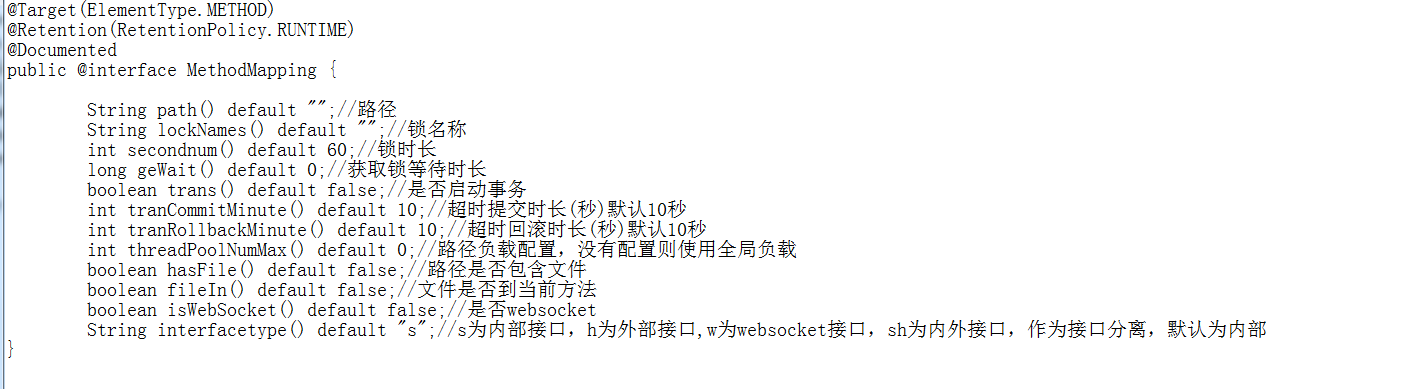
**public** String test1(ReqResBean rrb,SessionLine sessionLine){

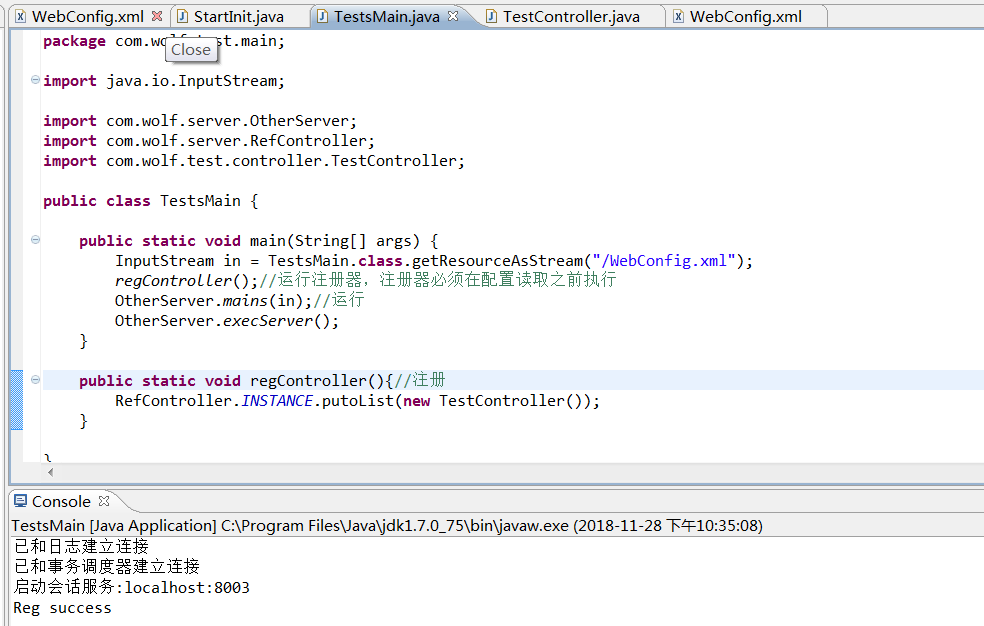
**return** "/SystemShow/SystemShows";

}



MethodMapping的参数可以参考RegServer项目





然后右键启动这个main

然后打开浏览器输入URL

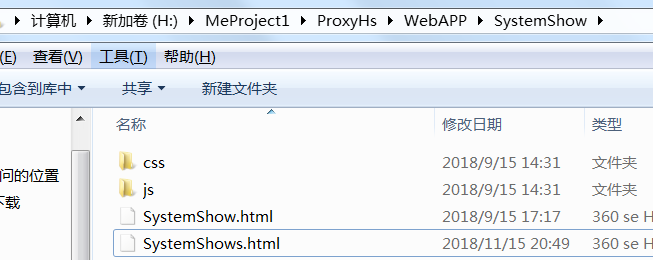
<http://localhost:5678/systemShow?sessionId=123123>

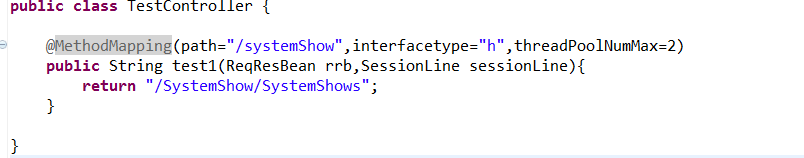


界面就可以打开了，这里注意一下，sessionId=123123目前没有做前置session拦截，也可以自己添加，这个后续会在框架中补充

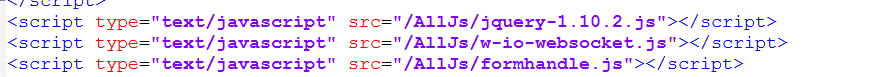
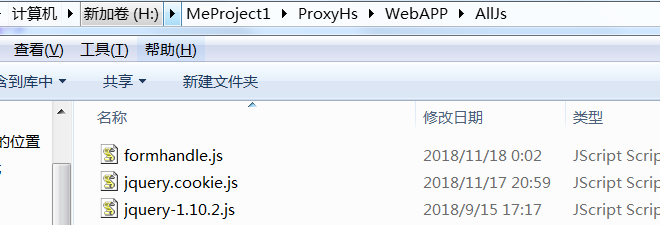
界面，这个是自己的界面，界面代码没完全开源，有些涉及到别人的开源代码，所以我就不再做开源了，界面其实可以自己做，就是路径规则需要改造

例如，我的前端文件路径是





路径访问规则基本如此，WebAPP下直接指向对应路径，但不需要后缀

静态资源文件加载也是这个路径加载规则，不管你的文件在第几层，一样的规则，要和tomcat，jboss的模式区分开来

这里还有个需要注意的地方

//用于刷新页面缓存

**function** sys\_RefResh(){

$.ajax({

url:'/systemRefreshStatic',

type:'get',

data:{'sessionId':'123123'},

dataType:'text',

beforeSend:**function** (R) {

R.setRequestHeader('Connection', 'Keep-Alive');//复用连接

},

success:**function**(data){

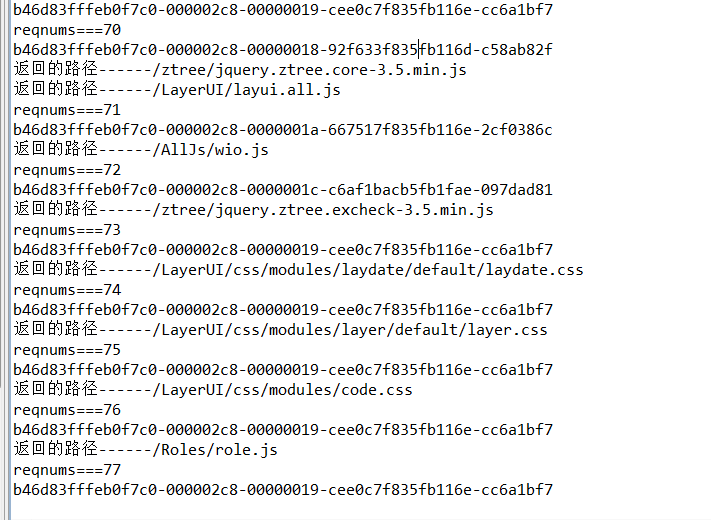
location.reload(**true**);

}

})

}

页面修改后，按f5是不能马上刷新代码的，需要执行这个ajax才行，执行这个ajax后，缓存将重新加载所有静态资源文件，再按f5刷新，所以我把这个请求写在了缓存刷新按钮



后台负载均衡，各种请求等打印可以在入口项目的打印看到

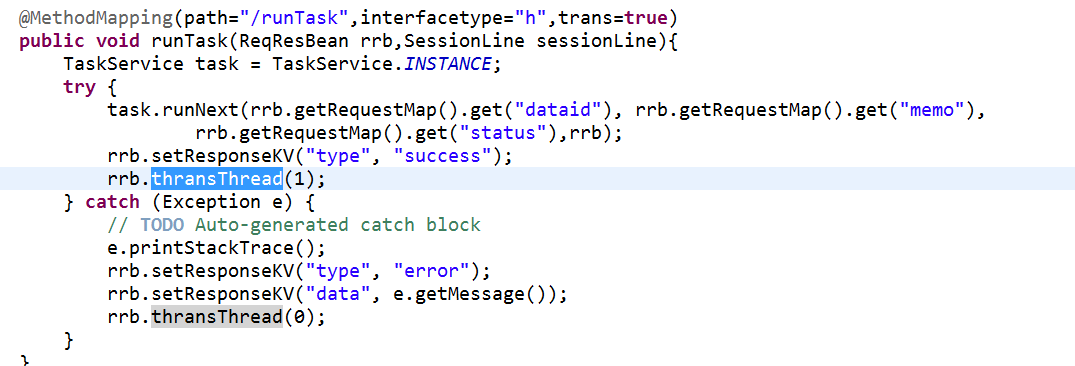
至于controller，可以和其他mvc一样开发

数据库连接池采用目前最快的HikariCP连接池，设置方面可以参考WioUser项目中的

Serconfig下的Config类，至于连接什么数据库，就看自己需求了

在controller类下部分代码示例

Cookie添加代码示例：rrb.addCookie("wsessionId", "123123");



一个基本示例

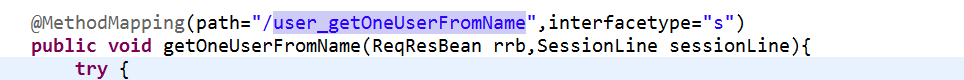
事务：需要设置trans=true，但是如果设置了，thransThread，1为提交，0为返回，如果没有做提交或者回滚，可能导致事务出现异常，内部接口调用下，事务目前在同步调用，伪异步调用下生效，真异步调用则无效

事务传播，由第一个事务方法生成主控事务，框架已做，调用链中每个方法并不需要都配置了事务，但事务传播下去必须是同步调用或者伪异步调用，不能是真异步调用。

调用可以参考login项目

同步调用：Object o = srb.send("/user\_getOneUserFromName", map, true, rrb);

第一个参数是请求的路径，path中的配置



第二个参数是传递的参数是Map<String,String>

第三个参数是是否同步调用，是否真异步调用

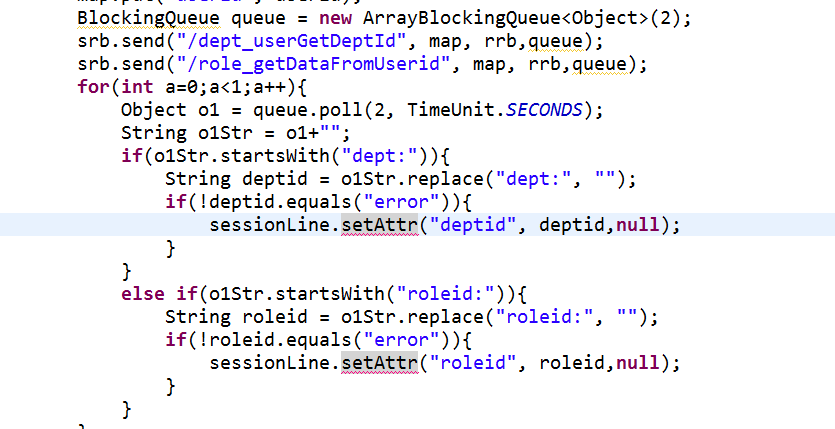
第四个参数是ReqResBean，controller方法带了

Interfacetype: s为内部接口调用，h是http，w是websocket

伪异步接口调用示例

srb.send("/dept\_userGetDeptId", map, rrb,queue);

srb.send("/role\_getDataFromUserid", map, rrb,queue);



伪异步接口调用其实是同步调用，但不需要立刻返回数据，可以先执行其他耗时的代码，再从queue队列中取数据，至于调用了多少接口，自己有数就行，queue相关知识，可以看看javaapi

获取request数据示例：

rrb.getRequestMap().get("id")

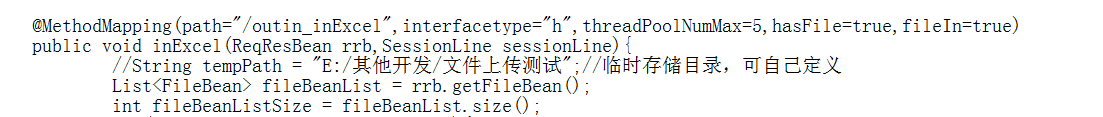
数据返回：

rrb.setResponseKV

rrb.setResponseMap

rrb.setResponseData

这里要重点说一下，不管是request还是response，都统一了http请求返回数据和内部接口调用都是统一的，为了上手更加方便



文件上传，可以参考WioOutInData项目，支持多个文件上传，虽然这个项目还在开发中，但文件上传已经测试过了

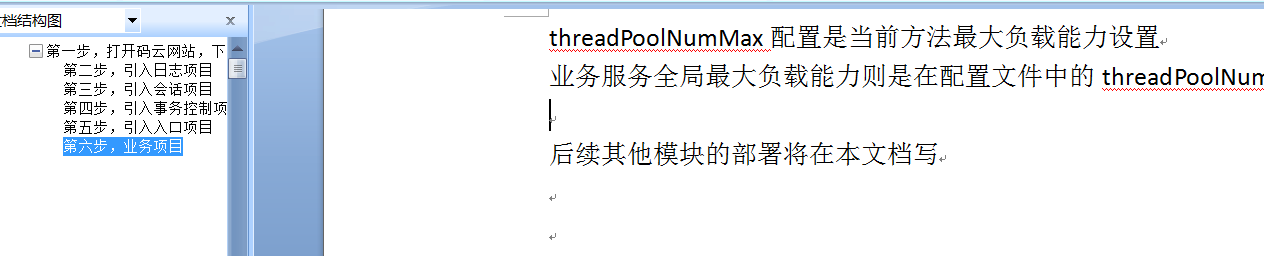
hasFile和fileIn必须设置为true

threadPoolNumMax配置是当前方法最大负载能力设置

业务服务全局最大负载能力则是在配置文件中的threadPoolNumMax

如果还是部署不成功，[请邮件784804638@qq.com](mailto:请邮件784804638@qq.com)，写上”w-io问题”

后续其他模块的部署将在本文档写



#### W-IO的其他部分

W-io从设计之初就考虑了更多的灵活性，不仅仅是基本的增删改查。W-io是基于netty4+protobuf3之上来封装的，甚至考虑了物联网发展方向。基本的mvc模式只是其中一种，目前还提供了非业务模块也非核心模块的其他服务，例如加密解密，mq等，其实会话也算一种，这些都可以脱离框架独立部署，会话模块需要从RegServer中切代码出来，加密解密和mq提供了客户端和服务端独立部署的能力。

当然，只要你懂netty，你完全可以开发属于自己的模块

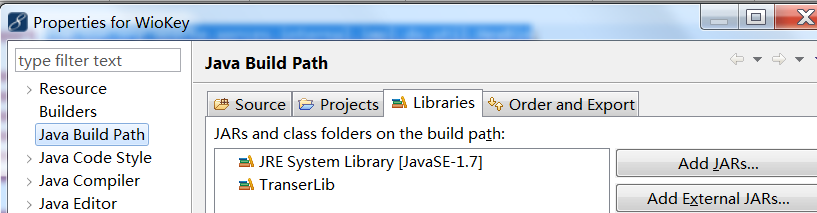
加密解密部署：

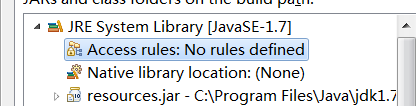
，先找到WioKey模块，按之前的套路，还是那四个jar包

接下来的一步需要注意

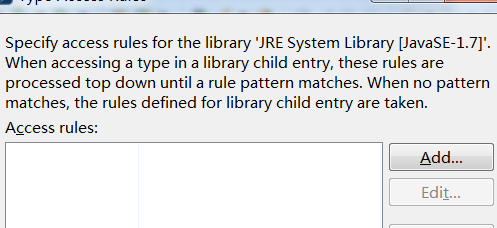
com.sun.org.apache.xerces.internal.impl.dv.util.HexBin不属于JDK标准库的范畴，但JDK却包含了这个，所以eclipse或者myeclipse会报错

先右键项目，打开界面如下图

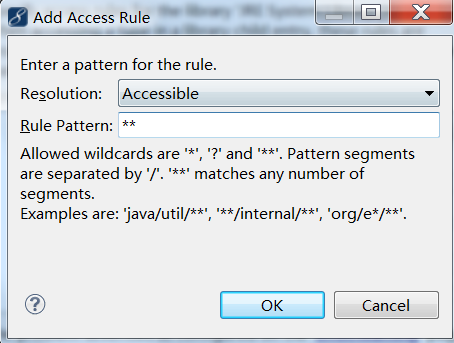




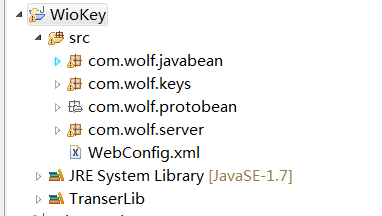
然后点开这个，双击Access rules



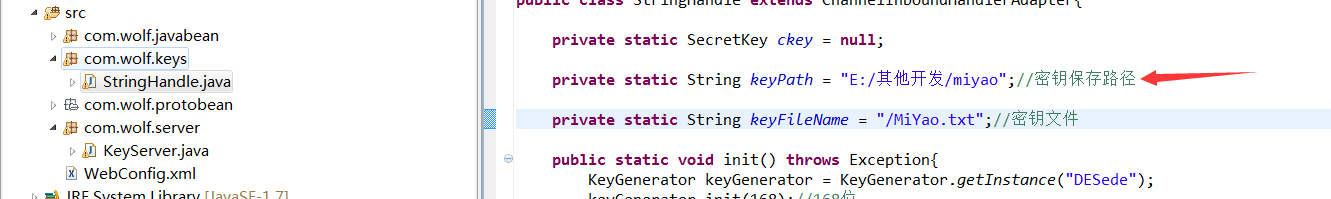
点击add



选择Accessible，下面输入框填入\*\*，然后点击OK



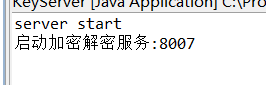
然后就不报错了



然后点开StringHandle，修改keyPath为自己电脑上的一个目录



然后打开KeyServer文件，run运行



执行成功

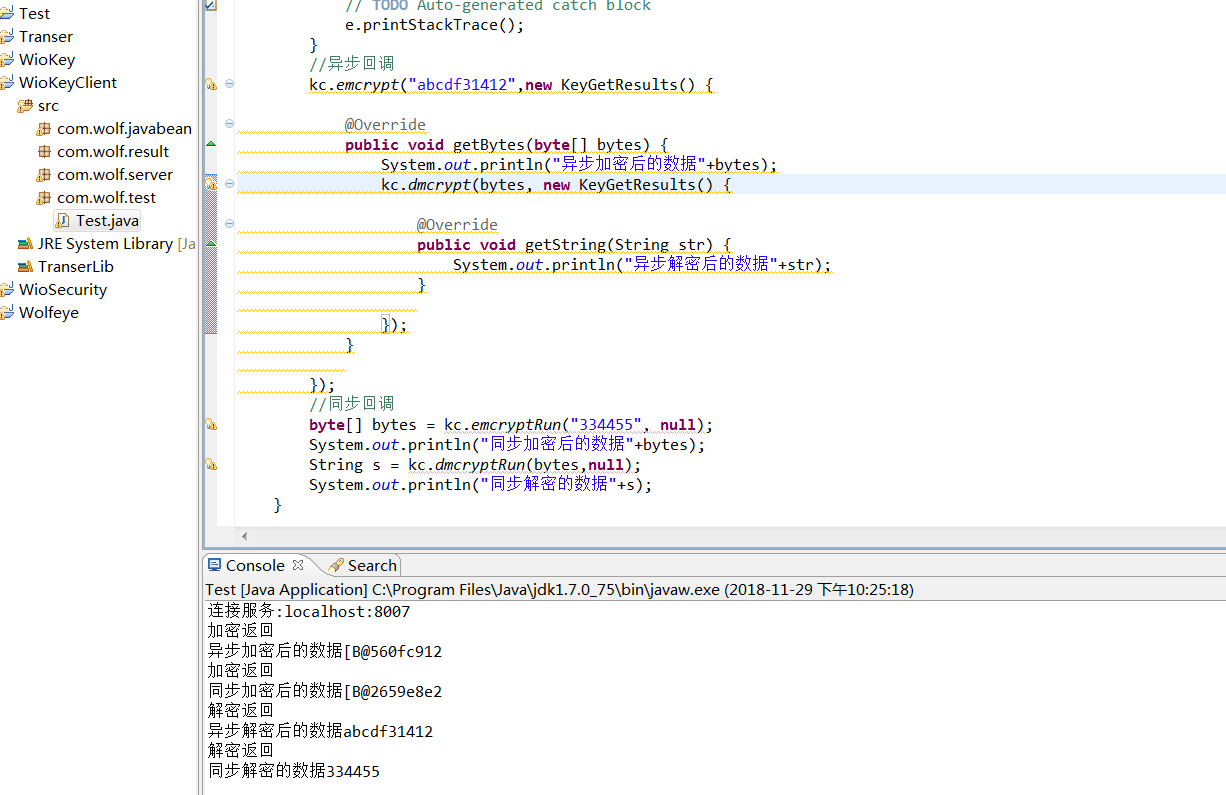


然后引入这个项目，这是加密解密客户端

依旧是那四个jar包



点开Test文件，可以运行测试代码试用一下



执行这个Test后的结果，里面包含了同步调用和异步调用。

加密解密用的是对称加密

#### W-IO的代码生成器

W-io代码生成器主要用于快速生成业务代码，目前界面包括新增，修改，删除，查询，导入，导出，提交，审核，通用数据表格，后端包括controller，server，dao，sql，javabean的生成，可自己修改和扩展，项目WioTemple

**关于和w-io内存相关**

W-io底层是基于netty4开发，在堆外内存和堆内内存方便需要多注意，目前已经过jmeter一万并发压测，包括数据插入和数据查询，一万并发下，入口项目占用内存稍微高一些，业务模块比较低，如果用内存池模式，内存占用达到300m左右，如果不采用内存池，内存占用70m以下，当然前提是设置jvm最低内存不能太高，至于用不用内存池，需要根据自己使用场景。

堆外内存的使用需要格外注意，堆外内存不同于堆内内存，堆外内存溢出的时候，用jconsole去监视jvm内存会发现，jvm内存完全足够，目前w-io框架提供了堆外内存监视类CacheSee，监视入口项目的堆外内存占用，可自己扩展，至于将静态资源文件刷入堆外内存提高加载效率，不建议在实际上线中保留将磁盘上静态资源文件刷入堆外内存的请求，应该在启动的时候或者新做个专门的页面去刷新，否则会降低加载效率。当然，如果在开发模式下，可以考虑F5事件去请求刷新

堆外内存需注意第二点 : 我对w-io框架进行压测的时候发现如果传输速度过快，可能会导致堆外内存溢出，即使是由writeAndFlush方法调用，依旧会导致堆外内存溢出，而堆内内存却完全足够，刚开始我还以为是堆内内存太多导致fullGC过少所以堆外内存不释放，后来调小老年代大小，问题依旧，后来在网上看了一些资料，也做了一些相关测试，原来如果发送速率过快，数据会积压导致堆外内存溢出，这种问题如果是监控jvm内存一般会找不到原因，可能jvm内存还非常充足，注意，堆外内存是直接内存，并不是jvm的内存，所以，一定要做好流量控制，不然问题都不知道去哪里找，w-io也将开发相关的处理策略