谷粒商城的架构

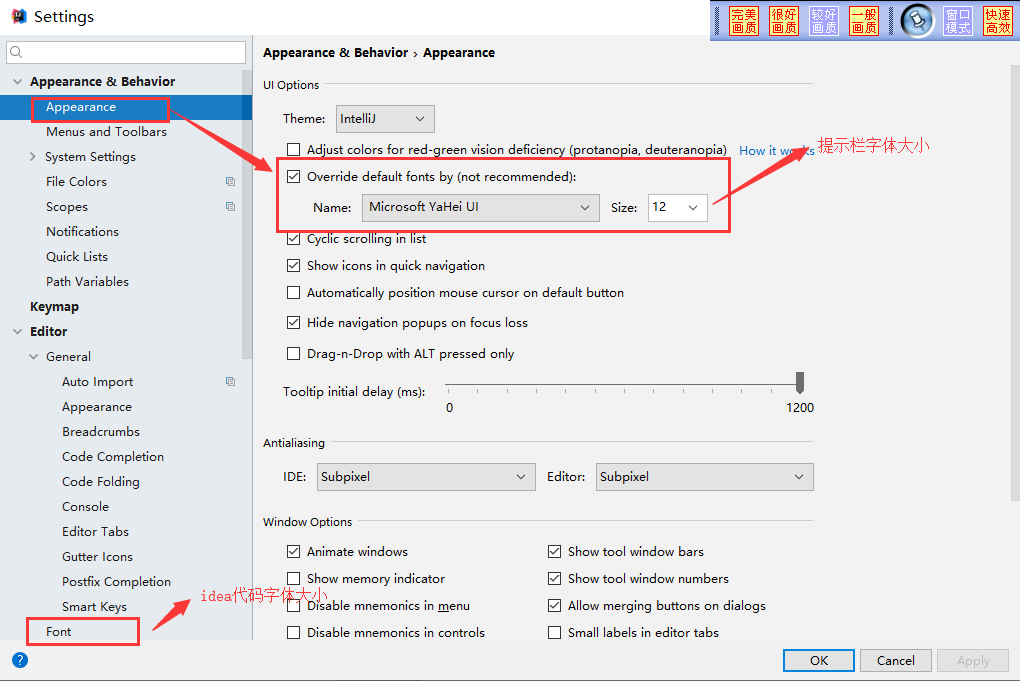
# 1 关于架构的介绍

|  |
| --- |
| 1 工程结构  以maven为基础，对项目的分层架构  2 项目架构  分布式(SOA)  Dubbo(ali) |

# 2 gmalll-parent父依赖的创建

|  |  |
| --- | --- |
| 1 用maven创建一个gmall-parent的工程  2 新建其他项目模块(子项目)的时候，继承自gmalll-parent  3 gmall-parent中父依赖使用springboot1.5   |  | | --- | | <**parent**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-parent</**artifactId**>  <**version**>1.5.21.RELEASE</**version**>  <**relativePath**/> *<!-- lookup parent from repository -->* </**parent**>  <**groupId**>com.atguigu.gmall</**groupId**> <**artifactId**>gmall-parent</**artifactId**> <**version**>1.0-SNAPSHOT</**version**> <**packaging**>pom</**packaging**> |   4 在gmall-parent中定义好项目的技术框架各种版本 |

# 3 调整idea的大小



# 4 抽取api工程（负责管理项目中所有的接口和bean）

|  |
| --- |
| 1 首先用maven创建一个gmall-api的工程  2 引入tk通用mapper(映射类)  3 将XXXService接口和所有的bean类都放到api中  4 service、service实现、controller、mapper、mapper.xml中所有的bean的引入全部修改引入路径 |

# 5 抽取util工程

|  |  |  |
| --- | --- | --- |
| 1 项目中的通用框架，是所有应用工程需要引入的包  例如：springboot、common-langs、common-beanutils   |  | | --- | | <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>fastjson</**artifactId**> </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-test</**artifactId**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-web</**artifactId**> </**dependency**>    <**dependency**>  <**groupId**>org.apache.httpcomponents</**groupId**>  <**artifactId**>httpclient</**artifactId**> </**dependency**>  <**dependency**>  <**groupId**>org.apache.commons</**groupId**>  <**artifactId**>commons-lang3</**artifactId**> </**dependency**>  <**dependency**>  <**groupId**>commons-beanutils</**groupId**>  <**artifactId**>commons-beanutils</**artifactId**> </**dependency**>  <**dependency**>  <**groupId**>commons-codec</**groupId**>  <**artifactId**>commons-codec</**artifactId**> </**dependency**> |   2 基于soa的架构理念，项目分为web前端controller(webUtil)  Jsp、thymeleaf、cookie工具类  加入commonUtil  3 基于soa的架构理念，项目分为web后端service(serviceUtil)  Mybatis、mysql、redis  加入commonUtil   |  | | --- | | <**dependency**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>gmall-common-util</**artifactId**>  <**version**>1.0-SNAPSHOT</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-jdbc</**artifactId**> </**dependency**>  <**dependency**>  <**groupId**>org.mybatis.spring.boot</**groupId**>  <**artifactId**>mybatis-spring-boot-starter</**artifactId**> </**dependency**>  <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**scope**>runtime</**scope**> </**dependency**>   <**dependency**>  <**groupId**>redis.clients</**groupId**>  <**artifactId**>jedis</**artifactId**> </**dependency**> | |

Service+mapper

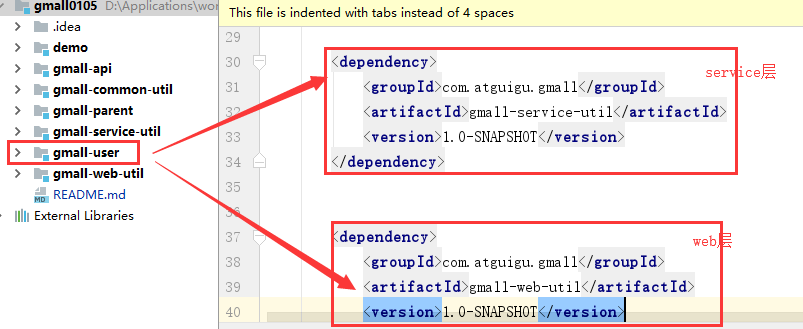
controller

Controller+service+mapper

dbuuo

Db

|  |
| --- |
| 1 新建一个web的前端controller模块的项目  Controller = parent + api + webUtil  2 新建一个web的后端service模块的项目  service = parent + api + serviceUtil |



# 6 soa面向服务(以dubbo为基础)

|  |
| --- |
| 1 dubbo的soa的工作原理，和springcloud类似  2 dubbo和springcloud的区别在于dubbo由自己的dubbo协议通讯，sc是由http协议(rest风格)  3 dubbo有一个注册中心的客户端在时时同步注册中心的服务信息  4 dubbo有一个javaweb的监控中心，负责监控服务的注册信息，甚至可以配置负载均衡 |

# 7 将dubbo框架引入到项目中

启动监控中心

|  |
| --- |
| 1 将dubbo监控中心和tomct上传只linux服务器  2 用unzip命令解压dubbo-admin.war监控中心  Unzip file文件名 -d 解压路径  3 配置tomcat的server.xml配置文件  <Context path="/dubbo" docBase="/opt/dubbo" debug="0" privileged="true" />  4 启动tomcat，打开监控中心 |

Dubbo-dadmin

Linux

Linux

Linux

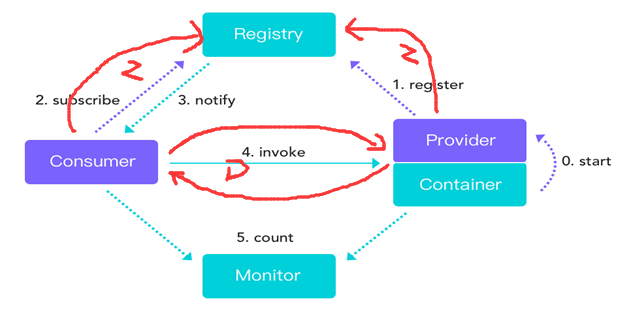
安装和配置zookeeper

|  |  |  |  |
| --- | --- | --- | --- |
| 1 解压zookeeper  Tar -zxvf  2 修改zookeeper的配置文件  Conf/zoo.cfg  建立一个/opt/zookeeper-3.4.11/data数据目录   |  | | --- | |  |   3 启动zookeeper  ./zkServer.sh start  ./zkServer.sh status  4 设置监控中心和zookeeper的开启自启动  A 来到/etc/init.d  B 创建启动脚本vi dubbo-admin   |  | | --- | |  |   C 用chkconfig –add dubbo-admin  D 来到etc/init.d目录  新建 vi zookeeper  写入脚本   |  | | --- | | #!/bin/bash  #chkconfig:2345 20 90  #description:zookeeper  #processname:zookeeper  ZK\_PATH=/opt/zookeeper  export JAVA\_HOME=/opt/jdk1.8.0\_152  case $1 in  start) sh $ZK\_PATH/bin/zkServer.sh start;;  stop) sh $ZK\_PATH/bin/zkServer.sh stop;;  status) sh $ZK\_PATH/bin/zkServer.sh status;;  restart) sh $ZK\_PATH/bin/zkServer.sh restart;;  \*) echo "require start|stop|status|restart" ;;  esac |   Chkconfig –add zookeeper  Service zookeeper start |

# 8 将项目改造为dubbo的分布式架构

|  |  |
| --- | --- |
| 1 将user项目拆分成user-service和user-web  2 引入dubbo框架  将dubbo框架引入到common-util中(因为web层和service层将来都需要使用dubbo进行通讯)   |  | | --- | | <**dependency**>  <**groupId**>com.alibaba</**groupId**>  <**artifactId**>dubbo</**artifactId**> </**dependency**>  <**dependency**>  <**groupId**>com.101tec</**groupId**>  <**artifactId**>zkclient</**artifactId**>  <**exclusions**>  <**exclusion**>  <**groupId**>org.slf4j</**groupId**>  <**artifactId**>slf4j-log4j12</**artifactId**>  </**exclusion**>  </**exclusions**> </**dependency**>  <**dependency**>  <**groupId**>com.gitee.reger</**groupId**>  <**artifactId**>spring-boot-starter-dubbo</**artifactId**> </**dependency**> |   3 刷新maven依赖  4 配置consumer和service  服务端增加  @Service  客户端增加  @Reference代替@Autowired |

# 9 dubbo的项目配置



|  |
| --- |
| 1 将spring的@Service改成dubbo的@Service  2 配置dubbo的配置文件  *# dubbo中的服务名称* **spring.dubbo.application**= *# dubbo的通讯协议名称* **spring.dubbo.protocol.name**= *# zookeeper注册中心的地址* **spring.dubbo.registry.address**= *# zookeeper的通讯协议的名称* **spring.dubbo.registry.protocol**= *# dubbo的服务的扫描路径* **spring.dubbo.base-package**= |

客户端

|  |
| --- |
| 1 将@Autowired改为@Refernece    2 配置文件  *# dubbo的配置 # dubbo中的服务名称* **spring.dubbo.application**=**user-web** *# dubbo的通讯协议名称* **spring.dubbo.protocol.name**=**dubbo** *# zookeeper注册中心的地址* **spring.dubbo.registry.address**=**192.168.222.20:2181** *# zookeeper的通讯协议的名称* **spring.dubbo.registry.protocol**=**zookeeper** *# dubbo的服务的扫描路径* **spring.dubbo.base-package**=**com.atguigu.gmall** |

# 10 dubbo配置的注意事项

|  |
| --- |
| 1 spring的Service改为dubbo的service  2将service的autowired改为reference  3 dubbo在进行dubbo协议通讯时，需要实现序列化接口(封装的数据对象)  4 dubbo的consumer在三秒钟之内每间隔一秒进行一次重新访问，默认一秒钟超时，三次访问之后会直接抛超时异常，所以我们在开发阶段，可以将consumer设置的超时时间延长，方便断点调试  *# 设置超时时间* **spring.dubbo.consumer.timeout**=**600000** *# 设置是否检查服务存在* **spring.dubbo.consumer.check**=**false** |