1.前期设计

1.1.云服务

1. **Iaas**:(Infrastructure as a Service):基础设施即服务,提供给消费者的服务是对所有计算基础设施的利用.
2. **PaaS**:(Platform as a Service):平台即服务.提供给消费者的服务是把客户采用提供的开发语言和工具开花的或收购的应用程序部署到供应商的云计算基础设施上去.
3. **SaaS**:(Software as a Service):软件即服务,提供给消费者完整的软件解决方法,组织用于可以通过Internet连接到该应用.

面向企业的SaaS产品主要包括:CRM(客户关系管理),ERP(企业资源计划管理),线上视频或者群主通话会议,HRM(人力资源管理),OA(办公系统),外勤管理,财务管理,审批管理.

1.2.项目需求

国际物流的业务闭环涉及到海关,船东,港区,仓库,拖车,报关.

项目的重点围绕四个方面:

1. **权限管理**:对登录系统的员工进行细粒度的权限控制
2. **货物管理**:提供货物的全流程管理,包含商品的详情,报价
3. **报运管理**:包括购销合同,出口报运,装箱,委托,发票
4. **统计管理**:以图形化界面的方式对销售,财务数据进行展示

原型分析的概念是指在获取一组基本需求之后,快速的构造出一个反应用户需求的初始系统原型.让用户看到未来系统的概貌.

UML统一建模语言包含很多图形(用例图,类图,状态图)

用例图主要用来描述用户与用例之间的关联关系,说明的是谁要使用这个系统,以及他们使用该系统可以做些什么.

PowerDesigner可以方便对管理信息系统进行分析设计

1.3.系统架构

后端采用Spring + SpringMVC + Mybatis + Dubbo

前端采用AdminLTE

工程依赖结构是 common => domain => dao => system_service => manager_web

1.4.多租户数据设计方案

目前基于多租户的数据库设计方案通常有如下三种:

- 独立数据库
- 共享数据库,独立Schema:多个或所有的租户使用同一个数据库服务,但是每一个租户一个Schema.
- 共享数据库,共享数据表:租户共享一个Database,同一套数据库表,在表中增加租户ID等租户标志的字段,表明该记录是属于哪个租户的.

1.5.数据库的设计与建模

数据库设计的三范式及反三范式

1. 第一范式:确保每一列的原子性,即每列不可拆分
2. 第二范式:在第一范式的基础上,非主字段必须依赖于主字段.一个表只做一件事
3. 第三范式:在第二范式的基础上,消除传递依赖
4. **反三范式**:是基于第三范式所调整,为了提高运行效率,必须降低范式标准,适当保留冗余数据

2.企业及部门管理

企业管理只有saas平台管理员才能进行操作

2.1.抽取BaseController

@ModelAttribute注解,出现在方法上,表示当前方法会在控制器的方法执行之前,先执行

我们可以把@ModelAttribute特性,应用在BaseController中,所有的Controller继承BaseController,即可实现在调用Controller时,先执行@ModelAttribute中的方法,这样一来一些公共的内容就可以再调用方法之前帮我们进行初始化.

```
    protected User userInfo;
    protected String userName;
    protected String userId;
    protected String deptId;
    protected String deptName;
    protected Integer degree;

    @ModelAttribute
    public void init() {
        userInfo = (User) session.getAttribute("userInfo");
        if (userInfo != null) {
            companyId = userInfo.getCompanyId();
            companyName = userInfo.getCompanyName();
            degree = userInfo.getDegree();
            userName = userInfo.getUserName();
            userId = userInfo.getUserId();
            deptId = userInfo.getDeptId();
            deptName = userInfo.getDeptName();
        }
    }
```

2.2.PageHelper分页插件

PegeHelper式国内优秀的开源的mybatis分页插件

```
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper</artifactId>
    <version>${pagehelper.version}</version>
</dependency>
```

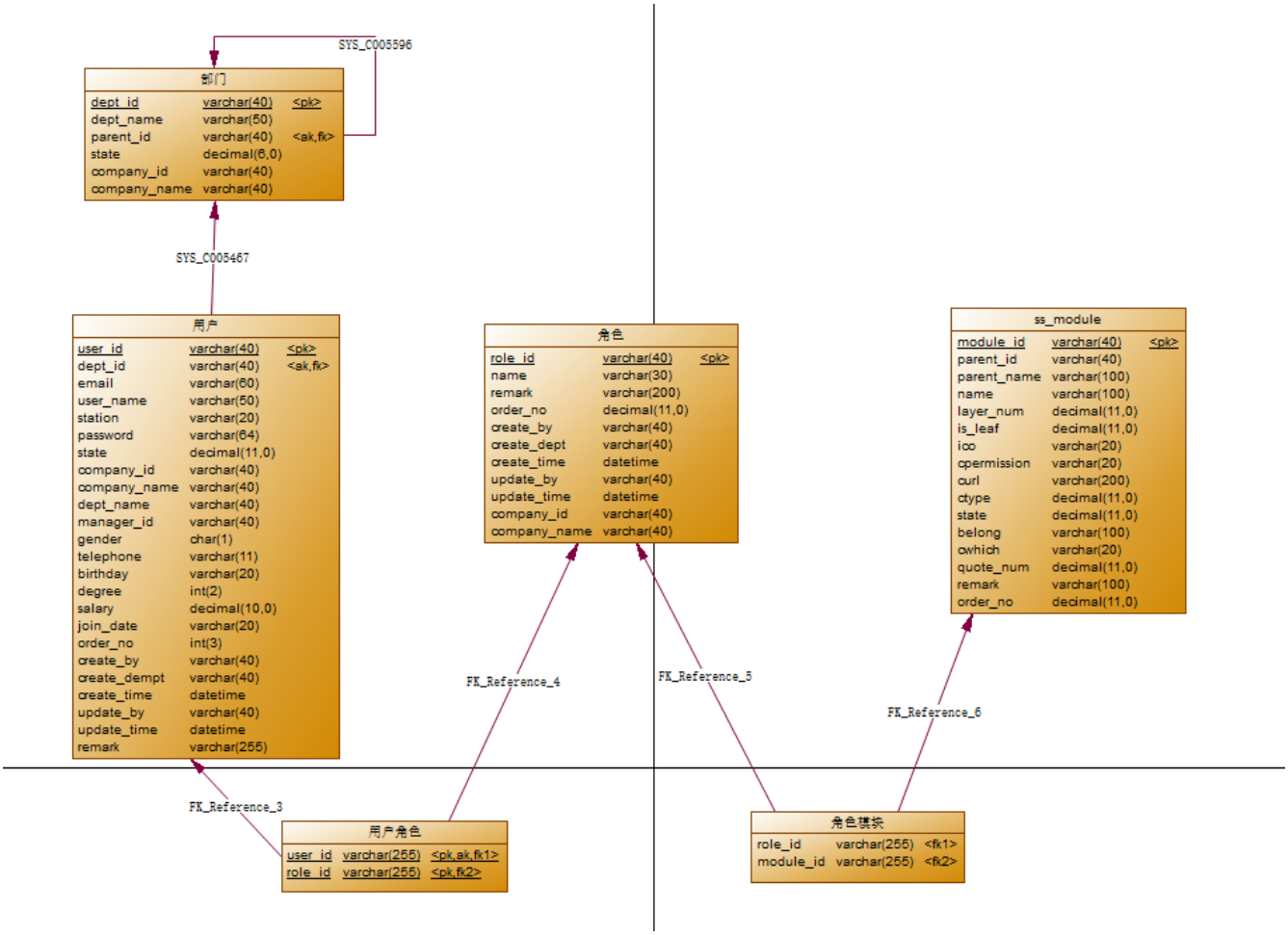
```
<context:property-placeholder location="classpath:/properties/db.properties"/>

<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
    <property name="driverClassName" value="${jdbc.driver}"/>
    <property name="url" value="${jdbc.url}"/>
    <property name="username" value="${jdbc.username}"/>
    <property name="password" value="${jdbc.password}"/>
</bean>

<bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="plugins">
        <array>
            <bean class="com.github.pagehelper.PageInterceptor">
                <property name="properties">
                    <!-- config params as the following -->
                    <props>
                        <prop key="helperDialect">mysql</prop>
                    </props>
                </property>
            </bean>
        </array>
    </property>
</bean>
```

1.传递两个参数,一个是当前页码,另一个是每页查询条数.

2.自动对PageHelper.startPage方法下的第一个sql查询进行分页.

3.RBAC权限模型

RBAC基于角色的权限访问控制,再RBAC中,权限与角色相关联,用户通过成为适当的角色的成员而得到这角色的权限



在分配角色和模块权限的时候,可以使用前端Ztree树的插件,前端Ztree树插件需要的是Json格式的数据,我们将数据放在List集合中,通过SpringMVC的@ResponseBody注解自动将List集合转为Json格式的数据.zTree需要的数据格式是这样的,在模块的数据库的设计中,是有module_id和parent_id的,正好匹配上了ztree的插件要求,

```
var zNodes =[
    { id:1, pId:0, name:"随意勾选 1", open:true},
    { id:11, pId:1, name:"随意勾选 1-1", open:true},
    { id:111, pId:11, name:"随意勾选 1-1-1"},
    { id:112, pId:11, name:"随意勾选 1-1-2"},
    { id:12, pId:1, name:"随意勾选 1-2", open:true},
    { id:121, pId:12, name:"随意勾选 1-2-1"},
    { id:122, pId:12, name:"随意勾选 1-2-2"},
    { id:2, pId:0, name:"随意勾选 2", checked:true, open:true},
    { id:21, pId:2, name:"随意勾选 2-1"},
    { id:22, pId:2, name:"随意勾选 2-2", open:true},
    { id:221, pId:22, name:"随意勾选 2-2-1", checked:true},
    { id:222, pId:22, name:"随意勾选 2-2-2"},
    { id:23, pId:2, name:"随意勾选 2-3"}
];
```

```
    @Override
    public LinkedList findModuleByRoleForAjax(String roleid, String companyId) {
        List<Module> modulesByRole = roleDao.findModluesByRole(roleid);
        List<Module> modules = moduleDao.findAll();
```

```
        LinkedList<Map<String,Object>> list = new LinkedList<>();
        for (Module module : modules) {
            HashMap<String, Object> map = new HashMap<>();
            //{ id:2, pId:0, name:"随意勾选 2", checked:true, open:true}
            map.put("id",module.getId());
            map.put("pId",module.getParentId());
            map.put("name",module.getName());
            //如果通过角色查出来的模块包含遍历的模块,那么就将添加属性checked="true"
            if(modulesByRole.contains(module)){
                map.put("checked",true);
            }
            list.add(map);
        }
        return list;
    }
```

前端提交数据Ztree数据,是通过tree对象调用getCheckedNodes()得到勾选的id数组,再用字符串将数组中的id拼接起来,然后存放在form表单的隐藏域中,通过提交表达将字符串提交过去.后端接收到字符串之后,在service层进行字符串切割,先将该角色id对应的中间表全部删除,再添加新的模块.

对于用户分配角色和角色分配模块是一样的操作.

```
    @Override
    public void updateRoleModule(String roleid, String moduleIds) {
        //根据roleid去数据库中删除该roleid的数据
        roleDao.deleteOnRoleModule(roleid);
        String[] strings = moduleIds.split(",");
        for (String string : strings) {
            roleDao.addOnRoleModule(roleid,string);
        }
    }
```

4.用户登录

说明:SaaS-Export系统,用户初步分为3种.第一种是SaaS平台的系统管理员,第二种是企业管理员,第三种是企业员工.系统管理员可以操作企业的一些信息,企业管理员可以操作企业下的所有操作,员工则需要通过RBAC模型查找出他所能操作的模块.

下面是用户登录之后,对用户所拥有模块进行查询.

```
    //根据用户信息确定能够使用的模块
    @Override
    public List<Module> findModuleByUid(User user1) {
        Integer degree = user1.getDegree();
        //当degree为0的时候,表示是系统管理员
        if(degree==0){
            //系统管理员的话可以操作Saas管理,企业管理,和模块管理
            return moduleDao.findByBelong(0);
        }else if(degree==1){
            //表明是企业管理员,可以操作企业的所有内容
            return moduleDao.findByBelong(1);
```

```
        }else{
            return moduleDao.findModuleByUid(user1.getUserId());
        }
    }
```

5.记录日志

首先开启AOP对注解的支持

```
<aop:aspectj-autoproxy/>
```

```
    //AOP
    @Around("pt1()")
    public Object saveLog(ProceedingJoinPoint proceedingJoinPoint) {
        System.out.println("进入切面");
        Object result = null;
        User userInfo = (User) session.getAttribute("userInfo");
        try {
            Object[] args = proceedingJoinPoint.getArgs();
            //获取签名
            MethodSignature signature =
(MethodSignature)proceedingJoinPoint.getSignature();
            Method method = signature.getMethod();
            if (method.isAnnotationPresent(RequestMapping.class)){
                RequestMapping requestMapping = method.getAnnotation(RequestMapping.class);
                /*String path = requestMapping.path()[0];*/
                String name = requestMapping.name();
                Log log = new Log();
                log.setAction(requestMapping.name());
                log.setCompanyId(userInfo.getCompanyId());
                log.setId(UUID.randomUUID().toString().replace("-",""));
                log.setUserName(userInfo.getUserName());
                log.setTime(new Date());
                log.setCompanyName(userInfo.getCompanyName());
                log.setMethod(method.getName());
                log.setIp(request.getRemoteAddr());
                logService.save(log);
            }
            result = proceedingJoinPoint.proceed(args);
            return result;
        } catch (Throwable throwable) {
            throwable.printStackTrace();
            return result;
        }
    }
```

记录日志,采用的是SpringAOP的环绕通知,再session中获取user信息,通过proceedingJoinPoint获取签名,再通过签名获取Method,通过判断方法上有没有@RequestMapping这个注解来进入记录日志方法,再通过Method得到方法上的注解,再通过注解得到@requestMapping的name,就可以知道用户在操作什么.

6.Shiro安全框架

6.1.配置

导入工程依赖

```
    <!--shiro和spring整合-->
    <dependency>
        <groupId>org.apache.shiro</groupId>
        <artifactId>shiro-spring</artifactId>
        <version>1.3.2</version>
    </dependency>

    <!--shiro核心包-->
    <dependency>
        <groupId>org.apache.shiro</groupId>
        <artifactId>shiro-core</artifactId>
        <version>1.3.2</version>
    </dependency>
```

web.xml的配置

```
        <!--spring提供的shiro代理对象-->
    <filter>
        <!--注意：filter-name一定要与shiro配置中的id值一致！-->
        <filter-name>shiroFilter</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
        <init-param>
            <param-name>targetFilterLifecycle</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>shiroFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

spring整合shiro

```
<!--安全管理器-->
    <bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
        <!--引用自定义的Realm-->
        <property name="realm" ref="authRealm"/>
    </bean>

    <!--自定义Realm域的编写-->
    <bean id="authRealm" class="com.fechin.web.shiro.AuthRealm">
        <!--注入自定义的密码比较器-->
        <property name="credentialsMatcher" ref="customCredentialsMatcher"/>
    </bean>
```

```
    <!--密码比较器,密码加密比较-->
    <bean id="customCredentialsMatcher"
class="com.fechin.web.shiro.CustomCredentialsMatcher"/>


    <!-- filter-name这个名字的值来自于web.xml中filter的名字 -->
    <bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
        <property name="securityManager" ref="securityManager"/>
        <!--登录页面  -->
        <property name="loginUrl" value="/login.jsp"></property>
        <!-- 授权失败后 -->
        <property name="unauthorizedUrl" value="/unauthorized.jsp"></property>

        <property name="filterChainDefinitions">
            <!-- /**代表下面的多级目录也过滤 -->
            <value>
                /system/module/* = perms["模块管理"]
                /company/** = perms["企业管理"]
                /index.jsp* = anon
                /login.jsp* = anon
                /login* = anon
                /logout* = anon
                /css/** = anon
                /img/** = anon
                /plugins/** = anon
                /make/** = anon
                /** = authc
                /*.* = authc
            </value>
        </property>
    </bean>


    <!-- 保证实现了Shiro内部lifecycle函数的bean执行 -->
    <bean id="lifecycleBeanPostProcessor"
class="org.apache.shiro.spring.LifecycleBeanPostProcessor"/>

    <!-- 生成代理，通过代理进行控制 -->
    <bean class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"
          depends-on="lifecycleBeanPostProcessor">
        <property name="proxyTargetClass" value="true"/>
    </bean>

    <!-- 安全管理器 -->
    <bean
class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">
        <property name="securityManager" ref="securityManager"/>
    </bean>

    <aop:aspectj-autoproxy proxy-target-class="true"/>
```

6.2.Authentication(用户校验)

编写登陆功能:

```
        @RequestMapping("/login")
    public String login(User user, Model model) {
        //在数据库查询之前先进行判定
        if (user.getEmail() == null) {
            model.addAttribute("error", "email不能为空");
            return "forward:/login.jsp";
        }
        if (user.getPassword() == null) {
            model.addAttribute("error", "密码不能为空");
            return "forward:/login.jsp";
        }
        //前面的判断非空判断保留
        Subject subject = SecurityUtils.getSubject();
        UsernamePasswordToken token = new UsernamePasswordToken(user.getEmail(),
user.getPassword());
        try {
            subject.login(token);
        } catch (AuthenticationException e) {
            e.printStackTrace();
            model.addAttribute("error", "用户名或密码错误");
            return "forward:/login.jsp";
        }

        User user1 = (User) subject.getPrincipal();
        session.setAttribute("userInfo", user1);
        List<Module> modules = moduleService.findModuleByUid(user1);
        for (Module module : modules) {
            System.out.println(module);
        }
        session.setAttribute("modules", modules);
        return "home/main";

    }
```

首先通过SecurityUtils获取subject,创建usernamePasswordToken,将username和password通过subject传过去,假如密码不匹配,login这个方法就会报错,通过捕获AuthenticationException异常来捕获该异常,如果异常捕获到,则跳转到登陆界面,并告知用户名或密码错误,否则就登陆成功,并进行RBAC的查询,将modules传递给前端,让前端控制显示的按钮.

```
public class AuthRealm extends AuthorizingRealm {
    @Autowired
    private UserService userService;
    @Autowired
    private ModuleService moduleService;

    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws
```

```
AuthenticationException {
        UsernamePasswordToken usernamePasswordToken = (UsernamePasswordToken) token;
        String email = usernamePasswordToken.getUsername();
        //String password = new String(usernamePasswordToken.getPassword());
        User user = userService.findByEmail(email);
        if (user == null) {
            return null;
        }
        SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(user,
user.getPassword(), this.getName());
        return info;
    }

    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
        //通过principals获取到User
        User user = (User) principals.getPrimaryPrincipal();

        List<Module> moduleByUid = moduleService.findModuleByUid(user);
        HashSet<String> set = new HashSet<>();
        SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
        for (Module module : moduleByUid) {
            set.add(module.getName());
        }
        authorizationInfo.setStringPermissions(set);
        return authorizationInfo;
    }
}
```

获得AuthenticationToken之后,将token强转成usernamePasswordToken,通过usernamePasswordToken获取到用户名,通过用户名去数据库中搜索用户,如果用户搜索不到,直接返回null,返回null给controller层,subject.login()就会抛异常,登陆失败.,如果用户存在,则创建new AuthenticationInfo(principal,credentials,realmName),并返回.

```
    * @param principal   the 'primary' principal associated with the specified realm.
    * @param credentials the credentials that verify the given principal.
    * @param realmName   the realm from where the principal and credentials were acquired.
    */
  public SimpleAuthenticationInfo(Object principal, Object credentials, String realmName) {
      this.principals = new SimplePrincipalCollection(principal, realmName);
      this.credentials = credentials;
  }
```

```
public class CustomCredentialsMatcher extends SimpleCredentialsMatcher {
    @Override
    public boolean doCredentialsMatch(AuthenticationToken token, AuthenticationInfo info) {
        UsernamePasswordToken token1 = (UsernamePasswordToken) token;
        String email = token1.getUsername();
        String password = new String(token1.getPassword());
        String mdPassword = Encrypt.md5(password, email);
        String credentials = (String) info.getCredentials();
        if (mdPassword.equals(credentials)){
            return true;
        }
        return false;
```

```
    }
}
```

自定义密码比较器,从token中获取用户输入的密码,从authenticationInfo中获取credentials,也就是密码,将token中用户输入的密码进行与添加用户的同样的加密方式进行加密,然后与creadentials进心比较,如果比较成功返回true,比较失败返回false.

6.3.Authorization(授权)

```
@Override
protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
    //通过principals获取到User
    User user = (User) principals.getPrimaryPrincipal();

    List<Module> moduleByUid = moduleService.findModuleByUid(user);
    HashSet<String> set = new HashSet<>();
    SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
    for (Module module : moduleByUid) {
        set.add(module.getName());
    }
    authorizationInfo.setStringPermissions(set);
    return authorizationInfo;
}
```

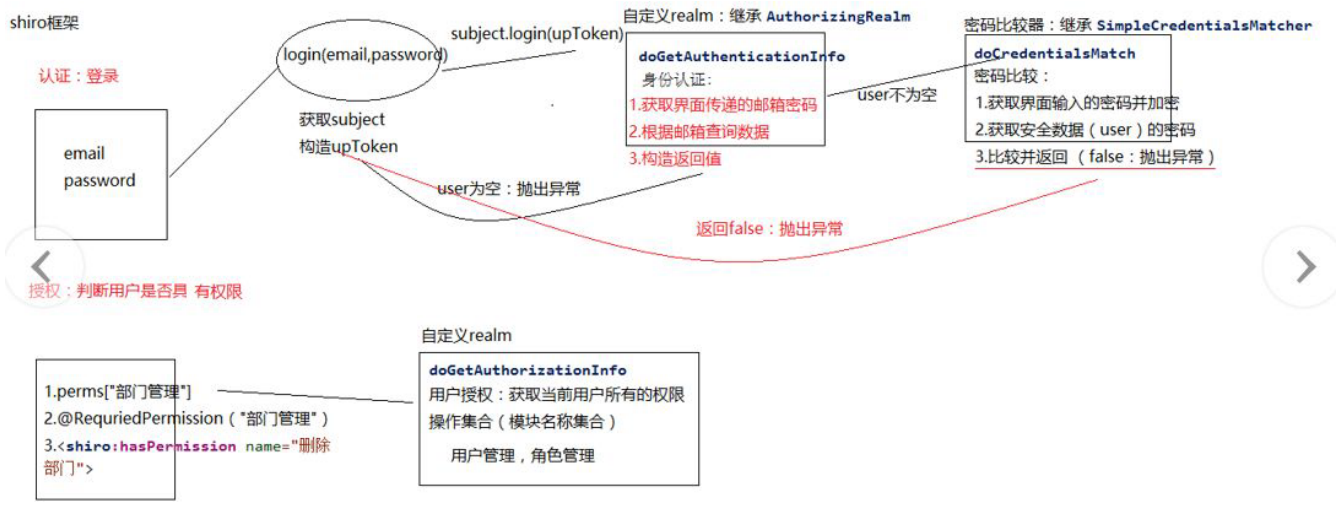
通过principal获取到user,通过user将module查询出来,将module的名字放在set集合中,authorizationInfo.setStringPermissions(set)将set集合返回到Security Manager.

```
<!-- filter-name这个名字的值来自于web.xml中filter的名字 -->
    <bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
        <property name="securityManager" ref="securityManager"/>
        <!--登录页面  -->
        <property name="loginUrl" value="/login.jsp"></property>
        <!-- 授权失败后 -->
        <property name="unauthorizedUrl" value="/unauthorized.jsp"></property>

        <property name="filterChainDefinitions">
            <!-- /**代表下面的多级目录也过滤 -->
            <value>
                /system/module/* = perms["模块管理"]
                /company/** = perms["企业管理"]
                /index.jsp* = anon
                /login.jsp* = anon
                /login* = anon
                /logout* = anon
                /css/** = anon
                /img/** = anon
                /plugins/** = anon
                /make/** = anon
                /** = authc
                /*.* = authc
            </value>
```

```
        </property>
    </bean>
```

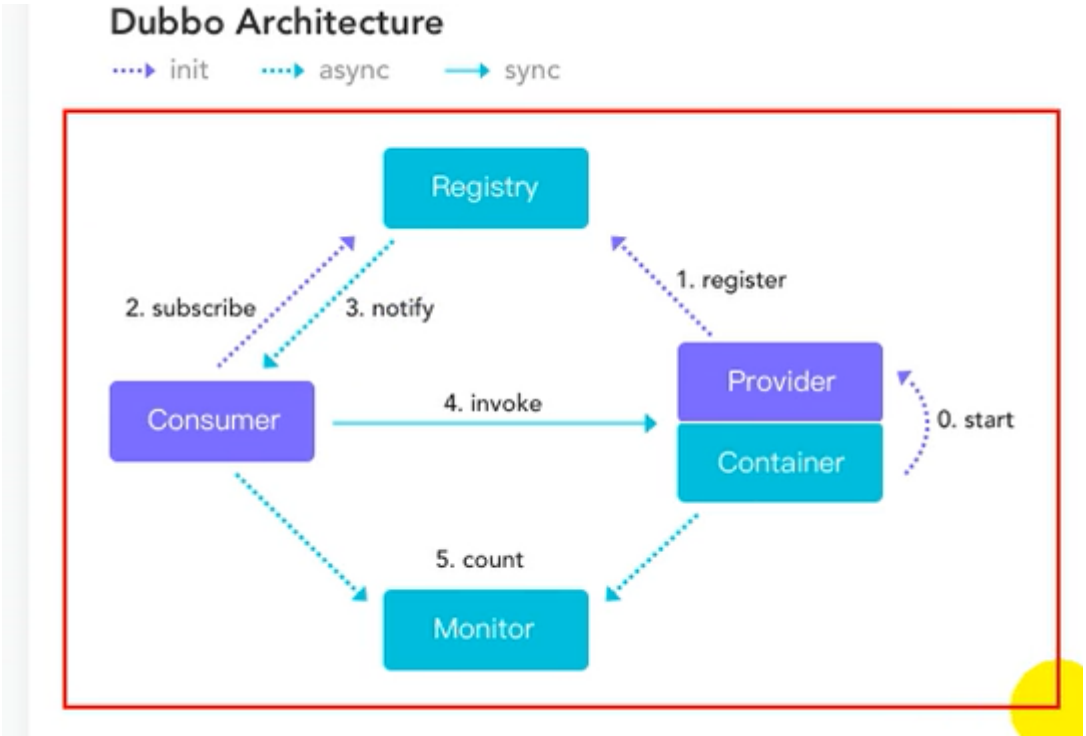
上述的anno表示的是不需要登录即可进行访问,authc表示登陆成功就可以进行访问,perms表示必须具备某种权限才可以访问.

6.4.执行步骤



7.Dubbo RPC框架

Apache Dubbo是一款高性能的Java RPC框架,前身是alibaba开源的高性能,轻量级的开源Java RPC框架,RPC全称为remote procedure call,是远程过程调用.



配置pom文件

```
    <!-- dubbo相关 -->
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>dubbo</artifactId>
        <version>2.6.6</version>
    </dependency>
    <dependency>
        <groupId>io.netty</groupId>
        <artifactId>netty-all</artifactId>
        <version>4.1.32.Final</version>
    </dependency>
    <dependency>
        <groupId>org.apache.curator</groupId>
        <artifactId>curator-framework</artifactId>
        <version>4.0.0</version>
        <exclusions>
            <exclusion>
                <groupId>org.apache.zookeeper</groupId>
                <artifactId>zookeeper</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <dependency>
        <groupId>org.apache.zookeeper</groupId>
        <artifactId>zookeeper</artifactId>
        <version>3.4.7</version>
    </dependency>
    <dependency>
        <groupId>com.github.sgroschupf</groupId>
        <artifactId>zkclient</artifactId>
        <version>0.1</version>
    </dependency>
```

在项目中dubbo是面向接口的,我们将service层提取出来,service层和web层共同依赖service的接口层.

7.1 Dubbo的Provider

在service层中我们需要编写dubbo的xml配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
```

```
        http://code.alibabatech.com/schema/dubbo
http://code.alibabatech.com/schema/dubbo/dubbo.xsd
        http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <!--当前应用的名称,名称不能和其它服务名称相同-->
    <dubbo:application name="export_cargo_service"/>

    <!--给当前服务提供可访问的注册中心地址-->
    <dubbo:registry address="zookeeper://127.0.0.1:2181" />

    <!--服务提供者,提供该服务的端口号,该端口号给dubbo进行查找-->
    <!--给消费提供可访问的地址:默认是20880-->
    <dubbo:protocol name="dubbo" port="20886"/>

    <!--开启dubbo的注解扫描-->
    <dubbo:annotation package="com.fechin.service"/>

</beans>
```

在dubbo的配置文件中,我们将该服务注册给注册中心Zookeeper,并提供该服务的端口号,不提供的话默认是20880,该端口号是给dubbo进行查找.并开启dubbo的注解扫描,这样service层中的类就可以使用Dubbo提供的@Service注解,该注解用来实现被远程调用.

该服务我么可以直接放在一个tomcat服务器中,通过监听器的方式读取该文件.

同样我们也可以使用一段java测试代码,读取该文件

```
public class CompanyProviderTest {
    public static void main(String[] args) throws Exception{
        ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("classpath*:spring/applicationContext-*.xml");
        context.start();//启动服务
        System.in.read();//任意键退出
    }
}
```

7.2 Dubbo的Consumer

我们将dubbo的配置文件直接放在springMVC的配置文件中

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/context
```

```
  http://www.springframework.org/schema/context/spring-context.xsd
  http://www.springframework.org/schema/aop
  http://www.springframework.org/schema/aop/spring-aop.xsd
  http://www.springframework.org/schema/tx
  http://www.springframework.org/schema/tx/spring-tx.xsd
  http://code.alibabatech.com/schema/dubbo
  http://code.alibabatech.com/schema/dubbo/dubbo.xsd
  http://www.springframework.org/schema/mvc
  http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <context:component-scan base-package="com.fechin.web"/>
    <mvc:annotation-driven/>
    <bean id="internalResourceViewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/"/>
        <property name="suffix" value=".jsp"/>
    </bean>

    <aop:aspectj-autoproxy/>
    <!--开启AOP对注解的支持-->

    <!--<mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/**"/>
            <mvc:exclude-mapping path="/login.do"/>
            <mvc:exclude-mapping path="/index.do"/>
            <bean id="isLoginInterceptor"
class="com.fechin.web.interceptors.IsLoginInterceptor"/>
        </mvc:interceptor>
    </mvc:interceptors>-->

    <bean id="customExceptionResolver"
class="com.fechin.web.exceptions.CustomExceptionResolver"/>

    <!--当前应用的名称,名称不能和其它服务名称相同-->
    <dubbo:application name="export_manager_web"/>

    <!--给当前服务提供可访问的注册中心地址-->
    <dubbo:registry address="zookeeper://127.0.0.1:2181" />

    <!--开启dubbo的注解扫描-->
    <dubbo:annotation package="com.fechin.web.controller"/>

    <!--在配置文件中加载qiniu.properties文件-->
    <context:property-placeholder location="classpath:properties/qiniu.properties"/>

    <!--id是确定-->
    <bean id="multipartResolver"
          class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <property name="maxUploadSize">
            <value>5242880</value>
        </property>
    </bean>
</beans>
```

之后通过Dubbo提供的@Reference注解去调用service层.

8.MyBatis逆向工程

8.1.jdbc的properties文件

```
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/saas-export
jdbc.username=root
jdbc.password=320512
```

8.2.mybatis-generator-config.xml配置文件

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE generatorConfiguration PUBLIC
        "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"
        "http://mybatis.org/dtd/mybatis-generator-config_1_0.dtd" >
<generatorConfiguration>
    <!--导入属性配置-->
    <properties resource="jdbc.properties"></properties>
    <!--指定特定数据库的jdbc驱动jar包的位置-->
    <!--<classPathEntry location="${jdbc.path}"/>-->

    <context id="context" targetRuntime="MyBatis3">
        <!-- 是否去除自动生成的注释 true: 是 ： false:否 -->
        <commentGenerator>
            <property name="suppressAllComments" value="true"/>
            <property name="suppressDate" value="true"/>
        </commentGenerator>

        <!--数据库连接的信息：驱动类、连接地址、用户名、密码 -->
        <jdbcConnection driverClass="${jdbc.driver}"
                        connectionURL="${jdbc.url}"
                        userId="${jdbc.username}"
                        password="${jdbc.password}"/>
        <!-- 默认false，把JDBC DECIMAL 和 NUMERIC 类型解析为 Integer，为 true时把JDBC DECIMAL 和
decimal(10,2)
            NUMERIC 类型解析为java.math.BigDecimal -->
        <javaTypeResolver>
            <property name="forceBigDecimals" value="false"/>
        </javaTypeResolver>

        <!--指定包名生成实体类 以及生成的地址 （可以自定义地址，如果路径不存在会自动创建） -->
        <javaModelGenerator targetPackage="com.fechin.domain.export"
targetProject=".\src\main\java">
            <!-- enableSubPackages:是否让schema作为包的后缀 -->
            <property name="enableSubPackages" value="false"/>
            <!-- 从数据库返回的值被清理前后的空格 -->
            <property name="trimStrings" value="true"/>
        </javaModelGenerator>
```

```
        <!--Mapper映射文件生成所在的目录 为每一个数据库的表生成对应的mapper文件 -->
        <sqlMapGenerator targetPackage="com.fechin.dao.export"
targetProject=".\src\main\resources">
            <!-- enableSubPackages:是否让schema作为包的后缀 -->
            <property name="enableSubPackages" value="false"/>
        </sqlMapGenerator>

        <!-- 客户端代码，生成易于使用的针对Model对象和XML配置文件 的代码
                type="ANNOTATEDMAPPER",生成Java Model 和基于注解的Mapper对象
                type="MIXEDMAPPER",生成基于注解的Java Model 和相应的Mapper对象
                type="XMLMAPPER",生成SQLMap XML文件和独立的Mapper接口
        -->
        <javaClientGenerator targetPackage="com.fechin.dao.export"
                             targetProject="src/main/java" type="XMLMAPPER">
            <!-- enableSubPackages:是否让schema作为包的后缀 -->
            <property name="enableSubPackages" value="false"/>
        </javaClientGenerator>

        <!-- 指定数据库表 -->
        <table schema="saas-export" tableName="co_export" domainObjectName="Export"
mapperName="ExportDao"
               enableCountByExample="false" enableDeleteByExample="false"
               enableSelectByExample="true" enableUpdateByExample="false"/>
        <!-- 指定数据库表 -->
        <table schema="saas-export" tableName="co_export_product"
domainObjectName="ExportProduct" mapperName="ExportProductDao"
               enableCountByExample="false" enableDeleteByExample="false"
               enableSelectByExample="true" enableUpdateByExample="false"/>
        <!-- 指定数据库表 -->
        <table schema="saas-export" tableName="co_ext_eproduct"
domainObjectName="ExtEproduct" mapperName="ExtEproductDao"
               enableCountByExample="false" enableDeleteByExample="false"
               enableSelectByExample="true" enableUpdateByExample="false"/>
    </context>
</generatorConfiguration>
```

8.3.编写MyBatis逆向工程代码

通过执行MyBatis逆向工程代码,生成对应xml中对应的实体,接口和mybatis映射文件

```
public class MybatisGenerator {
    public void generator() throws Exception{
        List<String> warnings = new ArrayList<String>();
        boolean overwrite = true;
        //指定 逆向工程配置文件
        InputStream in =
MybatisGenerator.class.getClassLoader().getResourceAsStream("mybatis-generator-config.xml");
        ConfigurationParser cp = new ConfigurationParser(warnings);
        Configuration config = cp.parseConfiguration(in);
        DefaultShellCallback callback = new DefaultShellCallback(overwrite);
        MyBatisGenerator myBatisGenerator = new MyBatisGenerator(config,
```

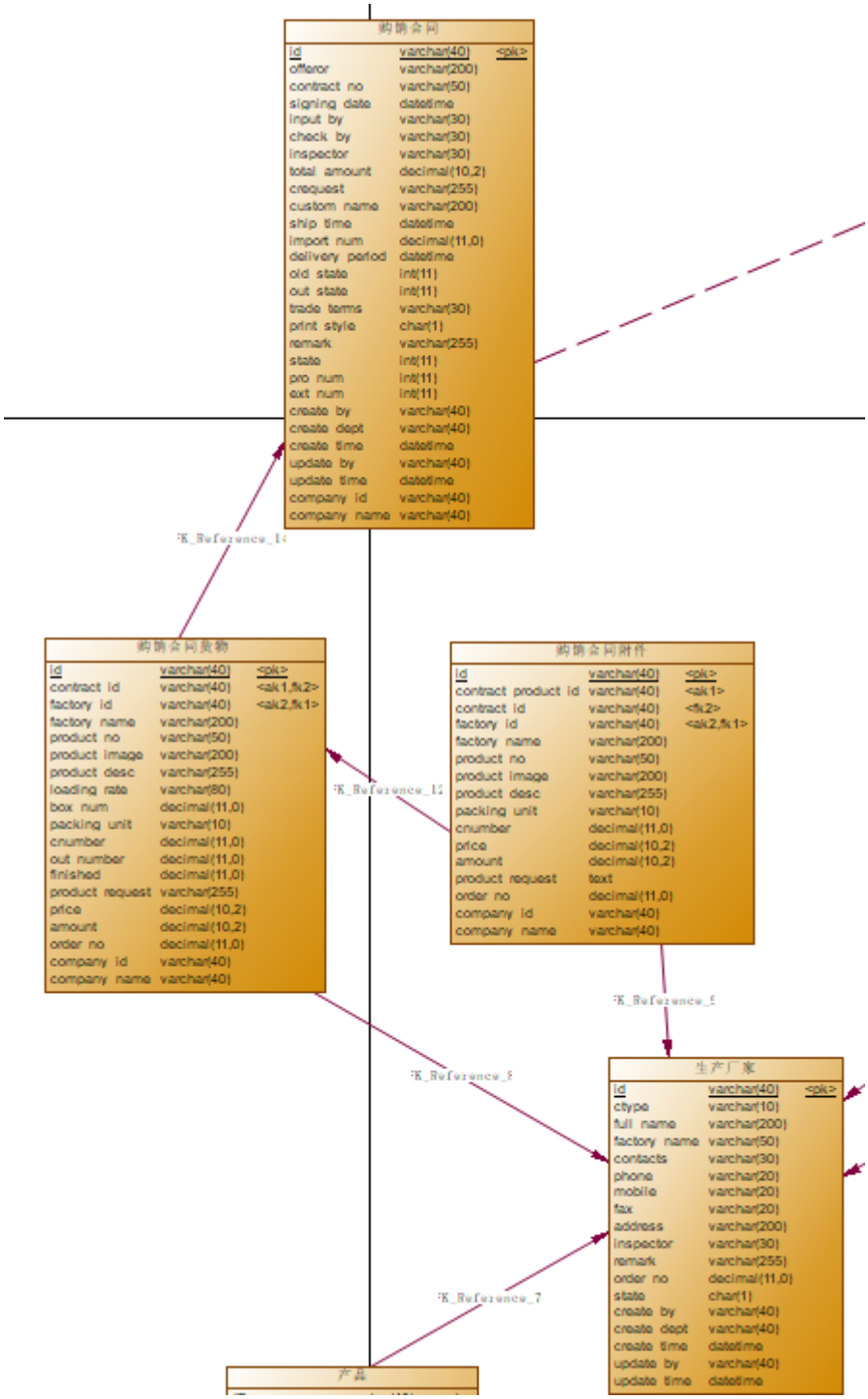
```
                callback, warnings);
        myBatisGenerator.generate(null);
        in.close();

    }
    public static void main(String[] args) throws Exception {
        try {
            MybatisGenerator generatorSqlmap = new MybatisGenerator();
            generatorSqlmap.generator();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

9.购销合同管理

公司销售和海外何苦签订合同,客户订单中的货物,共就联系这些厂家来生产,和生产厂家签订的合同就叫做"购销合同".

再购销合同的数据库设计上就用来反三范式的思想,如购销合同的总金额就是一个冗余字段,需要再平时添加货物和添加附件的时候,需要去更新购销合同中的总金额.

这其实也是分散计算思想的一种应用,如果没有分散计算,再查询购销合同的总金额时候,就需要管理加载购销合同下的所有货物,并要加载所有货物下的所有附件,这样实现购销合同总金额查询所消耗的时间太长.可以再平时添加货物的时候,添加附件的时候,分别计算出货物,附件的总金额,再更新购销合同的总金额.



9.1.购销合同(细粒度权限控制)

```
    @RequestMapping(path = "/list", name = "查找所有合同信息")
public String list(@RequestParam(defaultValue = "1") Integer page,
                   @RequestParam(defaultValue = "5") Integer size,
                   Model model) {
    //查询之前,设定好查找的example和排序
    ContractExample contractExample = new ContractExample();
    ContractExample.Criteria criteria = contractExample.createCriteria();
    criteria.andCompanyIdEqualTo(companyId);
    contractExample.setOrderByClause("update_time desc");
```

```
        //在合同的入口进行细粒度的权限控制
        /**
         * 1:是系统管理员,2:管理所有下属部门和人员,3:管理本部门,4:普通员工
         * 如果是普通员工,在查询时候,只能查询他所创建的合同
         * 如果是部门经理,查询的时候,只能查询到他所在部门的所有合同
         * 如果是管理层,能够查找到他部门及其子部门的所有合同
         */
        if(degree==4){
            criteria.andCreateByEqualTo(userId);
        }else if(degree==3){
            criteria.andCreateDeptEqualTo(deptId);
        }else if(degree==2){
            criteria.andCreateDeptLike(deptId+"%");
        }

        PageInfo<Contract> pageInfo = contractService.findByCompanyIdOnPages(page, size,
contractExample);
        model.addAttribute("page", pageInfo);
        return "cargo/contract/contract-list";

    }
```

查看合同:在合同的入口进行细粒度的权限控制,如果是普通员工,在查询的时候,只能操作他所创建的合同,根据创建者id进行查询.如果是部门经理,查询的时候,根据他所在部门的所有合同,如果是管理层能够查找到他部门及其子部门的所有合同.因为多租户数据库设计方案是所有的用户都共用同一张表,所以在参看合同的时候,需要加上companyId

添加合同,**删除合同**,**修改合同**.

9.2. 购销合同货物

查看合同货物:通过前端传入的合同id进行分页查询,在前端我们将添加货物和查看合同做在同一张页面,我们将前端所需要的分页之后的合同货物列表,商家的下拉框,和contractId都存放在ModelAndView中发送给前端页面.

添加合同货物:添加货物我们直接在前端页面输入相关数据信息,然后进入controller,该controller方法可以作为添加货物或者更新货物同时使用,具体怎么使用是通过传递过来的货物有没有货物id,如果有没有货物id,那么可以知道是新增加的货物,如果有货物id,可以知道是修改的货物.

```
        @RequestMapping(path = "/edit", name = "添加货物或更新货物")
    public String edit(ContractProduct contractProduct, MultipartFile productPhoto) throws
Exception {
        contractProduct.setCompanyName(companyName);
        contractProduct.setCompanyId(companyId);
        if (contractProduct.getId() == null) {
            if (productPhoto!=null){
                String img = "http://"+fileUpLoadUtils.upload(productPhoto);
                //这边返回的是域名/图片名称
                contractProduct.setProductImage(img);
            }
            contractProductService.save(contractProduct);
            return "redirect:/cargo/contractProduct/list.do?contractId=" +
contractProduct.getContractId();
```

```
        } else {
            contractProductService.update(contractProduct);
            return "redirect:/cargo/contractProduct/list.do?contractId=" +
contractProduct.getContractId();
        }
    }
```

在service层,添加货物,首先是将货物设定它的id,然后获取货物的价格和数量,如果货物的价格和数量都不为空,那么进行相乘计算出总价.通过该货物的合同id调用出合同的相关数据,对合同的总价和合同的货物数量进行更改,

```
@Override
public void save(ContractProduct contractProduct) {
    //Long boxNum = contractProduct.getBoxNum();
    //获取单价
    contractProduct.setId(UUID.randomUUID().toString().replace("-", ""));
    BigDecimal price = contractProduct.getPrice();
    //获取数量

    Long cnumber = contractProduct.getCnumber();
    BigDecimal amountB = new BigDecimal(0);
    if (price != null && cnumber != null) {
        //只有两个都不为空,才进行如下的操作
        BigDecimal cnumberB = new BigDecimal(cnumber);
        amountB = cnumberB.multiply(price);
    }
    contractProduct.setAmount(amountB);
    //接着需要与该商品有关的合同的数据
    String contractId = contractProduct.getContractId();
    Contract contract = contractDao.selectByPrimaryKey(contractId);
    //查完合同数据之后,获取合同总价,相加
    BigDecimal totalAmount = contract.getTotalAmount();
    if (totalAmount == null) {
        totalAmount = new BigDecimal(0);
    }
    BigDecimal addTotalAmount = totalAmount.add(amountB);
    //获取合同的商品Num,加一
    Integer proNum = contract.getProNum();
    int i = proNum + 1;
    //获取该合同的id
    String id = contract.getId();
    //新建一份合同,selective进行更新
    Contract contract1 = new Contract();
    contract1.setId(id);
    contract1.setProNum(i);
    contract1.setTotalAmount(addTotalAmount);
    //根据已有的数据进行更新
    contractDao.updateByPrimaryKeySelective(contract1);
    contractProductDao.insertSelective(contractProduct);
}
```

修改合同货物:修改合同货物的业务逻辑与添加合同货物的业务逻辑类似.但修改合同的业务逻辑中,需要根据货物id将原货物的信息查找出来,原数量,原单价,原总价,这样在更新合同的时候,就可以先减去旧数量和旧总价,再加上新数量和新总价.

删除合同货物:首先根据货物的id查询该货物的所有附件,将附件总的所有价格循环相加,然后将附件全部删除,接着对合同数据进行更新.

```
@Override
public void delete(String id, String contractId) {
    //根据id查询contractProduct
    ContractProduct contractProduct = contractProductDao.selectByPrimaryKey(id);

    //获取到它的附件
    List<ExtCproduct> extCproducts = contractProduct.getExtCproducts();
    BigDecimal tempAmount = new BigDecimal(0);
    for (ExtCproduct extCproduct : extCproducts) {
        BigDecimal amount = extCproduct.getAmount();
        //每循环一次,就加一次
        tempAmount = tempAmount.add(amount);
        extCproductDao.deleteByPrimaryKey(extCproduct.getId());
    }
    //总的被删除价格
    BigDecimal totalAmountWillBeDeleted = tempAmount.add(contractProduct.getAmount());

    //接着需要与该商品有关的合同的数据
    Contract contract = contractDao.selectByPrimaryKey(contractId);
    BigDecimal oldTotalAmount = contract.getTotalAmount();
    Integer oldProNum = contract.getProNum();
    Integer oldExtNum = contract.getExtNum();

    //新建一份合同,selective进行更新
    Contract contract1 = new Contract();
    contract1.setId(contractId);
    contract1.setTotalAmount(oldTotalAmount.subtract(totalAmountWillBeDeleted));
    contract1.setProNum(oldProNum - 1);
    contract1.setExtNum(oldExtNum - extCproducts.size());

    //对合同进行更新
    contractDao.updateByPrimaryKeySelective(contract1);
    //删除该货物
    contractProductDao.deleteByPrimaryKey(id);
}
```

9.3. 购销合同货物附件

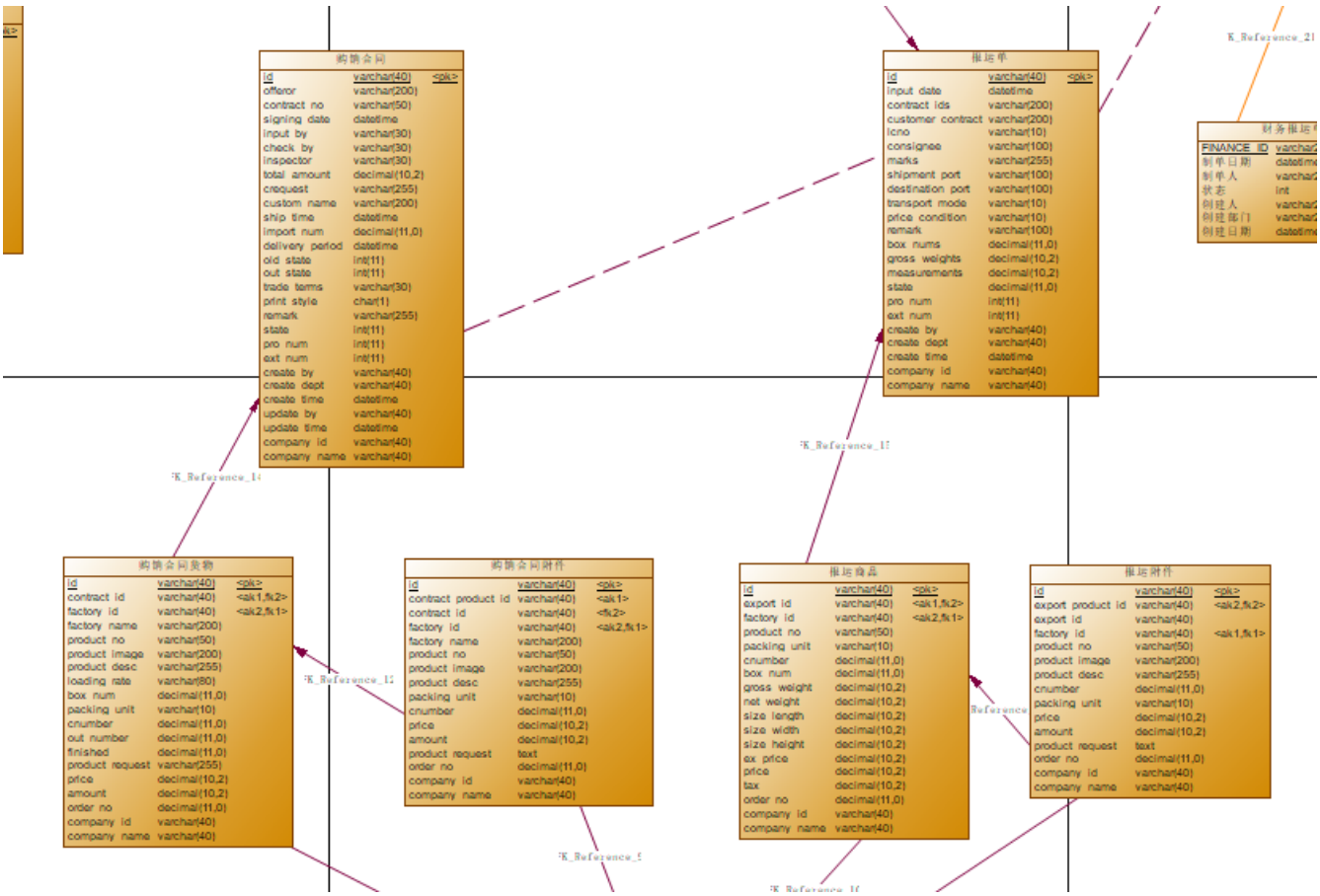
购销合同货物附件的业务逻辑与合同货物的业务逻辑是一样的.

10.出口报运

SaaS货代云平台通过国际物流把货物运往国外,这个过程需要海关人员的审批,所以公司对于出口的产品就要进行出口报运,将来公司的报运人员,就要拿着出口报运到到海关进行审批,当然我们在这里是使用的webservice的方式调用的海关的接口进行报运

每个货代平台对合同,货物和附件的定义要求不一样,但是在海关进行报关的时候,同类商品的数据格式要求是一样的,例如我们定义货物的时候,关注的是厂家,货号,单价等.在报关的时候,货物的单位,长宽高等数据就是必须的,所以此时我们需要两套表分别存储,进而也就有了数据搬家.但是这里我们面临者一个问题:当我们有合同号的时候,想要获取合同下的附件,需要先获取合同,然后再根据合同获取货物,然后再遍历每一个货物,取出货物的附件,于是可以使用**打断设计思想**.

打断设计思想:在传统的一对多关系中,都会在多方加入一个一方的主键作为它的外键,但是在打断设计思想的指导下,不会这样实现,它会在一方的表中加入一个冗余字段,用于保存多方的主键,并用指定的分隔符进行分隔.



在SaaS货代云平台的项目中:

对于已上报的合同,专门有模块对其进行管理,我们可以勾选复选框将数据提交给controller,在controller层我们可以使用list数组接收合同的id数组,也可以使用Stirng进行接收,如果使用String类型进行接收的话,SpringMVC会自动将合同id通过逗号拼接成合同id的字符串.然后设置报运单的基本信息:创建人,创建时间等,再进入service层进行处理.

```
@Override
public void saveExport(Export export) {
    String contractIds = export.getContractIds();

    String[] contractIdArray = contractIds.split(",");
    //需要将原合同的状态改掉
    ContractExample contractExample = new ContractExample();
    ContractExample.Criteria criteria2 = contractExample.createCriteria();
    criteria2.andIdIn(Arrays.asList(contractIdArray));
```

```
List<Contract> contracts = contractDao.selectByExample(contractExample);

//处理export
Integer proNum = 0;
Integer extNum = 0;
for (Contract contract : contracts) {
    //改变contract的状态
    contract.setState(2);
    proNum += contract.getProNum();
    extNum += contract.getExtNum();
    //对contract进行更新
    contractDao.updateByPrimaryKeySelective(contract);
}
//将export存放在数据库
export.setId(UUID.randomUUID().toString().replace("-", ""));
export.setCreateTime(new Date());
export.setProNum(proNum);
export.setExtNum(extNum);
export.setState(0L);
exportDao.insertSelective(export);

//可以根据contractId来查询出货物和附件
ContractProductExample example = new ContractProductExample();
ContractProductExample.Criteria criteria = example.createCriteria();
criteria.andContractIdIn(Arrays.asList(contractIdArray));
List<ContractProduct> contractProductList = contractProductDao.selectByExample(example);

for (ContractProduct contractProduct : contractProductList) {
    //在循环contractProduct的过程中创建exportProduct对象,并将这些对象存放在数据库中
    ExportProduct exportProduct = new ExportProduct();
    BeanUtils.copyProperties(contractProduct,exportProduct);
    //给contractProduct指定ID和export的id(外键)
    exportProduct.setId(UUID.randomUUID().toString().replace("-",""));
    exportProduct.setExportId(export.getId());
    //将exportProduct存入数据库
    exportProductDao.insertSelective(exportProduct);

    //根据contractProductId去数据库查询他的附属品
    ExtCproductExample extCproductExample = new ExtCproductExample();
    ExtCproductExample.Criteria criteria1 = extCproductExample.createCriteria();
    criteria1.andContractProductIdEqualTo(contractProduct.getId());
    List<ExtCproduct> extCproducts = extCproductDao.selectByExample(extCproductExample);

    for (ExtCproduct extCproduct : extCproducts) {
        ExtEproduct extEproduct = new ExtEproduct();
        BeanUtils.copyProperties(extCproduct,extEproduct);
        extEproduct.setId(UUID.randomUUID().toString().replace("-",""));
        extEproduct.setExportProductId(exportProduct.getId());
        extEproduct.setExportId(export.getId());
        extEproductDao.insertSelective(extEproduct);
    }

}
```

```
}
```

1. 首先根据合同id查找出所有的合同,并对合同的状态进行修改,修改为已报运.
2. 合同中包含有货物数量和附件数量,将包运单下的所有合同的数据进行整理,并将这些数据赋值给报运单.
3. 通过循环遍历的方式,再加上BeanUtils提供的copyProperties()将购销合同下的货物和附件部分属性赋值给报运单下的货物和附件.货物的部分属性再使用报运单的时候,需要手动指定,如:长宽高等.

11.文件上传

对于文件上传,首先form表单上要有enctype的属性为multipart/form-data,在input输入框中,声明属性为file,在spring-mvc的配置文件中需要配置一个文件上传解析器

```
        <!--id是确定-->
    <bean id="multipartResolver"
  class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
        <property name="maxUploadSize">
            <value>5242880</value>
        </property>
    </bean>
```

在spring-mvc配置文件中加载七牛的配置文件

```
<!--在配置文件中加载qiniu.properties文件-->
<context:property-placeholder location="classpath:properties/qiniu.properties"/>
```

七牛的配置文件中包含七牛的平台的相关信息

```
qiniu.accessKey=7VhfLuo1FAh9ctdJdunUi_Dvzw4TrYubQvxnxPot
qiniu.secretKey=gufzSn9RYooNEc-SL6PVNmIi5sCrxcgjer2GNEEw
qiniu.bucket=fechinchu
qiniu.rtValue=pu48vv7hc.bkt.clouddn.com
```

编写上传的工具类

```
package com.fechin.web.utils;

import com.google.gson.Gson;
import com.qiniu.common.QiniuException;
import com.qiniu.common.Zone;
import com.qiniu.http.Response;
import com.qiniu.storage.Configuration;
import com.qiniu.storage.UploadManager;
import com.qiniu.storage.model.DefaultPutRet;
import com.qiniu.util.Auth;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import org.springframework.web.multipart.MultipartFile;
```

```
import java.util.UUID;

@Component
public class FileUpLoadUtils {
    @Value("${qiniu.accessKey}")
    private String accessKey;
    @Value("${qiniu.secretKey}")
    private String secretKey;
    @Value("${qiniu.bucket}")
    private String bucket;
    @Value("${qiniu.rtValue}")
    private String rtValue;
    /**
     * 将图片上传到七牛云服务
     * 1.更新用户图片信息（用户id=key）
     * 2.访问图片
     * 存储空间分配的临时域名（免费用户有效期一个月）：http://pkbivgfrm.bkt.clouddn.com+上传的文件名
     * 3.对于更新之后访问图片，防止缓存
     * 更新图片之后：访问的时候，再请求连接添加上时间戳
     */
    public String upload(MultipartFile multipartFile) throws Exception {
        String img = "";
        try {
            //取出原始文件名
            String fileName = multipartFile.getOriginalFilename();
            //随机化文件名
            String uuid = UUID.randomUUID().toString().replace("-", "").toUpperCase();
            fileName = uuid + "_" + fileName;
            //构造一个带指定Zone对象的配置类
            //指定上传文件服务器地址：
            Configuration cfg = new Configuration(Zone.zone0());
            //...其他参数参考类注释
            //上传管理器
            UploadManager uploadManager = new UploadManager(cfg);
            //身份认证
            Auth auth = Auth.create(accessKey, secretKey);
            //指定覆盖上传
            String upToken = auth.uploadToken(bucket, fileName);
            //上传
            Response response = uploadManager.put(multipartFile.getBytes(), fileName,
upToken);
            //解析上传成功的结果
            DefaultPutRet putRet = new Gson().fromJson(response.bodyString(),
DefaultPutRet.class);
            img = rtValue + "/" + fileName;
        } catch (QiniuException ex) {
            System.err.println(ex.getMessage());
            Response r = ex.response;
            System.err.println(r.toString());
            try {
                System.err.println(r.bodyString());
            } catch (QiniuException ex2) {
            }
```

```
        }
        return img;
    }
}
```

编写controller层代码

```
        @RequestMapping(path = "/edit", name = "添加货物或更新货物")
    public String edit(ContractProduct contractProduct, MultipartFile productPhoto) throws
Exception {
        contractProduct.setCompanyName(companyName);
        contractProduct.setCompanyId(companyId);
        if (contractProduct.getId() == null) {
            if (productPhoto!=null){
                String img = "http://"+fileUpLoadUtils.upload(productPhoto);
                //这边返回的是域名/图片名称
                contractProduct.setProductImage(img);
            }
            contractProductService.save(contractProduct);
            return "redirect:/cargo/contractProduct/list.do?contractId=" +
contractProduct.getContractId();
        } else {
            contractProductService.update(contractProduct);
            return "redirect:/cargo/contractProduct/list.do?contractId=" +
contractProduct.getContractId();
        }
    }
```

12.POI报表

EXCEL的两种形式:目前世面上的EXCEL分为两个大的版本Excel2003和Excel2007

	EXCEL2003	**EXCEL2007**
后缀	xls	xlsx
结构	二进制格式,核心结构式符合文档类型结构	XML类型结构
单sheet数据量	行:65535;列256	行:1048575;列:16384
特点	存储容量有限	基于xml压缩,占用空间小,操作效率高

导入pom坐标

```
    <dependency>
        <groupId>org.apache.poi</groupId>
        <artifactId>poi-ooxml</artifactId>
        <version>4.0.1</version>
    </dependency>
```

编写打印报表的java代码:

通过servletContext来获取到模板的真实路径,创建一个模板excel的workbook,获取它的第一个sheet,获取它的标题行,并获取它标题所在的单元格,对该单元个进行数据的更改.之后通过读取模板的样式来将模板的样式存放在CellStyle数组中,之后就是创建行,创建单元格,将数据和样式进行遍历赋值就可以了.

```
@RequestMapping(path = "/printExcel", name = "打印报表")
public void printExcel(String inputDate) throws Exception {
    List<ContractAndProductVo> lists = contractService.findByCompanyIdAndTime(companyId,
inputDate);

    //获取模板的路径
    String realPath =
session.getServletContext().getRealPath("make/xlsprint/tOUTPRODUCT.xlsx");
    Workbook workbook = new XSSFWorkbook(realPath);
    Sheet sheet = workbook.getSheetAt(0);
    Row row = sheet.getRow(0);
    Cell cell = row.getCell(1);
    //获取到标题的单元格,对该单元格进行赋值
    String s = inputDate.replaceAll("-0", "-").replaceAll("-", "年") + "月份出货表";
    cell.setCellValue(s);
    //获取第三行的样式,并将该样式放在数组中
    Row row1 = sheet.getRow(2);
    CellStyle[] cellStyles = new CellStyle[8];
    for (int i = 1; i <=8; i++) {
        cellStyles[i-1] = row1.getCell(i).getCellStyle();
    }
    //获取完样式之后,开始报表赋值,首先定义一个rowIndex,初始值从2开始
    int rowIndex = 2;
    for (ContractAndProductVo list : lists) {
        Row row2 = sheet.createRow(rowIndex++);

        Cell cell1 = row2.createCell(1);
        cell1.setCellValue(list.getCustomName());
        cell1.setCellStyle(cellStyles[0]);

        cell1 = row2.createCell(2);
        cell1.setCellValue(list.getContractNo());
        cell1.setCellStyle(cellStyles[1]);

        cell1 = row2.createCell(3);
        cell1.setCellValue(list.getProductNo());
        cell1.setCellStyle(cellStyles[2]);

        cell1 = row2.createCell(4);
        cell1.setCellValue(list.getcNumber());
        cell1.setCellStyle(cellStyles[3]);

        cell1 = row2.createCell(5);
        cell1.setCellValue(list.getFactoryName());
        cell1.setCellStyle(cellStyles[4]);

        cell1 = row2.createCell(6);
        cell1.setCellValue(list.getDeliveryPeriod());
        cell1.setCellStyle(cellStyles[5]);
```

```
        cell1 = row2.createCell(7);
        cell1.setCellValue(list.getShipTime());
        cell1.setCellStyle(cellStyles[6]);

        cell1 = row2.createCell(8);
        cell1.setCellValue(list.getTradeTerms());
        cell1.setCellStyle(cellStyles[7]);

    }
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
    workbook.write(byteArrayOutputStream);
    DownloadUtil downloadUtil = new DownloadUtil();
    DownloadUtil.download(byteArrayOutputStream,response,"报表.xlsx");
}
```

13.WebService

Java中共有三种WebService规范,分别是 **JAX-WS**, **JAX-RS**, JAXM&SAAJ(废弃).

JAX-WS:主要以xml.

JAX-RS:支持xml和json.

13.1.Apache的CXF

Apache CXF = Celtix+Xfire,刚开始称为:Apache CeltiXfire,后来更名为Apache CXF,简称CXF,是一个开源的Web Service框架,CXF帮助你构建和开发web servcie,比如:SOAP1.1, 1.2,XML/HTTP,RESTful或者CORBA.

SOAP:面向服务应用协议.

13.2.RESTful编程风格

RESTful:一种风格而不是一种协议,它的理念是网络上的所有事物都被抽象为资源,每个资源对应为的资源标识符.

请求方式:GET(查询),POST(保存),PUT(修改),DELETE(删除).

```
public class XXXcontroller{
    /**
    http://localhost:8080/user/1/tom/123
    user,1.tom,123这些是究竟是路径还是请求参数,完全取决于@RequestMapping怎么写,如果
@RequestMapping("/user/1"),这种写法,1 就是路径,如果@RequestMapping("/user/{id}"),这种写法 1 就是一
个参数.
    */
    @Get
    @RequestMapping("/user/{id}")
    public String get(String id){
    }
}
```

13.3.基于JAX-RS规范

一.服务提供者代码编写

1.结合Spring的配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cxf="http://cxf.apache.org/core"
    xmlns:jaxws="http://cxf.apache.org/jaxws"
    xmlns:jaxrs="http://cxf.apache.org/jaxrs"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://cxf.apache.org/core
        http://cxf.apache.org/schemas/core.xsd
        http://cxf.apache.org/jaxws
        http://cxf.apache.org/schemas/jaxws.xsd
        http://cxf.apache.org/jaxrs
        http://cxf.apache.org/schemas/jaxrs.xsd
        ">
    <!--
        address:提供给客户端的访问地址
        serviceClass:就是服务提供的实现类
    -->
    <jaxrs:server address="/userService" serviceClass="com.fechin.service.UserServiceImpl">

    </jaxrs:server>
</beans>
```

2.web.xml配置文件

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5">
    <display-name>webservice_05jaxrs_spring</display-name>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:applicationContext.xml</param-value>
    </context-param>

    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
    </listener>

    <!-- CXF的核心控制器 -->
    <servlet>
        <servlet-name>cxfServlet</servlet-name>
```

```
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>cxfServlet</servlet-name>
        <url-pattern>/ws/*</url-pattern>
    </servlet-mapping>
</web-app>
```

3.domain实体类

```
/**
        <User>
                <id>1</id>
                <username>tom</username>
                <city>北京</city>
        </User>

*/
//name是什么.XML格式的根标签就是什么.
@XmlRootElement(name = "User")
public class User {
        private Integer id;
        private String username;
        private String city;
        private List<Car> cars = new ArrayList<Car>();
        public Integer getId() {
                return id;
        }
        public void setId(Integer id) {
                this.id = id;
        }
        public String getUsername() {
                return username;
        }
        public void setUsername(String username) {
                this.username = username;
        }
        public String getCity() {
                return city;
        }
        public void setCity(String city) {
                this.city = city;
        }
        public List<Car> getCars() {
                return cars;
        }
        public void setCars(List<Car> cars) {
                this.cars = cars;
        }
}
```

4.服务接口

```
@Produces("*/*")//支持所有
public interface IUserService {

        @POST//匹配请求方式
        @Path("/user")//匹配路径
        @Consumes({ "application/xml", "application/json" })//接收请求数据的格式
        void saveUser(User user);

        @PUT
        @Path("/user")
        @Consumes({ "application/xml", "application/json" })
        void updateUser(User user);

        @GET
        @Path("/user")
        @Produces({ "application/xml", "application/json" })//响应数据的格式
        List<User> findAllUsers();

        @GET
        @Path("/user/{id}")
        @Consumes("application/xml")
        @Produces({ "application/xml", "application/json" })
        User finUserById(@PathParam("id") Integer id);

        @DELETE
        @Path("/user/{id}")
        @Consumes({"application/xml", "application/json"})
        void deleteUser(@PathParam("id") Integer id);
}
```

我们还要提供服务的实现,自己编写服务的实现即可

二.服务提供者代码编写

```
public class TestConsumer {
    @Test
    public void test(){
        User user =
WebClient.create("http://localhost:8080/jaxrs/ws/userService/user/1").type(MediaType.APPLICA
TION_JSON_TYPE).get(User.class);
        System.out.println(user);
    }

    @Test
    public void test02(){
        User user = new User();
        user.setId(3);
        user.setUsername("jack");
        user.setCity("北京");
```

```
WebClient.create("http://localhost:8080/jaxrs/ws/userService/user").type(MediaType.APPLICATI
ON_JSON_TYPE).post(user);
    }
}
```

13.4.基于JAX-WS规范

SOAP:Soap = http+xml

WSDL:文档说明书,程序来解析WSDL

一.服务提供者代码编写

1.服务接口

```
@WebService //当写了此注解时，表明当前接口是一个webservice接口
public interface WeatherService {
        /**
         * 根据城市名称，获取城市天气
         * @param cityName
         * @return
         */
        String getWeatherByCityName(String cityName);
}
```

2.服务实现自己实现就可以了.

3.使用测试类来模拟服务.

```
public class JaxwsServerTest {
    public static void main(String[] args) {
        //1.创建工厂
        JaxWsServerFactoryBean factoryBean = new JaxWsServerFactoryBean();
        //2.指定服务接口对象
        factoryBean.setServiceBean(new WeatherServiceImpl());
        //3.指定发布地址
        factoryBean.setAddress("http://localhost:1234");
        //4.配置日志输出,第一个是请求过来控制台响应的信息,第二个响应信息.
        factoryBean.getInInterceptors().add(new LoggingInInterceptor());
        factoryBean.getOutInterceptors().add(new LoggingInInterceptor());
        //5.发布服务
        factoryBean.create();
    }
}
```

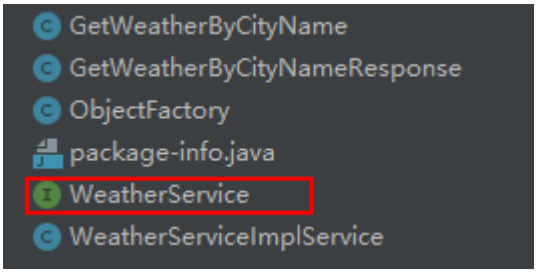
这样子我们就可以使用http://localhost:1234?wsdl,来获取xml格式的"说明书".

```
C:\JavaDevelopmentCodes\IDEAProjects_JavaEE\WebServiceJAX\jaxws_consumer\src\main>wsimport -s . http://localhost:1234/?wsdl
正在解析 WSDL...


正在生成代码...

正在编译代码...


C:\JavaDevelopmentCodes\IDEAProjects_JavaEE\WebServiceJAX\jaxws_consumer\src\main>
```

二.服务消费者代码编写

我们可以使用jdk给我们提供好的wsimport去生成"说明书"的java类和class对象.

如下图:



之后,我们就可以使用编写测试类:

```
public static void main(String[] args) {
        //1.创建工厂代理对象
        JaxWsProxyFactoryBean jaxWsProxyFactoryBean = new JaxWsProxyFactoryBean();
        //2.设置访问地址
        jaxWsProxyFactoryBean.setAddress("http://localhost:1234/?wsdl");
        //3.创建调用的代理对象
        jaxWsProxyFactoryBean.setServiceClass(WeatherService.class);
        WeatherService weatherService = (WeatherService) jaxWsProxyFactoryBean.create();
        //4.调用服务中的方法
        String weather = weatherService.getWeatherByCityName("北京");
        System.out.println(weather);
    }
```

3.与Spring整合

参考项目

14.图形报表Echarts

Echarts官网

15.PDF报表

15.1.开发步骤概述:

JasperReport开发步骤:

1.设计(Design)阶段:所谓的报表设计就是创建一些模板,模板包含了报表的布局和设计,模板设计完成之后,我们将模板保存为JRXML文件(JR表示的式JasperReports),其实就是一个xml文件.

2.执行(Execution)阶段:使用JRXML文件编译成为可执行的二进制文件(.jsper文件)集合数据进行执行,填充报表数据.

3.输出(Export)阶段:数据填充结束,可以指定输出为多种形式的报表

15.2.模板工具(Jaspersoft Studio):

15.3.开发步骤详述:

导入坐标

```
<dependency>
    <groupId>net.sf.jasperreports</groupId>
    <artifactId>jasperreports</artifactId>
    <version>6.5.0</version>
</dependency>
<dependency>
    <groupId>org.olap4j</groupId>
    <artifactId>olap4j</artifactId>
    <version>1.2.0</version>
</dependency>
<dependency>
    <groupId>com.lowagie</groupId>
    <artifactId>itext</artifactId>
    <version>2.1.7</version>
</dependency>
```

引入模板:将编译好的(.jasper)模板引入到当前的工程中

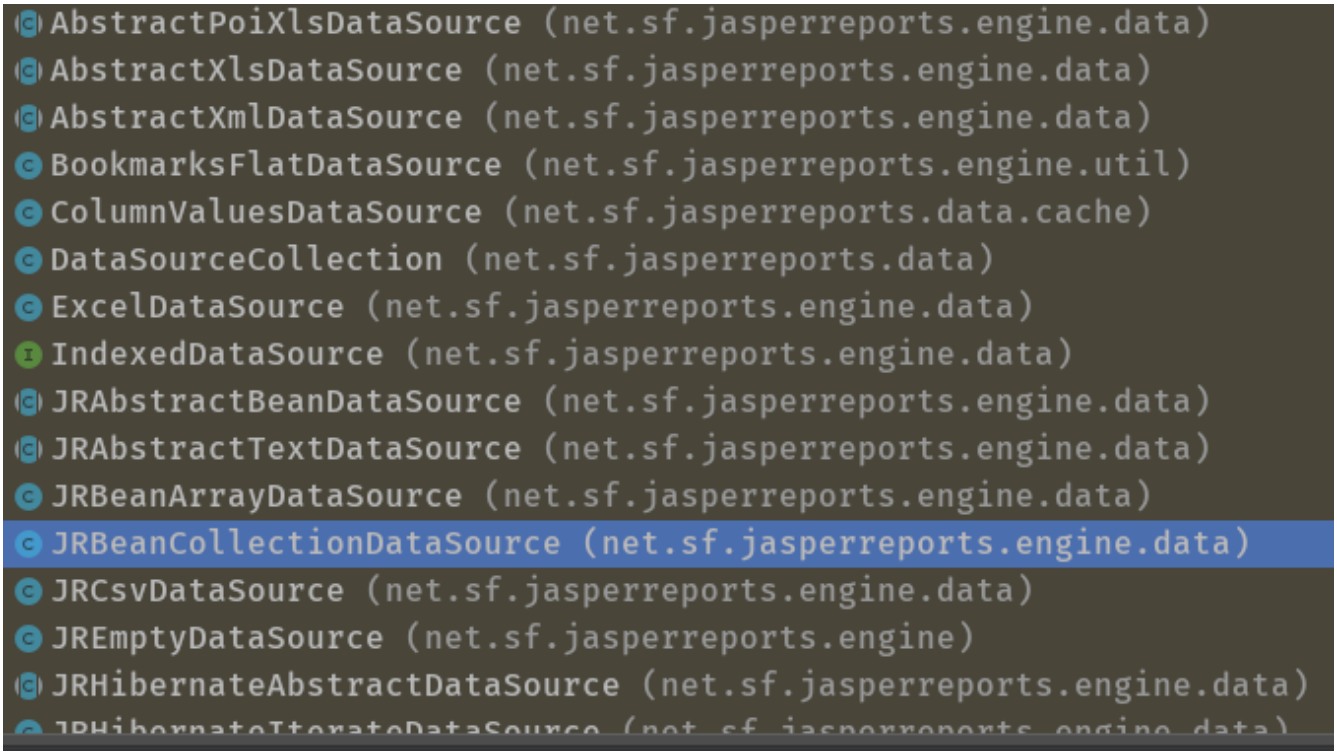
编写controller层代码:

1.首先获得jasper文件也是就是模板的据对路径

2.创建该文件的输入流

3.创建JasperPrint对象,该对象是通过JasperFillManager使用fillReport来获取的,该方法需要三个参数,第一个参数是输入流,读取该文件的模板,第二个是需要一个Map集合,该Map集合主要对应是模板中的Parameters参数,我们可以使用Spring提供的BeanMapUtils将从数据库中查出来的bean转化成Map.第二个参数有了,该方法的第三个参数需要的是JRDataSource,JRDataSource的实现类如下:

如果模板中没有Field,那么我们直接使用JREmptyDataSource,如果是一个Bean的集合,那么使用JRBeanCollectionDataSource.



4.通过JasperExportManager将export输出成PDF.

```
@RequestMapping(path = "/exportPdf", name = "pdf打印")
public void exportPdf(String id) throws Exception {
    Export export= exportService.findById(id);
    List<ExportProduct> exportProducts = exportProductService.findByExportId(id);
    Map<String, Object> stringObjectMap = BeanMapUtils.beanToMap(export);
    JRBeanCollectionDataSource js = new JRBeanCollectionDataSource(exportProducts);
    //1.获得jasper文件的绝对路径
    String realPath = session.getServletContext().getRealPath("/jasper/export.jasper");
    //2.创建文件输入流
    FileInputStream fileInputStream = new FileInputStream(realPath);
    //3.创建jrprint对象
    JasperPrint jasperPrint = JasperFillManager.fillReport(fileInputStream, stringObjectMap,
js);
    //4.打印(下载)
    JasperExportManager.exportReportToPdfStream(jasperPrint,response.getOutputStream());
}
```

15.4.中文乱码处理

设计阶段需要制定中文样式

编写加载中文乱码处理的properties文件:让JasperReport进行扫描,所以名称必须固定:**jasperreports_extension.properties**;

```
net.sf.jasperreports.extension.registry.factory.simple.font.families=net.sf.jasperreports.en
gine.fonts.SimpleFontExtensionsRegistryFactory
net.sf.jasperreports.extension.simple.font.families.lobstertwo=simkai/fonts.xml
```

该配置文件用来读取resources下的simkai/fonts.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>

<fontFamilies>
    <fontFamily name="楷体">
        <normal>simkai/simkai.ttf</normal>
        <bold>simkai/simkai.ttf</bold>
        <italic>simkai/simkai.ttf</italic>
        <boldItalic>simkai/simkai.ttf</boldItalic>
        <pdfEncoding>Identity-H</pdfEncoding>
        <pdfEmbedded>true</pdfEmbedded>
        <exportFonts>
            <export key="net.sf.jasperreports.html">'楷体', Arial, Helvetica, sans-
serif</export>
            <export key="net.sf.jasperreports.xhtml">'楷体', Arial, Helvetica, sans-
serif</export>
        </exportFonts>
        <!--
        <locales>
            <locale>en_US</locale>
            <locale>de_DE</locale>
        </locales>
        -->
    </fontFamily>
</fontFamilies>
```

该xml配置文件会读取simkai/simkai.ttf文件,我们可以从下载好字体的ttf文件,并将该字体文件放在xml中指定的文件夹中.

16.消息中间件(见md文件)