

**BOS物流项目**

# 教学计划

1、jQuery easyUI中动态添加选项卡

2、jquery ztree插件使用

* + 下载ztree
  + 基于标准json数据构造ztree
  + 基于简单json数据构造ztree（重点）

3、数据库建模工具PowerDesigner使用方式

4、myeclipse翻转引擎插件使用（了解）

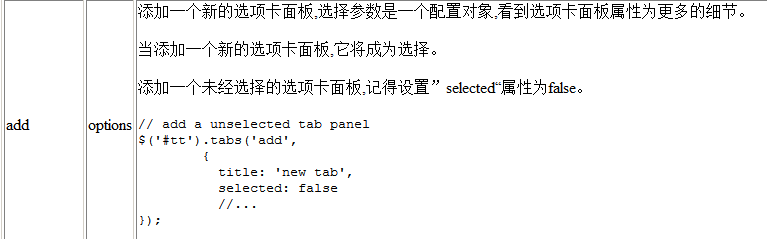
5、项目底层代码构建（重点）

* + 持久层代码抽取
  + 表现层代码抽取

6、实现BOS项目登录和注销功能

# jQuery easyUI中动态添加选项卡

* 用于动态添加一个选项卡



* 选中指定的选项卡和判断某个选项卡是否存在



<a id=*"but1"* class=*"easyui-linkbutton"*>添加一个选项卡</a>

<script type=*"text/javascript"*>

$(**function**(){

//页面加载完成后，为上面的按钮绑定事件

$("#but1").click(**function**(){

//判断“系统管理”选项卡是否存在

**var** e = $("#mytabs").tabs("exists","系统管理");

**if**(e){

//已经存在，选中就可以

$("#mytabs").tabs("select","系统管理");

}**else**{

//调用tabs对象的add方法动态添加一个选项卡

$("#mytabs").tabs("add",{

title:'系统管理',

iconCls:'icon-edit',

closable:**true**,

content:'<iframe frameborder="0" height="100%" width="100%" src="https://www.baidu.com"></iframe>'

});

}

});

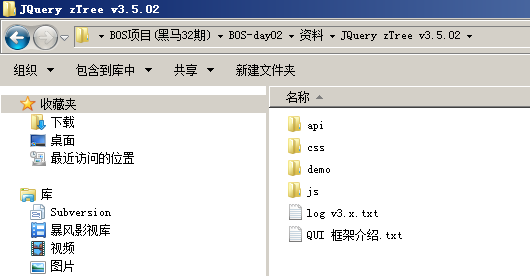
});

</script>

# jquery ztree插件使用

官网：ztree.me

目录结构：



* 在页面中引入ztree相关的文件：

<link rel="stylesheet" href="../../../css/zTreeStyle/zTreeStyle.css" type="text/css">

<script type="text/javascript" src="../../../js/jquery-1.4.4.min.js"></script>

<script type="text/javascript" src="../../../js/jquery.ztree.core-3.5.js"></script>

## 使用标准json数据构造ztree（了解）

<!-- 展示ztree效果 :使用标准json数据构造ztree-->

<ul id=*"ztree1"* class=*"ztree"*></ul>

<script type=*"text/javascript"*>

$(**function**(){

//页面加载完成后，执行这段代码----动态创建ztree

**var** setting = {};

//构造节点数据

**var** zNodes = [

{"name":"节点一","children":[

{"name":"节点一\_1"},

{"name":"节点一\_2"}

]},//每个json对象表示一个节点数据

{"name":"节点二"},

{"name":"节点三"}

];

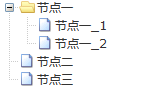
//调用API初始化ztree

$.fn.zTree.init($("#ztree1"), setting, zNodes);

});

</script>

效果：



## 使用简单json数据构造ztree（重点）

<!-- 展示ztree效果 :使用简单json数据构造ztree-->

<ul id=*"ztree2"* class=*"ztree"*></ul>

<script type=*"text/javascript"*>

$(**function**(){

//页面加载完成后，执行这段代码----动态创建ztree

**var** setting2 = {

data: {

simpleData: {

enable: **true**//使用简单json数据构造ztree节点

}

}

};

//构造节点数据

**var** zNodes2 = [

{"id":"1","pId":"2","name":"节点一"},//每个json对象表示一个节点数据

{"id":"2","pId":"3","name":"节点二"},

{"id":"3","pId":"0","name":"节点三"}

];

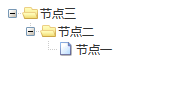
//调用API初始化ztree

$.fn.zTree.init($("#ztree2"), setting2, zNodes2);

});

</script>

效果：



## 发送ajax请求获取json数据构造ztree

<!-- 展示ztree效果 :发送ajax请求获取简单json数据构造ztree-->

<ul id=*"ztree3"* class=*"ztree"*></ul>

<script type=*"text/javascript"*>

$(**function**(){

//页面加载完成后，执行这段代码----动态创建ztree

**var** setting3 = {

data: {

simpleData: {

enable: **true**//使用简单json数据构造ztree节点

}

}

};

//发送ajax请求，获取json数据

//jQuery提供 的ajax方法：ajax、post、get、load、getJSON、getScript

**var** url = "${pageContext.request.contextPath}/json/menu.json";

$.post(url,{},**function**(data){

//调用API初始化ztree

$.fn.zTree.init($("#ztree3"), setting3, data);

},'json');

});

</script>

## 使用ztree提供的API为节点绑定事件

<!-- 展示ztree效果 :发送ajax请求获取简单json数据构造ztree-->

<ul id=*"ztree3"* class=*"ztree"*></ul>

<script type=*"text/javascript"*>

$(**function**(){

//页面加载完成后，执行这段代码----动态创建ztree

**var** setting3 = {

data: {

simpleData: {

enable: **true**//使用简单json数据构造ztree节点

}

},

callback: {

//为ztree节点绑定单击事件

onClick: **function**(event, treeId, treeNode){

**if**(treeNode.page != undefined){

//判断选项卡是否已经存在

**var** e = $("#mytabs").tabs("exists",treeNode.name);

**if**(e){

//已经存在，选中

$("#mytabs").tabs("select",treeNode.name);

}**else**{

//动态添加一个选项卡

$("#mytabs").tabs("add",{

title:treeNode.name,

closable:**true**,

content:'<iframe frameborder="0" height="100%" width="100%" src="'+treeNode.page+'"></iframe>'

});

}

}

}

}

};

//发送ajax请求，获取json数据

//jQuery提供 的ajax方法：ajax、post、get、load、getJSON、getScript

**var** url = "${pageContext.request.contextPath}/json/menu.json";

$.post(url,{},**function**(data){

//调用API初始化ztree

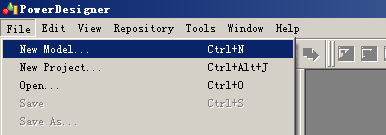
$.fn.zTree.init($("#ztree3"), setting3, data);

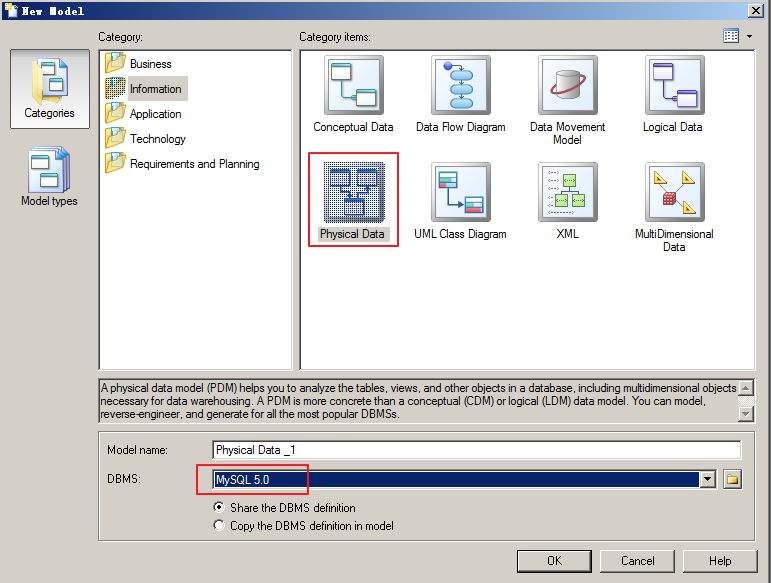
},'json');

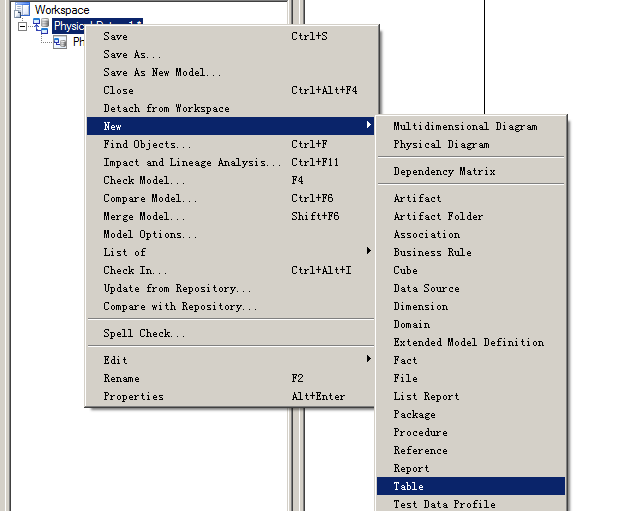
});

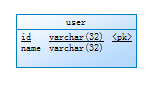
</script>

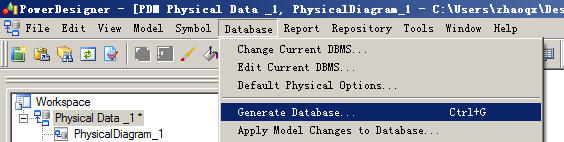
# 数据库建模工具PowerDesigner使用方式

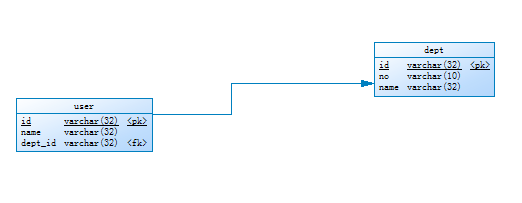








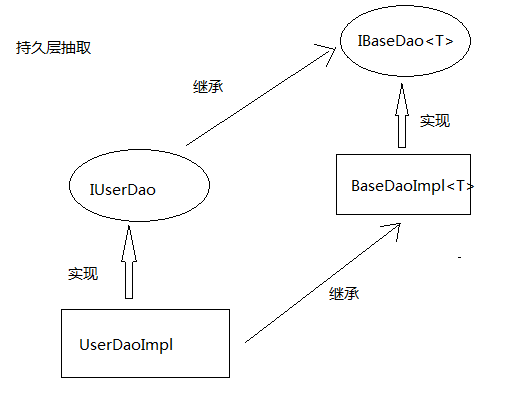




# myeclipse翻转引擎插件使用

# 项目底层代码构建（重点）

## 持久层代码抽取



IBaseDao接口：

/\*\*

\* 持久层通用接口

\* **@author** zhaoqx

\*

\* **@param** <T>

\*/

**public** **interface** IBaseDao<T> {

**public** **void** save(T entity);

**public** **void** delete(T entity);

**public** **void** update(T entity);

**public** T findById(Serializable id);

**public** List<T> findAll();

}

BaseDaoImpl<T>实现类：

/\*\*

\* 持久层通用实现

\* **@author** zhaoqx

\*

\* **@param** <T>

\*/

**public** **class** BaseDaoImpl<T> **extends** HibernateDaoSupport **implements** IBaseDao<T> {

//代表的是某个实体的类型

**private** Class<T> entityClass;

@Resource//根据类型注入spring工厂中的会话工厂对象sessionFactory

**public** **void** setMySessionFactory(SessionFactory sessionFactory){

**super**.setSessionFactory(sessionFactory);

}

//在父类（BaseDaoImpl）的构造方法中动态获得entityClass

**public** BaseDaoImpl() {

ParameterizedType superclass = (ParameterizedType) **this**.getClass().getGenericSuperclass();

//获得父类上声明的泛型数组

Type[] actualTypeArguments = superclass.getActualTypeArguments();

entityClass = (Class<T>) actualTypeArguments[0];

}

**public** **void** save(T entity) {

**this**.getHibernateTemplate().save(entity);

}

**public** **void** delete(T entity) {

**this**.getHibernateTemplate().delete(entity);

}

**public** **void** update(T entity) {

**this**.getHibernateTemplate().update(entity);

}

**public** T findById(Serializable id) {

**return** **this**.getHibernateTemplate().get(entityClass, id);

}

**public** List<T> findAll() {

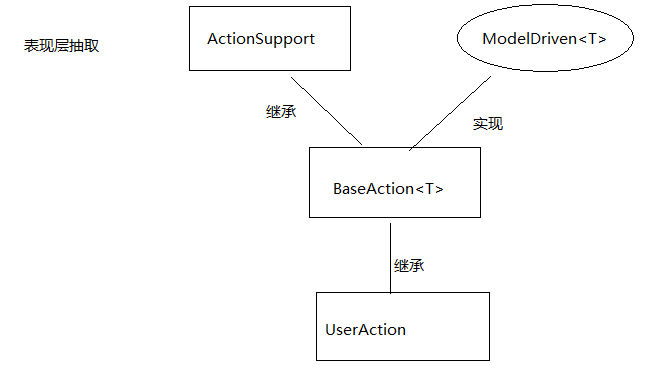
String hql = "FROM " + entityClass.getSimpleName();

**return** (List<T>) **this**.getHibernateTemplate().find(hql);

}

}

## 表现层代码抽取



BaseAction代码：

/\*\*

\* 表现层通用实现

\* **@author** zhaoqx

\*

\* **@param** <T>

\*/

**public** **class** BaseAction<T> **extends** ActionSupport **implements** ModelDriven<T> {

//模型对象

**private** T model;

**public** T getModel() {

**return** model;

}

//在构造方法中动态获取实体类型，通过反射创建model对象

**public** BaseAction() {

ParameterizedType genericSuperclass = (ParameterizedType) **this**.getClass().getGenericSuperclass();

//获得BaseAction上声明的泛型数组

Type[] actualTypeArguments = genericSuperclass.getActualTypeArguments();

Class<T> entityClass = (Class<T>) actualTypeArguments[0];

//通过反射创建对象

**try** {

model = entityClass.newInstance();

} **catch** (InstantiationException e) {

e.printStackTrace();

} **catch** (IllegalAccessException e) {

e.printStackTrace();

}

}

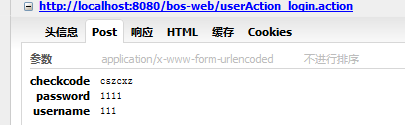
}

# 实现BOS项目登录

登录页面：/login.jsp



第一步：调整login.jsp页面，为登录按钮指定单击事件，提交表单



第二步：创建UserAction，提供login登录方法

@Controller

@Scope("prototype")

**public** **class** UserAction **extends** BaseAction<User> {

//属性驱动，接收页面输入的验证码

**private** String checkcode;

**public** **void** setCheckcode(String checkcode) {

**this**.checkcode = checkcode;

}

@Autowired

**private** IUserService userService;

/\*\*

\* 用户登录

\*/

**public** String login(){

//从Session中获取生成的验证码

String validatecode = (String) ServletActionContext.*getRequest*().getSession().getAttribute("key");

//校验验证码是否输入正确

**if**(StringUtils.*isNotBlank*(checkcode) && checkcode.equals(validatecode)){

//输入的验证码正确

User user = userService.login(model);

**if**(user != **null**){

//登录成功,将user对象放入session，跳转到首页

ServletActionContext.*getRequest*().getSession().setAttribute("loginUser", user);

**return** ***HOME***;

}**else**{

//登录失败，,设置提示信息，跳转到登录页面

//输入的验证码错误,设置提示信息，跳转到登录页面

**this**.addActionError("用户名或者密码输入错误！");

**return** ***LOGIN***;

}

}**else**{

//输入的验证码错误,设置提示信息，跳转到登录页面

**this**.addActionError("输入的验证码错误！");

**return** ***LOGIN***;

}

}

}

第三步：提供UserService接口和实现类

@Service

@Transactional

**public** **class** UserServiceImpl **implements** IUserService{

@Autowired

**private** IUserDao userDao;

/\*\*\*

\* 用户登录

\*/

**public** User login(User user) {

//使用MD5加密密码

String password = MD5Utils.*md5*(user.getPassword());

**return** userDao.findUserByUsernameAndPassword(user.getUsername(),password);

}

}

第四步：提供UserDao接口和实现类

@Repository

**public** **class** UserDaoImpl **extends** BaseDaoImpl<User> **implements** IUserDao {

/\*\*

\* 根据用户名和密码查询用户

\*/

**public** User findUserByUsernameAndPassword(String username, String password) {

String hql = "FROM User u WHERE u.username = ? AND u.password = ?";

List<User> list = (List<User>) **this**.getHibernateTemplate().find(hql, username,password);

**if**(list != **null** && list.size() > 0){

**return** list.get(0);

}

**return** **null**;

}

}

第五步：配置struts.xml

<!-- 用户管理 -->

<action name=*"userAction\_\*"* class=*"userAction"* method=*"{1}"*>

<result name=*"login"*>/login.jsp</result>

<result name=*"home"*>/index.jsp</result>

</action>

# 注销功能

将session销毁，跳转到登录页面

