社会保险核心平台三版培训

技术培训－系统设计与开发

北京利博赛社保信息技术有限公司

目录

1 系统设计

2 系统与页面开发



系 统 设 计



系统设计

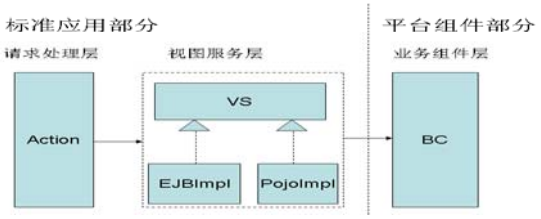
- 设计原则
- 总体设计与层级职责
- 与核心平台二版的差异


系统设计——设计原则

核心平台三版的设计原则为方便本地开发，增加组件的复用度来减少开发。

抛开框架从应用构建的角度，核心平台整体上分为标准应用部分与平台组件部分。标准应用部分包括请求处理层与视图服层，是核三标准应用特定实现部分，当本地化时业务变化时，这部分对应的请求处理和视图服务往往需要相应的改变，复用能力相对较低；核三的复用部分主要在业务组件层，业务组件可被不同的视图服务层调用与组合，以完成不同的业务需求。

业务组件是平台的主要复用单元，内聚于领域模型的一个主业务概念，提供与主业务概念相关的一组业务服务。




系统设计

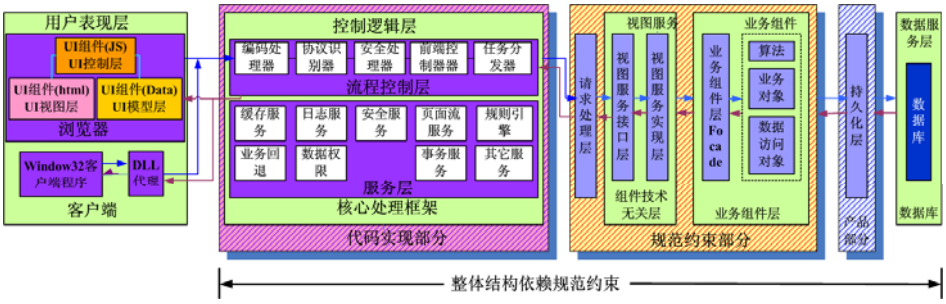
- 设计原则
- 总体设计与层级职责
- 与核心平台二版的差异

系统设计——总体设计与层次职责

总体设计：

总体设计用于规范系统调用的流程和步骤，并对业务层之间的相互关联关系加以约束，使不同业务设计中类的职责和协作关系统一 。

系统层次调用关系图：



核心平台三版的总体架构可以分为以下层次：界面视图层、控制层、请求处理层、视图服务层、业务组件层。

系统设计——总体设计与层次职责

各层次的职责

界面视图层

BSS结构下的视图层主要采用JSP技术实现，核心平台三版提供了一套用于快速JSP UI开发的标签库。

CSS结构下的视图层为传统的Win32客户端程序，可根据采用的开发工具和语言进行适当的视图层开发。

控制层

控制层由核心平台三版的核心框架提供，用于控制程序的控制流转，在业务开发时不需要对本层进行开发。


系统设计——总体设计与层次职责


请求处理层

调用相对应的视图服务接口层的方法来完成任务
一个业务单元对应一个视图服务层接口

- 处理浏览器请求的Action：
  处理一次浏览器请求，并为客户端返回本次请求所要求的数据，指定返回客户端的页面。
- 处理Soap请求的Action：
  处理一次传统客户端请求，并为客户端返回本次请求所要求的数据将请求端请求的Soap数据对象转换为业务参数Javabean对象将业务返回的数据转换为Soap对象返回框架。
- 对应的类：**XXXAction**


系统设计——总体设计与层次职责


视图服务层

服务视图层负责调用业务组件BC层的组件完成一次交互请求所需要的业务逻辑操作。

- 通过VS接口层屏蔽技术实现细节，实现过程中使用POJO，但可以自动平滑的发布为EJB，在EJB模式下，VS组件相当与EJB Facade
- 负责业务流程逻辑的串接，可以理解为是组合服务，可能需要协调多个BC组件完成业务逻辑。
- 事务控制单元，VS的方法是启动事务控制的地方 。
- 视图服务层可细分为：视图服务接口层和视图服务实现层。

系统设计——总体设计与层次职责

- 视图服务接口层

业务处理类（PO）的方法调用接口，这一层不依赖于任何组件技术（如EJB、POJO等），这一层为PO方法调用规定了接口。

实现方式：Java Interface

- 视图服务实现层

业务接口层的组件相关技术实现，这一层实现了具体的组件技术（如EJB、POJO等），这一层不实现具体的业务逻辑，仅调用业务组件层的具体业务组件来完成业务逻辑的处理

对应的类：XXXVSImpl

系统设计——总体设计与层次职责

- 业务组件层

业务组件层可细分为：业务组件Façade层、算法对象层、业务对象层、数据访问层

- 业务组件Façade层

为业务组件提供一个一致的对外接口（接口意义）

- 算法对象层

封装可变的业务算法

- 业务对象层

实现业务逻辑

如果有交易则做交易的相应处理

- 数据访问层

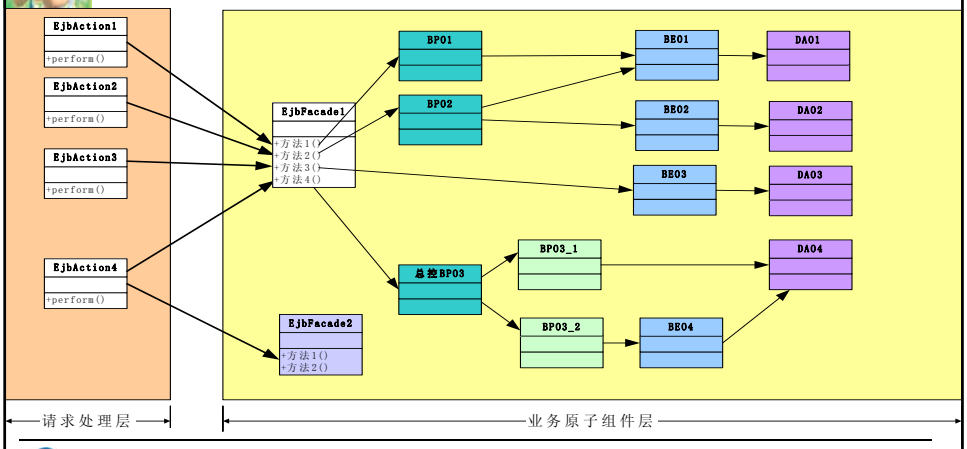
主要完成数据库的访问

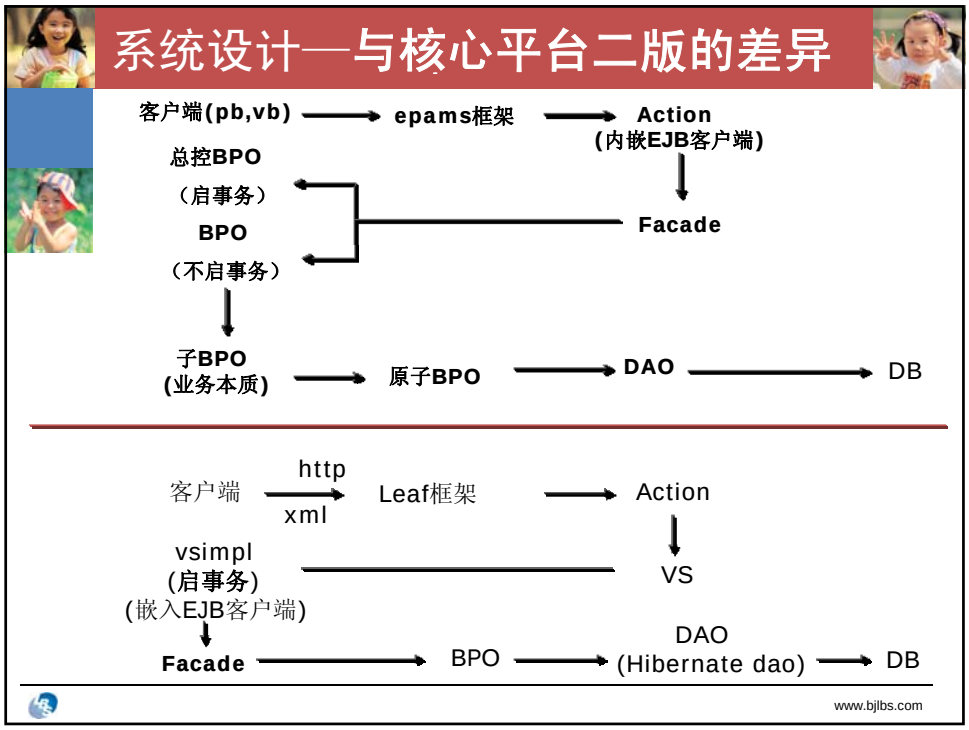

系统设计

- 设计原则
- 总体设计与层级职责
- 与核心平台二版的差异


系统设计—与核心平台二版的差异

核心平台二版系统层次调用关系图




系统设计—与核心平台二版的差异
客户端(pb,vb)
epams框架
Action
(内嵌EJB客户端)
总控BPO
（启事务）
BPO
（不启事务）
Facade
子BPO
(业务本质)
原子BPO
DAO
DB
客户端
http
xml
Leaf框架
Action
vsimpl
(启事务)
(嵌入EJB客户端)
VS
Facade
BPO
DAO
(Hibernate dao)
DB
www.bjlbs.com


系统与页面开发
www.bjlbs.com

系统与页面开发

- 系 统 设 计
- 页 面 设 计

系统开发

- 界面视图层
- 请求处理层
- 视图服务层
- 业务组件层

系统开发——界面视图层

开发JSP可以使用核心平台三版提供的标签库，标签库的声明方法如下：

```
<%@ taglib uri="/WEB-INF/sicp3.tld" prefix="sicp3"%>
```

JSP标记文件声明：

```
<%@ taglib prefix="tags" tagdir="/WEB-INF/tags"%>
```

在使用标签库之前，需要确认WEB-INF目录下存在相应的tld文件，并且在web.xml文件中要正确地配置tld文件，示例配置如下：

```
<taglib>
        <taglib-uri>/WEB-INF/sicp3.tld</taglib-uri>
          <taglib-location>/WEB-INF/sicp3.tld</taglib-location>
</taglib>
```

系统开发

- 界面视图层
- 请求处理层
- 视图服务层
- 业务组件层


系统开发——请求处理层

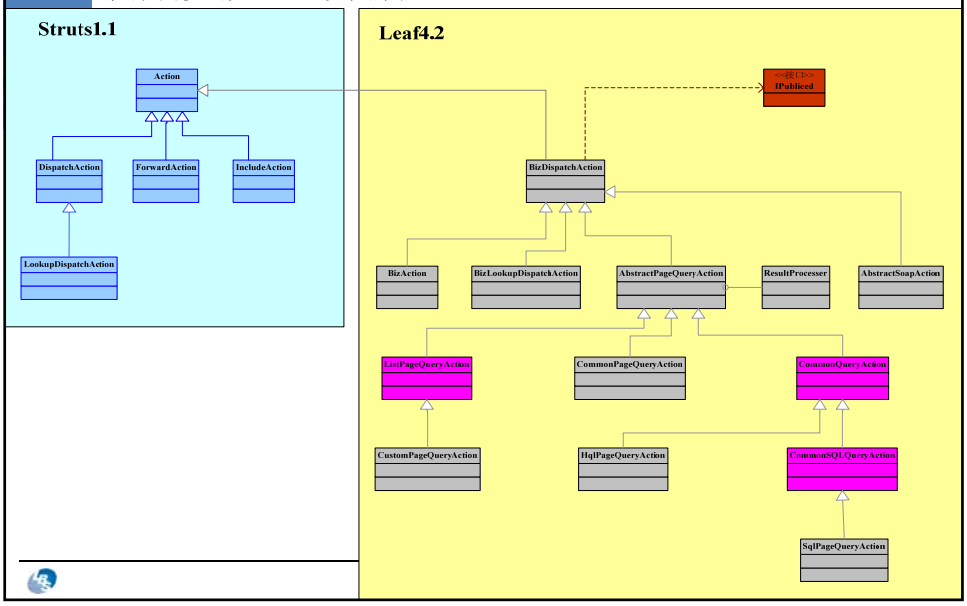
Action的主要职责是将Form对象转换成DTO对象，并调用视图服务层（VS），将复杂的业务逻辑交给VS的方法来处理，处理完后再调用父类提供的方法进行必要的异常和日志处理。

Action只是起到传递的作用，具体的业务逻辑应该交由视图服务层处理，所以不要在Action中包含业务处理逻辑。

Action需要继承框架提供的父类。

- 如果Action中包括分页查询，那么需要继承CommonQueryAction /CommonSQLQueryAction/ListPageQueryActon中任一个Action类；
- 如没有分页查询，则需继承BizDispatchAction；
- 如为传统客户端请求，则需继承AbstractSoapAction。


框架提供的父Action类关系图




系统开发——请求处理层



➢ 异常及日志的处理

Action中都要捕获异常，一般应截取Exception异常和BusinessException
异常，防止系统异常直接抛到前台。

saveSuccessfulMsg()：
保存成功消息，以便页面errors标签获取成功消息在页面显示。
saveErrors()：
保存错误信息，以便框架进行异常和日志处理。


系统开发——请求处理层



■ 代码样例

```
public class TempleteAction extends CommonQueryAction {
  public ActionForward query(ActionMapping mapping,
              ActionForm actionForm, HttpServletRequest req,
                  HttpServletResponse res) throws Exception {
     ActionForward af = mapping.findForward("templete");
    TempleteDTO tdto = new TempleteDTO();
    try {
          TempleteForm tf = (TempleteForm) actionForm;
           BeanHelper.copyProperties(tf, tdto);
           TempleteVS tvs = (TempleteVS) getService(" TempleteVS");
           HQLInfo hi = tvs.query(tdto);
           init(req, af, hi);
           saveSuccessfulMsg(req, "查询成功！");
     } catch (BusinessException e) {
           saveErrors(req, e);
           af = mapping.findForward("queryerror");
      }catch (Exception e) {
           saveErrors(req, e);
           af = new  ActionForward("/samples/commonError.do");
      } return af;
   }
}
```

系统开发——请求处理层

- 页面流转

struts-samples.xml:

```
<form-beans>
        <form-bean name="templeteForm"
                type="com.lbs.leaf.samples.templete.rp.TempleteForm" />
</form-beans>
<action-mappings>
<action path ="/samples/commonError"
                forward="/samples/common/commonerror.jsp"></action>
<action najip" templeteForm " parameter="method"
            path="/samples/templeteAction" scope="request"
                type="com.lbs.leaf.samples.templete.rp. TempleteAction">
   <forward name="templete" path="/samples/templete/templete.jsp" />
   <forward name=" queryerror" path="/samples/templete/qerror.jsp" />
</action>
</actiojipappings>
```

web.xml:

```
<init-param>
    <param-name>config</param-name>
            <param-value>/WEB-INF/struts-samples.xml</param-value>
</init-param>
```


系统开发

- 界面视图层
- 请求处理层
- 视图服务层
- 业务组件层


系统开发——视图服务层

➢ 视图服务接口层

```
public interface TempleteVS {
        HQLInfo query(TempleteDTO dto);
}
```

➢ 视图服务实现层

```
public class TempleteVSImpl implements TempleteVS {
    public HQLInfo query(Templete tdto) {
                        ……
    }
}
```

VS配置：

/WEB-INF/conf/ServiceConfig.xml文件中，加入TempleteVS的配置

```
<service id=" TempleteVS "
      pojoname="com.lbs.leaf.samples.templete.vs.impl.TempleteVSImpl"
                                  ejbname="" type="0" desc="样例代码" / >
```

action中的使用：

```
TempleteVS tvs = (TempleteVS) getService(" TempleteVS");
```


系统开发——视图服务层

➢ 视图服务实现层

- **VSImpl开发**
  - 控制事务
  - 控制通用回退
  - 调用业务组件中**Façade**类中的业务方法，将业务组装成能够完成一次用户请求的业务处理


系统开发——视图服务层

- 在VSImpl中控制事务的方法
  - **AOP**的方式

这种方式只能在VSImpl类中使用才有效，不能在其它类中使用。只需要将需要事务控制的代码单独放到一个方法名以TA结尾的Protected方法中即可。

  - **API** 的方式

这种事务控制方式可以在任何地方使用。如果有特殊业务不能在VSImpl中控制事务，必须将事务控制放到业务组件层的时候，就可以使用这种方式。


系统开发——视图服务层

- AOP的方式

```
public class TempleteBSImpl implements TempleteBS {
        public void deleteObj(Templete tdto) {
                deleteObjTA(tdto);
         }
         protected void deleteObjTA(Templete tdto) {
                                .....
     }
}
```


系统开发——视图服务层

- API 的方式

```
public class TempleteBSImpl implements TempleteBS {
        public void delete(Templete tdto) {
                useAPIDelete(tdto);
        }
       protected void useAPIDelete(final Templete tdto) {
// 使用事务处理工具，在事务上下文中执行ICommand对象的execute方法
                TransUtil.trans(new ICommand() {
                 public Object execute() {
                        TempleteFacade tbc = new TempleteFacade ();
                        tbc.deleteObj(tdto);
                        return null;
                  }
                 });
            }
}
```


系统开发——视图服务层

- 在VSImpl中控制事务与回退的方法
- AOP的方式

使用AOP的形式可同时支持事务和通用回退，使用方法名+ROLTA作为新的方法名。

- API 的方式

在数据库数据发生修改时，存在两种情况：单条修改和批量修改。
单条修改使用Bizlogger.rolLog(ICommand biz, String bizid)方法来保存记录业务日志与数据摘要信息。重写execute()方法，在execute()方法中保存业务日志，保存数据摘要信息，保存修改的业务；
当批量修改时，对于提交的Collection类型数据进行循环，单条记录业务日志与数据摘要。

系统开发——视图服务层

- **AOP的方式**

```
public void update2(final Rollogdemo rld) {
    // 使用APO启事物同时支持通用回退
    updateROLTA(rld);
}

protected void updateROLTA(final Rollogdemo rld) {
    TempleteFacade tfc = new TempleteFacade();
    final Date startTime = Bizlogger.currentTime();// 获得业务开始时间
    final BizDigestInfoParameters infops;
    infops = new BizDigestInfoParameters();
    final BizDigestDataParameters dataps;
    dataps = new BizDigestDataParameters();
    final long opseno = DbRollUtil.getOpseno();
    // 根据业务流水号，字段名称，字段描述，是否可见添加到数据摘要信息表中
    infops.addBizDigestInfo(opseno, "id", "序号", true);
    infops.addBizDigestInfo(opseno, "col1", "数据1", true);
    infops.addBizDigestInfo(opseno, "col2", "数据2", true);
    infops.addBizDigestInfo(opseno, "col3", "数据3", true);
    infops.addBizDigestInfo(opseno, "col4", "数据4", true);
    final BizDigestDataBean[] bean = new BizDigestDataBean[5];
    // 根据名称和值保存到数据摘要数据列表中
    bean[0] = new BizDigestDataBean("id", rld.getId());
    bean[1] = new BizDigestDataBean("col1", rld.getCol1());
    bean[2] = new BizDigestDataBean("col2", rld.getCol2());
    bean[3] = new BizDigestDataBean("col3", rld.getCol3());
    bean[4] = new BizDigestDataBean("col4", rld.getCol4());
    dataps.addBizDigestData(opseno, bean);
    BizDigestUtil.save(infops, dataps);// 保存摘要
    tfc.updateObj(rld);// 保存更新信息
    Bizlogger.rolLog("commonRolExample", startTime);// 记录业务日志
}
```

系统开发——视图服务层

- **API 的方式**

```
public void update(final Rollogdemo rld) {
    useAPIUpdate(rld);
}
public void useAPIUpdate(final Rollogdemo rld) {
    final BizDigestInfoParameters infops;
    infops = new BizDigestInfoParameters();
    final BizDigestDataParameters dataps;
    dataps = new BizDigestDataParameters();
    // 开始记录
    Bizlogger.rolLog(new ICommand() {
        public Object execute() {
            final TempleteFacade tfc = new TempleteFacade();
            final long opseno = DbRollUtil.getOpseno();
            // 根据业务流水号，字段名称，字段描述，是否可见添加到数据摘要信息表中
            infops.addBizDigestInfo(opseno, "id", "序号", true);
            infops.addBizDigestInfo(opseno, "col1", "数据1", true);
            infops.addBizDigestInfo(opseno, "col2", "数据2", true);
            infops.addBizDigestInfo(opseno, "col3", "数据3", true);
            infops.addBizDigestInfo(opseno, "col4", "数据4", true);
            final BizDigestDataBean[] bean = new BizDigestDataBean[5];
            // 根据名称和值保存到数据摘要数据列表中
            bean[0] = new BizDigestDataBean("id", rld.getId());
            bean[1] = new BizDigestDataBean("col1", rld.getCol1());
            bean[2] = new BizDigestDataBean("col2", rld.getCol2());
            bean[3] = new BizDigestDataBean("col3", rld.getCol3());
            bean[4] = new BizDigestDataBean("col4", rld.getCol4());
            dataps.addBizDigestData(opseno, bean);
            BizDigestUtil.save(infops, dataps);// 保存摘要
            tfc.updateObj(rld);// 保存更新信息
            return null;
        }
    }, "commonRolExample");
}
```


系统开发——视图服务层

异常处理

视图服务层业务组件方法只需要抛出自己的业务异常就可以，不需要截取异常。

```
public void addTA(CorpInfoDTO  dto) {
    if (null == dto.getAab001()) {
        throw new BusinessException("该单位信息不存在，请确认");
    }
     ……
}
```


系统开发

- 界面视图层
- 请求处理层
- 视图服务层
- 业务组件层

系统开发——业务组件层

业务组件façade层

Façade是一个业务组件包BC中的一个类，该类集中暴露了BC包中的所有业务接口，它是实际的业务处理对象的一个代理。当外部系统或其它务组件需要使用该BC包中的业务和算法时，需要通过调用Façade类中的接口方法实现。

算法对象层

算法类是对一些可能会发生变化、或者可以复用的算法的封装，它也是普通的JavaBean，根据需要开发即可。

业务对象层

业务对象PO是普通的JavaBean，根据需要实现相应的业务逻辑即可。

数据访问对象层

为了支持数据库的可移植性，将数据库访问的代码抽取成DAO类，在数据库移植时，只需要修改DAO类即可。在DAO类中主要实现查询语句的生成和其它一些特定于数据库的操作。DAO类也是普通的JavaBean。

系统开发——业务组件层

查询语句的构造

- **QueryFactory. getHQLInfo(String baseQueryString, String[] baseParaNames, Type[] baseParaTypes, Object bean)**

baseQueryString：查询字符串
baseParaNames ： 占位符变量数组，是hql中所有参数的数组集合
baseParaTypes ： 所有占位符变量的数据类型，和paraNames相对应，是hibernate自己定义的类型，和java中类型相似。
bean ： 存有form表单数据的dto的实例


系统开发——业务组件层

HQLInfo getHQLInfo:

```
StringBuffer hqlBF =
            new StringBuffer("select para from Parameter para where ");

 hqlBF.append(" para.aae140 = :aae140 and para.aaa001 = :aaa001 ")
 .append(" and para.aaa002 like :aaa002 and para.aaa003 = :aaa003 ")
 .append(" and para.aaa004 = :aaa004 and para.aaa005 = :aaa005 ")
 .append(" and para.aae030 = :aae030 and para.aae031 = :aae031 ")
    .append(" order by
     para.aae140,para.aaa001,para.aaa003,para.aae030 ");

String[] paraNames =
                  new String[]{"aae140", "aaa001", "aaa002", "aaa003",
                                "aaa004", "aaa005","aae030","aae031"};

Type[] types = new Type[]{Hibernate.STRING, Hibernate.STRING,
                   Hibernate.STRING,Hibernate.STRING,Hibernate.STRING,
                     Hibernate.DOUBLE,Hibernate.DATE,Hibernate.DATE};

HQLInfo hqlinfo =
    QueryFactory.getHQLInfo(hqlBF.toString(),paraNames,types,
    dto);
```


系统开发——业务组件层

- 异常处理

业务组件方法只需要抛出自己的业务异常就可以，不需要截取异常。

```
public void add(SysRole sr) {
    if (null == sr.getRoleid()) {
        throw new BusinessException("该角色信息有问题，请确认");
    }
     persistence.saveObj(sr);
}
```


系统与页面开发

- 系 统 设 计
- 页 面 设 计


页面开发

- 页面布局及JSP HEAD
- Grid标签
- 其他常用标签
- JSP标记技术
- 自定义JSP标签


页面开发——JSP HEAD及页面布局

➢ JSP HEAD标签

- **<sicp3:base/>标签与<sicp3:head></sicp3:head>标签同样代表html标签中<head>标签,标签默认定义引用的js与css文件在Config.xml文件中。（sicp3工程默认引入style.jsp与Globals.jsp文件）**
- 如页面中无自定义JS方法,可以直接使用:

```
<html><sicp3:base/><body>...或
<html><sicp3:head/><body>...
```

- 如页面中有自定义JS方法,可以使用:

```
<html><sicp3:head>
        <script type="text/javascript">
      jip--
              function ...
       //-->
       </script>
  </sicp3:head><body>...
```

- 如页面需特定指向其他页面，可使用:

```
<html><sicp3:head>
        <sicp3:base target="top"/>
  </sicp3:head><body>...
```


页面开发——JSP HEAD及页面布局

➢ 页面布局标签

■ editlayout标签

格式化控件布局的标签。放在<table>之后。会将所在table默认分成宽度相等的6列。

- **代码样例：**

```
<table class="tableInput">
<sicp3:editorlayout />
    <tr>
        <sicp3:texteditor property="userid" label="编号" />
        <sicp3:texteditor property="username" label="姓名" />
        <sicp3:texteditor property="age" label="年龄" />
    </tr>
</table>
```

- **代码图例：**


页面开发——JSP HEAD及页面布局

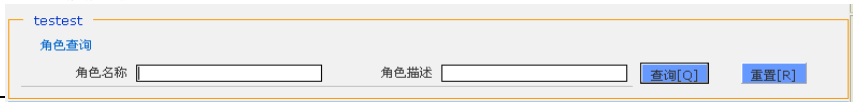
- 页面布局标签
  - **groupbox标签**

把页面信息进行分类。

- 代码样例：

```
<sicp3:groupbox title="testest">
    <sicp3:query action="/roleQueryAction.do?method=query" topic="角色查
    询“  notitle="false">
        <tr>
        <sicp3:texteditor property="roleName" label="角色名称"
    disable="false"></sicp3:texteditor>
        <sicp3:texteditor property="roledesc" label="角色描述"
    disable="false"></sicp3:texteditor>
        </tr>
    </sicp3:query>
</sicp3:groupbox>
```

- 代码图例：




页面开发

- 页面布局及JSP HEAD
- Grid标签
- 其他常用标签
- JSP标记技术
- 自定义JSP标签

页面开发——Grid标签

➢ Grid图例：

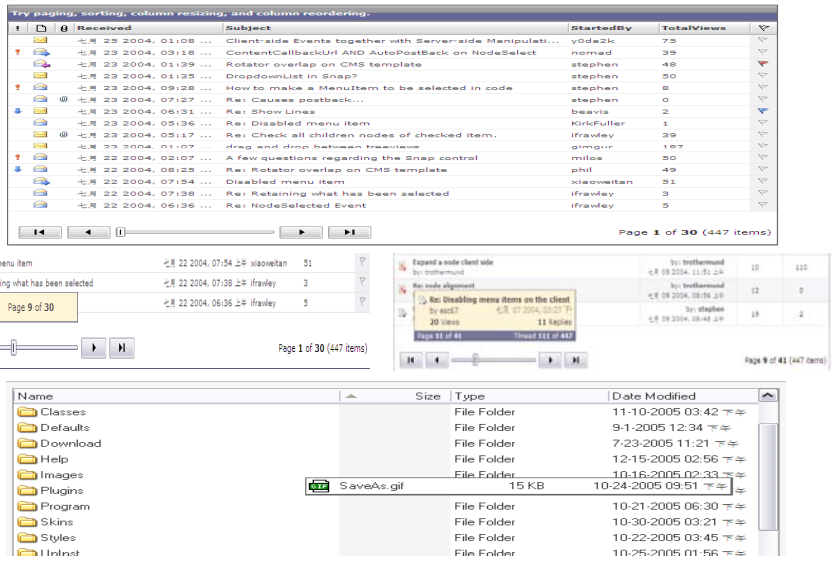
序号	角色名称	操作
4028811a1ddc1dbd011ddc987cf70cca	1	编辑 | 删除
818180891ca16ed1011ca17ee63804c8	BVCRE	编辑 | 删除
4028811a1ddc1dbd011ddc9898ad0ccd	2	编辑 | 删除
4028811a1ddc1dbd011ddc98b1f20d34	3	编辑 | 删除
4028811a1de0d0b6011de0da834b0425	testcp3	编辑 | 删除
guestrole	访客角色	编辑 | 删除
publicrole	公共角色	编辑 | 删除
402881681d8971ea011d89737178026a	氨基督教	编辑 | 删除
402881e518348d61011834ec97ad1248	部门1	编辑 | 删除
402881e518348d61011834f0418b135d	部门1	编辑 | 删除

当前是第1页 共2页 （共19项） |

序号	角色名称	操作
4028811a1ddc1dbd011ddc987cf70cca	1	修改 | 取消
	* 角色描述:	1
818180891ca16ed1011ca17ee63804c8	BVCRE	编辑 | 删除
4028811a1ddc1dbd011ddc9898ad0ccd	2	编辑 | 删除
4028811a1ddc1dbd011ddc98b1f20d34	3	编辑 | 删除
4028811a1de0d0b6011de0da834b0425	testcp3	编辑 | 删除
guestrole	访客角色	编辑 | 删除
publicrole	公共角色	编辑 | 删除
402881681d8971ea011d89737178026a	氨基督教	编辑 | 删除
402881e518348d61011834ec97ad1248	部门1	编辑 | 删除
402881e518348d61011834f0418b135d	部门1	编辑 | 删除

1 2 当前是第1页 共2页 （共19项） |

页面开发——Grid标签



页面开发——Grid标签

➢ 标签特点

■ 多种数据请求方式

- Client：一次请求获得所有的数据，然后在页面上通过js控制数据的组织、显示。
- Callback：通过callback请求参数向后台发出请求以获取数据，然后在页面上显示。比如翻页等操作。
- Server：通过刷新整个页面达到向后台发送请求获取数据的目的，得到数据后在页面上重新组织、显示。

■ 页面元素的控制

Grid标签基本上可以控制页面上所有元素的显示方式（格式、位置、**css**控制、模版控制等）

在这里边有其要说明的是模版定制，可以显示出非常优美、丰富、意想不到的页面。

页面开发——Grid标签

➢样例说明

■ 样例代码

```
<!-- grid标签 -->
<xart:grid id="Grid1" pagerButtonWidth="41" pagerButtonHeight="22" sliderHeight="22" sliderWidth="100"
sliderGripWidth="9" sliderPopupOffsetX="100" width="100%" height="260" allowHorizontalScrolling="false"
pagerStyle="Numbered" requestURI="/samples/editGridAction.do?method=query" runningMode="Callback"
pageSize="10" allowEditing="true" enableViewState="true"  pagerTextCssClass="PagerText"
editOnClickSelectedItem="false" pagerImagesFolderUrl="images/pager/" allowMultipleSelect="true"
cssClass="Grid" footerCssClass="GridFooter" headerCssClass="GridHeader" autoCallBackOnInsert="true"
sliderPopupClientTemplateId="SliderTemplate" autoCallBackOnUpdate="true" autoCallBackOnDelete="true">
    <xart:gridlevels>
        <xart:gridlevel dataMember="querydata" showTableHeading="false" showSelectorCells="false"
        columnReorderIndicatorImageUrl="/images/grid/reorder.gif" hoverRowCssClass="HoverRow"
        headingSelectorCellCssClass="SelectorCell" headingCellCssClass="HeadingCell" rowCssClass="Row"
        headingRowCssClass="HeadingRow" headingTextCssClass="HeadingCellText" dataCellCssClass="DataCell"
        selectedRowCssClass="SelectedRow" sortAscendingImageUrl="/images/asc.gif" sortImageWidth="9"
        sortDescendingImageUrl="/images/desc.gif" sortImageHeight="9" editCellCssClass="EditDataCell"
        editFieldCssClass="EditDataField" selectorCellWidth="10" alternatingRowCssClass="AlternatingRow"
            insertCommandClientTemplateId="InsertCommandTemplate"
            editCommandClientTemplateId="EditCommandTemplate">
            <xart:gridcolumns>
                <xart:gridcolumn dataField="roleid" headingText="序号"
                    allowEditing="false"></xart:gridcolumn>
                <xart:gridcolumn dataField="rolename" headingText="角色名称" ></xart:gridcolumn>
                <xart:gridcolumn dataField="roledesc" headingText="角色描述" visible="false"
                                                        dispalyWhenEditing="true"></xart:gridcolumn>
                <xart:gridcolumn dataField="roledesc" headingText="角色描述" visible="false" >
                </xart:gridcolumn>
                <xart:gridcolumn headingText="操作"
                    dataCellClientTemplateId="EditTemplate"
                    editControlType="EditCommand" width="100" align="Center"></xart:gridcolumn>
            </xart:gridcolumns>
        </xart:gridlevel>
    </xart:gridlevels>
```

页面开发——Grid标签

```
<xart:clienttemplate id="EditTemplate">
    <a href="javascript:editGrid('## DataItem.ClientId ##');">编辑</a> |
    <a href="javascript:deleteRow('## DataItem.ClientId ##','/samples/editGridAction.do?method=delete')">
    删除</a>
</xart:clienttemplate>

<xart:clienttemplate id="EditCommandTemplate">
    <a href="javascript:editRow('/samples/editGridAction.do?method=update');">修改</a> |
    <a href="javascript:Grid1.EditCancel();">取消</a>
</xart:clienttemplate>

<xart:clienttemplate id="InsertCommandTemplate">
    <a href="javascript:insertRow('/samples/editGridAction.do?method=add');">插入</a> |
    <a href="javascript:Grid1.EditCancel();">取消</a>
</xart:clienttemplate>

<xart:clienttemplate id="SliderTemplate">
    <table cellspacing="0" cellpadding="0" border="0">
        <tr>
            <td class="SliderPopup" align="center">Page <b>##DataItem.PageIndex + 1 ##</b>
                                                    of <b>## Grid1.PageCount ##</b></td>
        </tr>
    </table>
</xart:clienttemplate>
</xart:grid>
```

页面开发——Grid标签

- 标签说明

Grid标签由grid，gridlevels，gridlevel，gridcolumns，Gridcolumn，clienttemplate几个标签组成。

- grid标签 ：用于数据的显示（排序、分组等）
- gridlevels标签 ：gridlevel标签的父标签
- gridlevel标签：输出标签中的层信息
- gridcolumns标签：gridcolumn标签的父标签
- gridcolumn标签：输出数据列数据信息
- clienttemplate标签：客户端模板标签，用于在页面上输出静态内容（Grid标签中默认有sliderPopupClientTemplate翻页信息模板和tableHeadingClientTemplate表头信息所引用的模版）

页面开发——Grid标签

重要属性说明

sicp3:grid标签

- id:标签唯一标识符。在js方法中可直接使用

如：

```
// 向后台发送ajax请求并返回数据
function srchRole()
 {
  var param = getFormValue(document.all('editGridForm'));
  Grid1.AjaxCallback(param);
 }
```

- runningMode:标签的数据访问方式：callback；client；server
- pageSize:每页数据行数。
- requestURI:标签翻页查询请求的URI地址，请求为获得grid数据的真实请求。如上例中requestURI=“/samples/editGridAction.do?method=query”与query标签请求获得grid数据的请求相同。
- pageStyle:分页样式。Numbered-数字；Slider-拖动条。
- allowBatchUpdate:是否批量提交。allowBatchUpdate=“1”为批量提交。

页面开发——Grid标签

sicp3:gridlevel标签

- dataMember:数据源信息标示，可选择的数据源:xml,xls文件以及数据库数据，默认名为querydata。
- selectorCellsType：选择列的显示方式，默认是图片，当值为checkbox时则为选择框。
- showSelectorCells：是否可显示选择单元格，为true时可用；默认为空。
- insertCommandClientTemplateId：插入数据的一般客户端模板。属性值与clienttemplate标签的id值相对应。
- editCommandClientTemplateId：编辑数据的一般客户端模板。属性值与clienttemplate标签的id值相对应。
- pageQueryData:多grid时区分别的grid用。

页面开发——Grid标签

- **sicp3:gridcolumn标签**

- dataField：字段信息，与查询出的数据相对应。
- headingText：头部显示该列信息文本。
- allowEditing：是否允许编辑列,默认为：true，允许编辑此列。
- visible：是否显示此列,默认为：true，显示此列。
- dispalyWhenEditing： 当visible=“false”时，编辑时是否显示此列，为true时显示；默认为false。
- editControlType：单元格被编辑时的显示方式（TextBox、TextArea、EditCommand等）。
- dataCellClientTemplateId：单元格所引用的客户端模版标识名。

页面开发——Grid标签

以上示例中属性很多，写起来效率不是很高，有鉴于此，在grid、gridlevel标签中引入了tagTemplateId这个属性。该属性值所代表的是一系列属性的集合。这一系列属性是在tagPropertyInitConfig.xml文档中定义的。

tagTemplateId属性值对应tagPropertyInitConfig.xml文件中<bean>的id值。

```
<!-- grid标签 -->
<xart:grid id="Grid1" tagTemplateId="grid" requestURI="/samples/editGridAction.do?method=query">
    <xart:gridlevels>
        <xart:gridlevel dataMember="querydata" tagTemplateId="gridlevel" hoverRowCssClass="HoverRow"
            insertCommandClientTemplateId="InsertCommandTemplate"
            editCommandClientTemplateId="EditCommandTemplate">
            <xart:gridcolumns>
                <xart:gridcolumn dataField="roleid" headingText="序号"
                    allowEditing="false"></xart:gridcolumn>
                <xart:gridcolumn dataField="rolename" headingText="角色名称"></xart:gridcolumn>
                <xart:gridcolumn dataField="roledesc" headingText="角色描述"></xart:gridcolumn>
                <xart:gridcolumn headingText="操作"
                    dataCellClientTemplateId="EditTemplate"
                    editControlType="EditCommand" width="100" align="Center"></xart:gridcolumn>
            </xart:gridcolumns>
        </xart:gridlevel>
    </xart:gridlevels>
    <xart:clienttemplate id="EditTemplate">
        <a href="javascript:editGrid('## DataItem.ClientId ##');">编辑</a> |
        <a href="javascript:deleteRow('## DataItem.ClientId ##','/samples/editGridAction.do?method=delete')">
        删除</a>
    </xart:clienttemplate>
    <xart:clienttemplate id="EditCommandTemplate">
        <a href="javascript:editRow('/samples/editGridAction.do?method=update');">修改</a> |
        <a href="javascript:Grid1.EditCancel();">取消</a>
    </xart:clienttemplate>
    <xart:clienttemplate id="InsertCommandTemplate">
        <a href="javascript:insertRow('/samples/editGridAction.do?method=add');">插入</a> |
        <a href="javascript:Grid1.EditCancel();">取消</a>
    </xart:clienttemplate>
</xart:grid>
```


页面开发

- 页面布局及JSP HEAD
- Grid标签
- 其他常用标签
- JSP标记技术
- 自定义JSP标签


页面开发——其他常用标签

基本标签

body标签

该标签包装了HTML的body标记,并且使光标置于带有form的第一个可编辑区域上。

如果在body的onload事件中初试化页面，可以在页面中覆盖page_init()的javascript函数，进行页面初试化。

```
<sicp3:body>
                ...
</ sicp3:body>
```

form标签

该标签产生HTML语句：<form name=… action=……> ……</form>，创建了表单。当用户单击提交按钮时，表单将把数据发送到 ACTION 标签属性列出的 URL。METHOD 标签属性的值决定了将服务器发送到服务器的方式。

```
<sicp3:form action="/uiTestAction" method="post"
>
                ...
</sicp3:form>
```

页面开发——其他常用标签

- error标签

<sicp3:errors />创建一个errors标签。使用该标签，在出现业务异时，可将抛出的异常显示在对话框上。该标签没有属性，使用时在jsp页面上</sicp3:body>之前写上<sicp3:errors />即可。

```
<sicp3:errors />
```

- bottom标签

<sicp3:bottom />该标签用来显示页面尾部信息。放**</form>**标签之后。

```
<sicp3:bottom />
```


页面开发——其他常用标签

- 显示标签
- title标签

该标签封装了特定格式的页面主题。

```
<sicp3:title title="测试title" />
```

- tabletitle标签

该标签用来显示表头信息。

```
<sicp3: tabletitle title="测试tabletitle " />
```

页面开发——其他常用标签

- Operator标签、Operdate标签和Operorg标签
(labelOperator标签、labelOperdate标签和labelOperorg标签)

Operator标签、Operdate标签和Operorg标签 分别封装了操作员信息、操作日期信息、操作机构信息。

```
<table class="tableInput">
    <sicp3:editorlayout />
        <tr>
            <td>操作员</td>
            <td><sicp3:operator property="userid" /></td>
            <td>操作日期</td>
             <td><sicp3:operdate property="userid" /></td>
             <td>操作机构</td>
             <td><sicp3:operorg property="userid" /></td>
        </tr>
</table>
```

页面开发——其他常用标签

- 录入标签
- hidden标签

用来隐藏一个元素，不被用户看到。只为传递值而用。

```
<sicp3:hidden property="stringData" value=""/>
```

- textbuttoneditor标签

带有按钮和label的输入框标签


```
        <table>
                <sicp3:editorlayout />
                <tr>
                <sicp3:textbuttoneditor          property="aac001"
label="个人编号"
onRbuttonclick="
showEmployeeList(document.getElementById('aac001').value)"/>
                </tr>
        </table>
```

页面开发——其他常用标签

- text标签（texteditor标签）

该标签类似struts的text标签，但增加了校验功能，并缓存状态，增加了mask属性，该属性用来定义该字段的类型（date"为日期、"email"为邮件、"card"为身份证、"url"为http地址,"######"为邮编,"money"为钱等格式）。

```
<table class="tableInput">
<sicp3:editorlayout />
        <tr>
                <td>普通text</td>
                <td><sicp3:text property="userid" /></td>
                <td>日期</td>
                <td><sicp3:text property="userid" mask="date"
id="aaa"/></td>
                <td>url</td>
        <td><sicp3:text property="userid" mask="url" disable="false"
/></td>
        </tr>
</table>
```

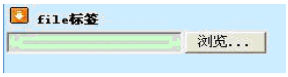

页面开发——其他常用标签

- textarea标签(textareaeditor标签)

<sicp3:textarea>产生HTML语句：<textarea>XXXXX</textarea>，指定多行文本输入控件。

```
<sicp3:tabletitle title="textarea标签" />
<table class="tableInput">
        <sicp3:editorlayout />
        <tr>
                <td height="0" align="center">textarea例子</td>
                <td width="90%"><sicp3:textarea cols="12" rows="5"
                        property="operatorname" /></td>
        </tr>
</table>
```

页面开发——其他常用标签



file标签

创建文件上载控件，该控件带有一个文本框和一个浏览按钮。用户可以在文本框中输入文件路径或者单击浏览按钮在文件系统中浏览。

- 注意：
  - 该标签必须出现在 FORM 元素内。
  - 必须为该标签指定 property 标签属性的值。
  - FORM 元素 METHOD 标签属性的值必须设置为 post。
  - FORM 元素 entype属性的值必须设置为 multipart/form-data。
  - 要处理上载到服务器的文件，服务器端进程必须可以处理 multipart/form-data 的提交。

```
<sicp3:form action="/uiTestAction" method="post" enctype="multipart/form-data" >
        <sicp3:title title="file标签" />
        <table>
                <sicp3:editorlayout />
                <tr>
                <td><sicp3:file property="userid" size="20" /></td>
                </tr>
        </table>
</sicp3:form>
```


页面开发——其他常用标签

选择标签

codeselect标签（codeselecteditor标签）

codeselect标签主要是处理系统启动加载的代码表，框架在系统初始化时，加载代码表的内容。业务开发人员在开发页面时，可以利用框架提供的标签库来实现select功能。Codeselect也支持自定义数据源，如下所示。如果业务需要建立新的永久代码表，可以在action中将TreeMap类型list放入ServletContext中，其key值是type值，如aab001，这样aab001就成为代码了。

页面开发——其他常用标签

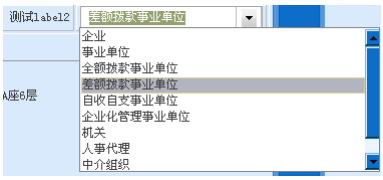
```
<%TreeMap list = new TreeMap();
          list.put("0", "女");
          list.put("1", "男");
          list.put("2", "其他");
   //数据源可以通过action中放置，如request.setAttribute(“data”,list)
          pageContext.setAttribute("data", list);%>
<table  class="tableInput">
          <sicp3:editorlayout />
          <tr>
                    <td>codeselect 可选</td>
                    <td><sicp3:codeselect property="aab019" /></td>
                    <td>codeselect 不可选</td>
                    <td><sicp3:codeselect property="aab019" disable="true"/></td>
          </tr>
          <tr>
                    <td>codeselect 占多列(colspan加在所在列中,不加在标签本身)</td>
                    <td colspan="3"><sicp3:codeselect property="aae140"/></td>
                    <td>codeselect 自定义数据</td>
                    <td><sicp3:codeselect property="operatorname"
dataMeta="data"/></td>
          </tr>
</table>
```


页面开发——其他常用标签

- codelist标签（codelisteditor标签）

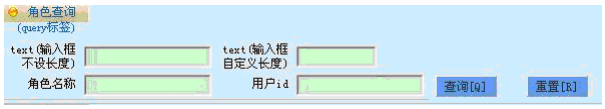
  复杂的代码显示标签，可显示多列，使代码值与代码名称对应显示，codelisteditor标签与codelist标签差别在于，codelisteditor标签可以显示Lable作为标签说明

```
<table class="tableInput">
          <sicp3:editorlayout />
          <tr>
<sicp3:codelisteditor id="codelist3" mode="multi" label="测试label1“
width="120" property="AAB019" selectedValue="21" dropDownStyle="0" />
<sicp3:codelisteditor id="codelist4" label="测试label2" property="AAB019"
selectedValue="22" dropDownStyle="0“ width="120“/>
          </tr>
</table>
```



页面开发——其他常用标签

➢ 查询标签



■ query标签

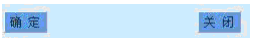
查询容器标签，可支持嵌套text,texteditor等标签。该标签内置“查询[Q]”和“重置[R]”按钮，内置格式包含<table></table>和editlayout标签。

```
<sicp3:query action="/uiTestAction" topic="角色查询(query标签)" notitle="false">
        <tr>
                <td>text(输入框不设长度)</td>
                <td><sicp3:text property="aab019" maxlength="30"label="角色名称"/></td>
                <td>text(输入框自定义长度)</td>
               <td width=“80” ><sicp3:text property=“aab019” maxlength=“30” label=“角色名称
“/></td>
        </tr>
        <tr>
                <sicp3:texteditor property="aae140" maxlength="30" label="角色名称" />
                <sicp3:texteditor property="userid" maxlength="30" label="用户id" />
        </tr>
</sicp3:query>
```


页面开发——其他常用标签

➢ 按钮标签

■ button标签



普通的单独按钮。

```
<table>
<sicp3:editorlayout/>
<tr>
        <td><sicp3:button value="确 定" onclick="alert('确定')"/></td>
        <td><sicp3:button value="关 闭" onclick="closeWindow()"/>
        </td>
</tr>
</table>
```

■ buttons标签

带格式的按钮容器。

```
<sicp3:buttons>
        <sicp3:button value="确 定[Q]" onclick="alert('确定')" />
        <sicp3:button value="关 闭[C]" onclick="closeWindow()" />
        <sicp3:button value="标 志[K]" onclick="alert('标志')" />
</sicp3:buttons>
```

页面开发——其他常用标签

- 地区树标签
  - treeinput标签

<sicp3:treeinput>为地区树标签，可根据自定义actionUrl属性请求获得所需地区树，还可自定义显示文字等。

```
<td>所属行政区划</td>
<td>
<sicp3:treeinput property="aab301" mask="region"
isPopedom="true"
actionUrl="/sysmanage/areatreeAction.do?method=region" />
</td>
```


页面开发

- 注意事项：
  - <sicp3:base/>或者<sicp3:head>标签不能放在含有iframe标签的框架页面中。
  - text相关标签，增加verfyParityBit属性，是否校验身份证最后一位，当mask="card"时使用，为true时不校验;默认为null。
  - **text,codeselect标签中，disable,disabled,readonly说明**

Text标签：
disable：不可JS设置状态；可回显；可通过JS设置value，但前台页面不显示，可通过form传递到后台action中
disabled：可JS设置状态；可回显；可通过JS设置value，前台页面可显示，但不能通过form传递到后台action中
readonly：可JS设置状态；可回显；可通过JS设置value，前台页面可显示，并通过form传递到后台action中
codeselect标签：
disable：不可JS设置状态；可回显；可通过JS设置value，但前台页面不显示，可通过form传递到后台action中
disabled：可JS设置状态；可回显；可通过JS设置value，前台页面可显示，但不能通过form传递到后台action中


页面开发

- grid编辑状态时编辑框事件设置：
  - 编辑框id命名规则：gridid_input_rowid_colid，rowid与colid均从0开始计算，rowid在点击触发编辑事件时以传入
  -

```
function a(rowId){
    id="Grid1_input_"+rowId+"_2";
    document.getElementById(id).value="123";
 }
// 修改数据
function editGrid(rowId) {
    Grid1.Edit(Grid1.GetRowFromClientId(rowId));
    //设置要触发事件的编辑框
    id="Grid1_input_"+rowId+"_1";
   //设置事件JS方法
   document.getElementById(id).onblur=new
                                          a(rowId);
}
```


页面开发

- 页面布局及JSP HEAD
- Grid标签
- 其他常用标签
- JSP标记技术
- 自定义JSP标签

页面开发——JSP标记技术

JSP 2.0 的标记文件类似于 JSP 页，使用 JSP 语法，然后 JSP 容器获取您的 JSP 标记文件，分析这些标记文件，生成 Java 标记处理程序，并自动编译它们。标记文件是从 JSP 页进行调用的，JSP 页使用与 <prefix:tagFileName> 模式相匹配的自定义标记。

为了使标记文件能够被 JSP 容器所识别，标记文件必须使用 .tag 文件扩展名进行命名，并且必须被放置在您的 Web 应用程序的 /WEB-INF/tags 目录中或者 /WEB-INF/tags 的子目录中。

页面开发——JSP标记技术

编写tag文件

```
<%@ tag page Encoding="GBK" body-content="empty" small-icon=""
display-name="jip标记示例" description="jip标记示例"%>
<%@ taglib uri="/WEB-INF/tagDemo.tld" prefix="tag"%>
<%@ attribute name="lable" required="true" description="欢迎词"%>
<%@ attribute name="color" required="false" description="页面颜色"%>
<%@ attribute name="nameId" required="false" description="姓名输入框唯一标识"%>
<%@ attribute name="pwdId" required="false" description="密码输入框唯一标识"%>
<script language="javascript">
   function wel(){
       alert("WELCOME");
    }
</script>
<%
String id = com.lbs.leaf.util.StringHelper.genUUID();
if (color == null || "".equals(color)) {
      if (Integer.parseInt(id.substring(id.length() - 1, id.length())) % 2 == 0) {
            color="red";
       } else {
            color="green";
       }
}
%>
<tag:inputDemo lable="<%=lable %>" color="<%=color %>"
nameId="<%=nameId %>" pwdId="<%=pwdId %>"></tag:inputDemo>
<input type="button" value="welcome" onclick="wel"/>
```


页面开发——JSP标记技术

- JSP页面引用

```
<%@ page language="java" contentType="text/html;
                    charset=GB18030“ pageEncoding="GB18030"%>
<%@ taglib prefix="tags" tagdir="/WEB-INF/tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type“content="text/html;charset=GB18030">
 <title>Insert title here</title>
</head>
<body>
  <tags:demo lable="" />
</body>
</html>
```


页面开发

- 页面布局及JSP HEAD
- Grid标签
- 其他常用标签
- JSP标记技术
- 自定义JSP标签


页面开发——自定义JSP标签

- 开发标签实现类
- 编写标签描述文件
- 修改web.xml文件
- 编写JSP页面文件


页面开发——自定义JSP标签

- 开发标签实现类

  - 处理标签的类必须扩展javax.servlet.jip.TagSupport。
  - TagSupport类的主要属性：
    - parent属性：代表嵌套了当前标签的上层标签的处理类
    - pageContex属性：代表Web应用中的**javax.servlet.jsp.PageContext**对象
  - JSP容器在调用doStartTag或者doEndTag方法前，会先调用setPageContext和setParent方法，设置pageContext和parent。因此在标签处理类中可以直接访问pageContext变量。
  - 在TagSupport的构造方法中不能访问pageContext成员变量，因为此时JSP容器还没有调用。
    setPageContext方法对pageContext进行初始化。


页面开发——自定义JSP标签

- TagSupport处理标签的方法

TagSupport类提供了两个处理标签的方法：

```
public int doStartTag() throws JspException
public int doEndTag()  throws JspException
```

- doStartTag：JSP容器遇到自定义标签的起始标志，就会调用doStartTag()方法 ,doStartTag()方法返回一个整数值，用来决定程序的后续流程。
  - Tag.SKIP_BODY：表示<prefix:someTag>…</prefix:someTag>之间的内容被忽略
  - Tag.EVAL_BODY_INCLUDE：表示标签之间的内容被正常执行

- doEndTag：JSP容器遇到自定义标签的结束标志，就会调用doEndTag()方法。doEndTag ()方法也返回一个整数值，用来决定程序后续流程。
  - Tag.SKIP_PAGE：表示立刻停止执行网页，网页上未处理的静态内容和JSP程序均被忽略，任何已有的输出内容立刻返回到客户的浏览器上。
  - Tag_EVAL_PAGE：表示按照正常的流程继续执行JSP网页


页面开发——自定义JSP标签

- 创建标签处理类
- 引入必需的资源：

```
import javax.servlet.jip.*;
import javax.servlet.http.*;
import java.util.*;
import java.io.*;
```

- 继承TagSupport类并覆盖doStartTag()/doEndTag()方法
- 从ServletContext对象中获取java.util.Properties对象
- 从Properties对象中获取key对应的属性值
- 对获取的属性进行相应的处理并输出结果

页面开发——自定义JSP标签

- 开发标签实现类
- 编写标签描述文件
- 修改web.xml文件
- 编写JSP页面文件

页面开发——自定义JSP标签

- 标签库描述文件，简称**TLD**，采用**XML**文件格式，定义了用户的标签库。**TLD**文件中的元素可以分成**3**类：
  - A.<taglib>：　标签库元素
  - B.<tag>：　标签元素
  - C.<attribute>：标签属性元素
- 标签库元素<taglib>用来设定标签库的相关信息，它的常用属性有：
  - A.shortname：　指定Tag Library默认的前缀名(prefix)
  - B.uri：　设定Tag Library的惟一访问表示符

页面开发——自定义JSP标签

- 标签元素<tag>用来定义一个标签，它的常见属性有：
  - A. name：　　设定Tag的名字
  - B. tagclass：　　设定Tag的处理类
  - C. bodycontent：　设定标签的主体(body)内容
    - 1). empty：表示标签中没有body
    - 2). JSP：表示标签的body中可以加入JSP程序代码
    - 3). tagdependent：表示标签中的内容由标签自己去处理
- 标签属性元素<attribute>用来定义标签的属性，它的常见属性有：
  - A. name：属性名称
  - B. required：属性是否必需的，默认为false
  - C. rtexprvalue：是否支持EL表达式

页面开发——自定义JSP标签

```
<?xml version="1.0" encoding="gb2312"?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library
1.2//EN" "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
<taglib>
        //标签库版本
        <tlib-version>1.2</tlib-version>
        //标签库依赖的JSP版本
        <jsp-version>1.2</jsp-version>
        //标签库名
        <short-name>sicp3</short-name>
        // 设定Tag Library的惟一访问表示符
       <uri>http://sicp3.bjlbs.com.cn/tags/tags-sicp3</uri>
        //关于此标签库的描述信息
       <description>标签库，包括html小标签及sicp3ui大标签<description>
      <tag>
              //定义标签的名
              <name>wizardbutton</name>
             //指出标签处理程序类
    <tag-class>com.lbs.leaf.ui.taglib.html.WizardButtonTag</tag-class>
              //此标签体的定义
              <body-content>empty</body-content>
              //关于此标签的描述信息
              <description>向导用到的上一步，下一步按钮</description>
       </tag>
</taglib>
```

页面开发——自定义JSP标签

- 开发标签实现类
- 编写标签描述文件
- 修改web.xml文件
- 编写JSP页面文件

页面开发——自定义JSP标签

J2EE 1.3

```
<taglib>
    <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
</taglib>
```

J2EE1.4

```
<jip-config>
  <taglib>
      <taglib-uri>/WEB-INF/tlds/tagDemo.tld</taglib-uri>
      <taglib-location>/WEB-INF/tlds/tagDemo.tld</taglib-location>
  </taglib>
</jsp-config>
```

页面开发——自定义JSP标签

- 开发标签实现类
- 编写标签描述文件
- 修改web.xml文件
- 编写JSP页面文件

页面开发——自定义JSP标签

```
<%@ page contentType="text/html; charset=GBK"
language="java"%>
<%@ taglib uri="/WEB-INF/sicp3.tld" prefix="sicp3"%>
<html>
   <sicp3:body>
      <sicp3:title title="text、texteditor、textarea、textbuttoneditor、
                                                    colortxt标签" />
      <sicp3:tabletitle title="text标签" />
      <table class="tableInput">
          <tr>
             <sicp3:texteditor property="userid" label="普通text"/>
          </tr>
       </table>
    </sicp3:body>
</html>
```

名词解释

❖ AOP

面向方面编程：Aspect Oriented Programming

AOP是OOP的延续，是Aspect Oriented Programming的缩写，意思是面向切面编程。可以通过预编译方式和运行期动态代理实现在不修改源代码的情况下给程序动态统一添加功能的一种技术。

AOP实际是GoF设计模式的延续，设计模式孜孜不倦追求的是调用者和被调用者之间的解耦，AOP可以说也是这种目标的一种实现。

Simiscp3 Thank You !