DBFound 快速开发平台

DBFound 宗旨：解决开发技术复杂、难度高、开发速度慢等问题。
提供快速、便利、高效率的开发平台。

DBFound 简介：通过配置实现前端与后端的交互，实现功能强大、快捷简便的操作方式。只需配置一系列的 Model 实体文件，外界通过访问 Model 文件就能访问数据库，从而进行数据的增删改查。通过 jstl 标签配置就能实现强大而美观的 UI 界面。

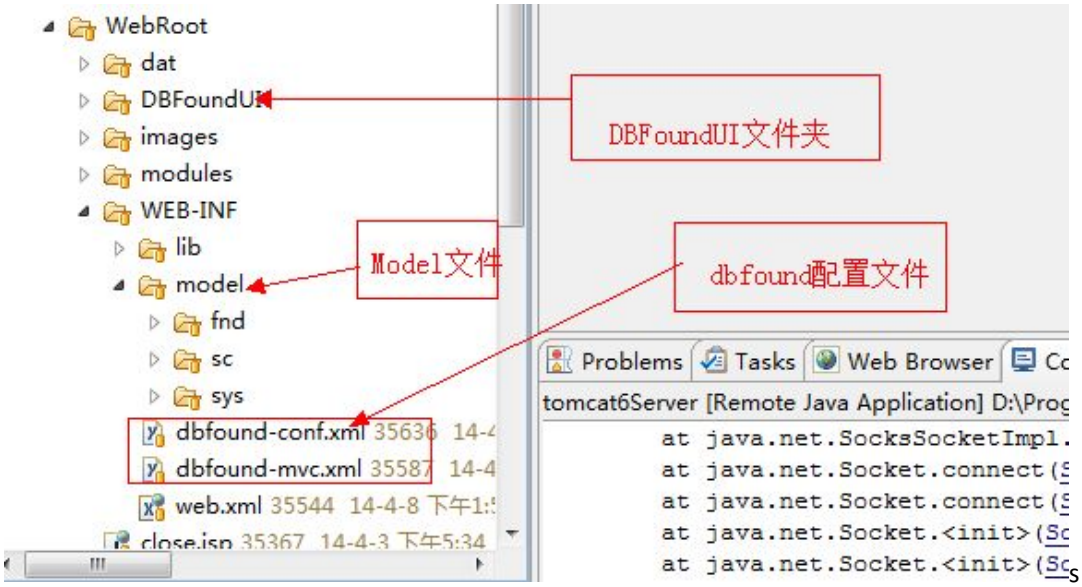
DBFound 标签分为两类：Model 实体标签 和 JSP 标签

JSP 标签又分为： UI 标签 和 控制标签

一、平台搭建 案例演示

新建一个 web 项目，如下图所示目录结构部署 DBFound 框架。

Model 实体文件的跟目录为 WEB-INF 下面的 model 文件夹。persistence.xml 放在 WEB-INF 下。DBFoundUI 放在 WebRoot 下。



图一：框架部署结构图

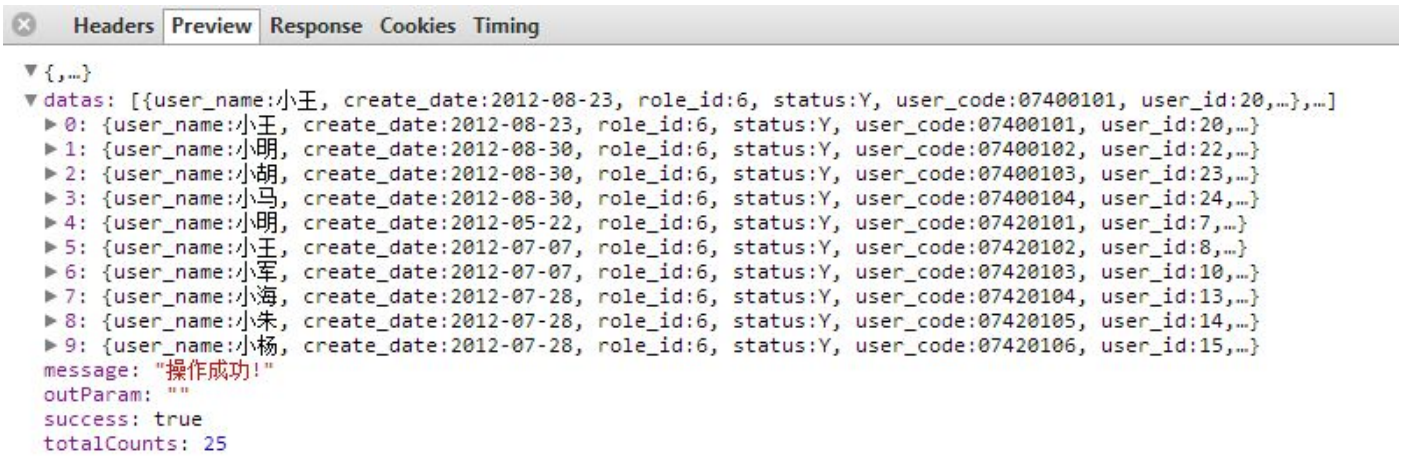
1.1 query 实例演示

```
<?xml version="2.0" encoding="UTF-8"?>
<model xmlns:d="http://dbfound.googlecode.com/model"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            "xsi:schemaLocation="http://dbfound.googlecode.com/model
                http://dbfound.googlecode.com/svn/tags/v2/model.xsd">
    <query name="load" pagerSize="10">
        <sql>
          <![CDATA[
            SELECT
                t.teacher id,
                t.teacher code,
                t.teacher_name,
                t.email,
                t.telphone num,
                t.last_update_by,
                DATE FORMAT(t.create date,'%Y-%m-%d') create date,
                DATE FORMAT(t.last update date,'%Y-%m-%d') last update date
            FROM fnd_teacher t
            #WHERE CLAUSE#
            order by t.teacher code
          ]]>
        </sql>
        <filter name="timefrom" express="create date &gt;= ${@timefrom} " />
        <filter name="timeto" express="create_date &lt;= ${@timeto}" />
        <filter name="teacher code" express="teacher code like ${@teacher code}" />
        <filter name="teacher name" express="teacher name like ${@teacher name}" />
    </query>
</model>
```

上述为一个查询 表 fnd_teacher 的一个 model 文件,目录：model/fnd/teacher.xml，有四个过滤条件 timefrom、timeto、teacher_code、teacher_name。当前台传入这些参数时，过滤条件生效。

我们可以通过 http://localhost:8080/dbfound/fnd/teacher.query!load 访问。

得到如同下图的 json 数据。执行 sql 自动根据传输过来的参数，拼接过滤条件。



1.2 execute 实例演示

```
<?xml version="2.0" encoding="UTF-8"?>
<model xmlns:d="http://dbfound.googlecode.com/model"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                "xsi:schemaLocation="http://dbfound.googlecode.com/model
                http://dbfound.googlecode.com/svn/tags/v2/model.xsd">

    <execute name="add">
        <param name="teacher code" dataType="varchar" />
        <param name="teacher name" dataType="varchar" />
        <param name="email" dataType="varchar" />
        <param name="telphone num" dataType="varchar" />
        <param name="user id" dataType="number" scope="session" />
        <sqls>
            <collisionSql
                where="exists (select 1 from fnd teacher where teacher code= ${@teacher code})"
                message="CODE:#{@teacher code}已经使用！" />
            <executeSql>
               <![CDATA[
                INSERT INTO fnd teacher
                    (teacher code,
                    teacher name,
                    email,
                    telphone num,
                    create by,
                    last update by,
                    create date,
                    last update date)
                VALUES (${@teacher code},
                    ${@teacher name},
                    ${@email},
                    ${@telphone num},
                    ${@user id},
                    ${@user id},
                    NOW(),
                    NOW())
                ]]>
            </executeSql>
        </sqls>
    </execute>
</model>
```

上述为一个添加表 fnd_teacher 的一个 model 文件,目录：model/fnd/teacherAdd.xml，参数前台传入，用${@参数名}获取。

我们可以通过 http://localhost:8080/dbfound/fnd/teacherAdd.execute!add 访问。

得到一个 json 的反馈对象，告知是否操作成功。

*****注解

fnd/teacherAdd 表示访问 model 目下下面 fnd 子目录下面的 teacherAdd.xml。

.execute 说明访问 execute 对象。

!add 表示访问 名为 add 的 execute 对象。

******

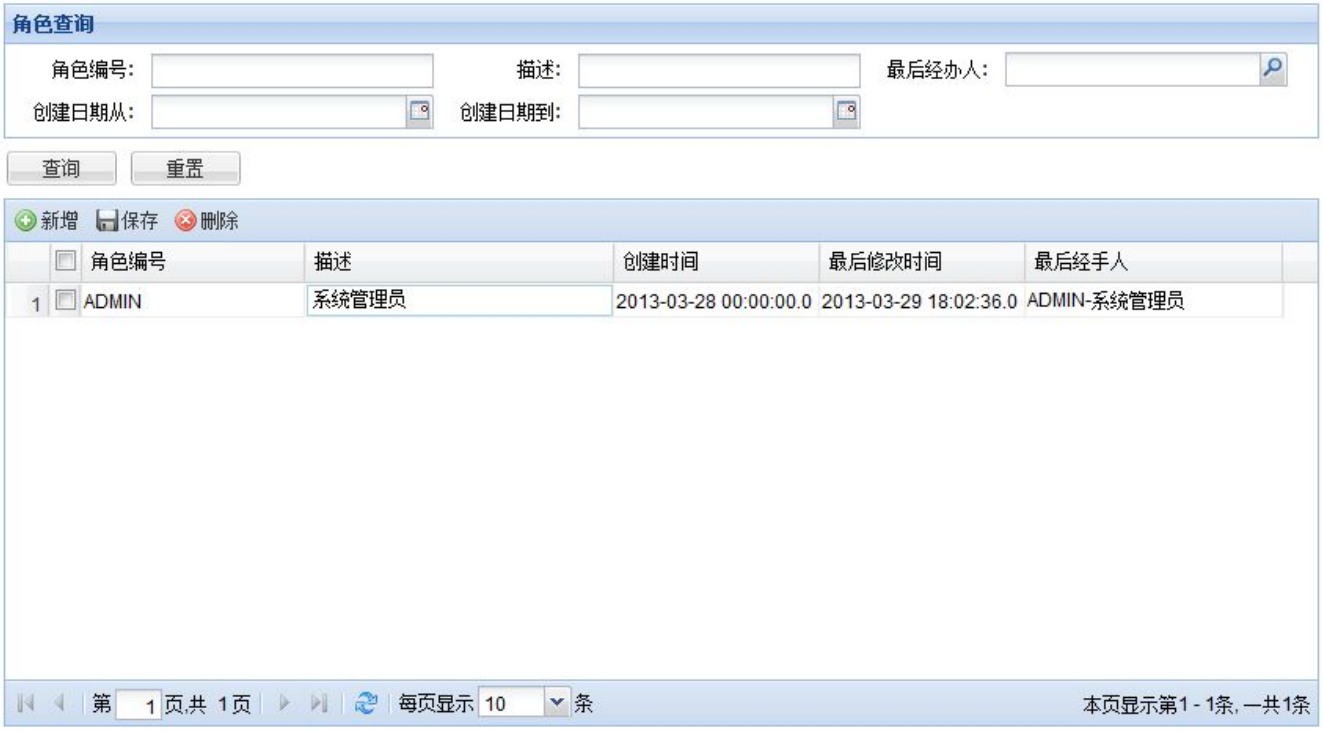
query 案例一样

1.3 UI 界面展示

UI 布局提供八大 UI 组件：grid、form、buttonGroup、tab、panel、tree、treeGrid、menu。

注：对于复杂布局，建议先用 div 布局，然后用组件填充。

下图为简单的 form-button-grid 布局



实现代码如下：

```
<d:form id="queryForm" title="角色查询" labelWidth="80">
     <d:line columnWidth="0.33">
          <d:field name="role code" upper="true" width="180" editor="textfield" prompt="角色编号"/>
           。。。。。。。。。。。。
     </d:line>
     <d:line columnWidth="0.33">
          <d:field name="timefrom" width="180" editor="datefield" prompt="创建日期从" />
          <d:field name="timeto" width="180" editor="datefield" prompt="创建日期到" />
     </d:line>
</d:form>
<d:buttonGroup>
     <d:button id="query" title="查询" click="query" />
     <d:button title="重置" click="reset" />
</d:buttonGroup>
<d:grid id="roleGrid" isCellEditable="isCellEditable" queryForm="queryForm" model="sys/role" >
     <d:toolBar>
          <d:gridButton type="add"/>
          <d:gridButton type="save"/>
          <d:gridButton type="delete" />
     </d:toolBar>
     <d:columns>
          。。。。。
          <d:column name="create date" prompt="创建时间" width="120" />
          <d:column name="last update date" prompt="最后修改时间" width="120" />
          <d:column name="last_update_user" prompt="最后经手人" width="150" />
     </d:columns>
</d:grid>
```

二、Model 实体标签

2.1 model 标签

```
<model persistenceUnit="mysql" xmlns="http://dbfound.googlecode.com/model">
```

描　述：Model 为最顶层 root 标签 命名空间为 http://dbfound.googlecode.com/model
属　性：persistenceUnit：用来指定使用那个数据库连接单元，
与 persistence.xml 中对应一致。可以不设置（调用默认单元）。
子标签：<param> 、 <execute> 、 <query>

2.2 param 标签

```
<param name="user_id" dataType="number" scope="session" />
```

描　述：param 标签可以在<model>、<execute>、<query>中定义,作为 sql 执行的参数，
modle 中定义的参数为全局的，所有的 execute 和 query 中都能够使用。
属　性：name：必须，作为唯一标示
dataType：声明为那种数据类型（varchar、number、file）
scope：声明从那个空间里面取值（session、request、param、page、outParam）
sourcePath：声明取值的路径，遵循 el 表达式规范 如 session.user.id
autoSession：是否将参数放入到 session 中（true，false）
autoCookie：是否将参数放入到 cookie 中（true，false）
fileSaveType：文件是保存到数据库，还是文件夹（db，disk）
ioType：声明是输入，还是输出参数（in，out）
Value： 指定默认值

注　意：param 取值有两种方式 ${@paramName} 和 #{@paramName}。前者为动态设值，后者为静态设值。

2.3 query 标签

```
<query name="load" pagerSize="10">
```

描　述：query 为 model 标签的子标签，在 model 中可以定义多个 query，用 name
属性区分，一个 model 下 面只能有一个默认的 query 对象，及没有 name 属性。
属　性：name：唯一标示 model 中的多个 query 对象
pagerSize：设置分页大小
子标签：<param>、 <sql> 、<filter>

2.4 sql 标签

```
<sql>
   select * from sys_user where user_id = ${@user_id}
</sql>
```

描　述：sql 为 query 的子标签，用于定义 query 中对应的 sql 语句；
语句中可以用${@param 名}取定义的 param 参数。

2.5 filter 标签

```
<filter name="user_id" express="user_id = ${@user_id}" >
```

描　述：sql 为 query 的子标签，用于定义 query 中对应的过滤条件
属　性：name：作为唯一标示，当前台传过来的参数与 name 属性匹配时
过滤条件生效。
Express：按照值进行过滤，express 中可以遵循${@变量名}取值
声　明：express 取值默认是从 parameter 里面取值，如果 param 没有，则
寻找 request，如果还是没有，选择 currentPath 里面的值。
#WHERE_CLAUSE# 和 #ADN_CLAUSE# 为 express 的落脚处

2.6 execute 标签

```
<execute name="add">
```

描　述：execute 为 model 标签的子标签，在 model 中可以定义多个 execute，
用 name 属性区分，一个 model 下 面只能有一个默认的 execute 对象，
及没有 name 属性。
属　性：name：唯一标示 model 中的多个 execute 对象
子标签：<param>、 <sqls>

2.7 sqls 标签

```
<sqls>
      <executeSql></executeSql>
      ...子标签...
</sqls>
```

描　述：sqls 为 execute 中需要执行的 sql 集合
属　性：无
子标签：<executeSql>、<batchSql>、<whenSql>、<querySql>、<collisionSql>、
<javaAction>、<execute>、<query>（调用 model 中的 query 对象）

2.8 executeSql 标签

```
<executeSql>
    insert into sys_user(user_id)values(${@user_id})
</executeSql>
```

描　述：executeSql 为 sqls 的子标签，用于定义 execute 执行对应的 sql 语句；
语句中可以用${@param 名}取定义的 param 参数。

2.9 batchSql 标签

```
<batchSql cursor="select * from sys_user">
      <executeSql></executeSql>
      ...子标签...
</batchSql>
```

描　述：把 cursor 作为查询游标，查询出多行记录，然后循环的执行其子标签。
语句中可以用${@param 名}取定义的 param 参数。查询出来的结果
将放入到 outParam 这个 map 中，参数 param 可以取到里面的值。
子标签：<executeSql>、<batchSql>、<whenSql>、<querySql>、<collisionSql>、
<javaAction>、<execute>、<query>（调用 model 中的 query 对象）

2.10 whenSql 标签

```
<whenSql when="${@user_id} = 1">
      <executeSql></executeSql>
      ...子标签...
</whenSql>
```

描　述：把 when 作为条件，当成立的时候执行其子标签。
语句中可以用${@param 名}取定义的 param 参数。
子标签：<executeSql>、<batchSql>、<whenSql>、<querySql>、<collisionSql>、
<javaAction>、<execute>、<query>（调用 model 中的 query 对象）

2.11 querysql 标签

```
<querySql>
    Select * from sys_user
</querySql>
```

描　述：querySql 为 sqls 的子标签，用于定义 execute 执行对应的 sql 语句；
语句中可以用${@param 名}取定义的 param 参数。查询出来的结果对对应参数进行赋值操作。

2.12 collisionSql 标签

```
<collisionSql where="${@user_id} = 1"  message="user_id 为 1 的数据已存在！">
```

描　述：把 where 作为条件，当成立的时候抛出 message 定义的消息。
属　性：where：碰撞条件
Message：抛出的错误消息

2.13 java 标签

```
<java className="com.nfwork.test" method="sayHello">
```

描　述：调用 java 方法，对应的 class 要继承 ActionSupport 类。从来可以得到 pagerContext、params 等 dbound 内置对象
属　性：className：类名
method：方法名

三、jsp 控制标签

```
<%@ taglib uri="dbfound-tags" prefix="d"%>
<d:controlBody>
   <d:batchExecute modelName="sys/role" sourcePath="table"></d:batchExecute>
</d:controlBody>
```

上述为一个简单的 jsp 控制标签实例。controlBody 声明一个独立的事务处理。

在带展现层的 jsp，应该用下面这种方式声明事务。

```
<d:initProcedure>
  <d:query modelName="sys/role" name="load" rootPath="user_ds" />
</d:initProcedure>
```

3.1 execute 标签

```
<d:execute modelName="sys/role" name="add" sourcePath="user_ds" />
```

描　述：execute 标签用于 JSP 中调用 model 实体中定义的 execute 对象。
属　性：modelName：调用那一个 model 文件
Name：调用 model 文件中的哪一个 execute 对象
sourcePath：取数据的路径

3.2 query 标签

```
<d:query modelName="sys/role" name="load" rootPath="user_ds" />
```

描　述：query 标签用于 JSP 中调用 model 实体中定义的 query 对象。
属　性：modelName：调用那一个 model 文件
name：调用 model 文件中的哪一个 query 对象
rootPath：取存储数据结构的路径
sourcePath：读取数据参数的路径

3.3 batchExecute 标签

```
<d:batchExecute modelName="sys/role" sourcePath="table">
```

描　述：batchExecute 标签用于 JSP 中批量执行 model 实体中定义的 execute 对象。
属　性：modelName：调用那一个 model 文件；
name：调用 model 文件中的哪一个 execute 对象，当 name 不指定时根据每一行的"_status"状态为判断调用那个 execute 对象，NEW 调用 add, OLD 调用 update,没有指定状态位时调用 addOrUpdate；
sourcePath：读取数据参数的路径；

3.4 forward 标签

```
<d:forward modelName="sys/role" name="load" />
```

描　述：forward 标签用于 JSP 中跳转到实体中定义的 query 对象。
属　性：modelName：调用那一个 model 文件；
name：调用 model 文件中的哪一个 query 对象；

3.5 其它控制标签

dbfound 继承了 jstl 标签，并做了扩展 使其支持解析 json 数据格式。

```
<d:forEach items="" var="" sourcePath="" ></d:forEach>
<d:if test=""></d:if>
<d:choose>
   <d:when test=""></d:when>
   <d:otherwise></d:otherwise>
</d:choose>
```

等等其它标签。

四、jsp UI 标签

4.1 网格容器 grid

```
<d:grid id="userGrid" model="sys/user" autoQuery="true" >
  <d:toolBar>
      <d:gridButton type="add" afterAction="initDefaultValue"/>
      <d:gridButton type="save"/>
  </d:toolBar>
  <d:columns>
      <d:column name="user_code"editor="textfield" prompt="用户编号" width="120" />
      <d:column name="user_name" editor="textfield" prompt="姓名" width="120" />
      <d:column name="last_update_user" prompt="最后经手人" width="110" />
  </d:columns>
</d:grid>
```

扩展对象函数：

```
function userGrid.query();   //grid 数据加载
function userGrid.addLine(json,autoCommit); //grid 添加一行，
                                        //json 为数据,autoCommit 是否提交修改
function userGrid.getCurrentRecordData(); //得到当前行记录，返回 json 对象
function userGrid.setCurrentRecordData(json); //设置当前行数据，并且提交修改
function userGrid.initCurrentRecordData(json); //设置当前行数据，不提交修改

function userGrid.getSelectionsData(returnJson);
//选中的行记录，当参数为 true 时返回 Object 数组，false 返回 String 字符串

function userGrid.getModifiedData(returnJson);
//发生修改的行记录，当参数为 true 时返回 Object 数组，false 返回 String 字符串

function userGrid.getSelectionsModifiedData(returnJson);
//选中并且修改的记录，当参数为 true 时返回 Object 数组，false 返回 String 字符串
```

属　性：id：声明一个 grid 对象，全局唯一；

model：绑定后台逻辑运算的 model 文件，如 sys/user；

autoQuery ：自动查询；

queryForm：对应过滤的查询表单；

isCellEditable：判断单元格是否可以编辑，对应 js 函数，返回 true 可以编辑。

```
function isCellEditable(col, row,name,record) {
   //col 列号  row 行号  name 为列名  record 对应行记录
}
```

pagerSize：每页显示多少行；

queryUrl：查询对应的服务器地址，优先级高于 model 的查询地址；

selectable：是否可以行选择，默认为 true；

singleSelect：是否单行选择，默认 false
forceFit：列是否强制占满窗体，默认 true
viewForm：行展示 form，展示一行记录；
title：对应标题；
navBar：是否显示分页工具条，默认为 true
rowNumber：是否显示行号，默认为 true
height：高度；
width：宽度，不设置时 屏幕自适应；

子标签：gridButton 定义 grid 左上角的控制按钮。

属性：id：唯一标示；
title：对应显示文本；
action：服务器响应 url；
disabled：是否失效；
icon：对应图标；
type：用于系统内置类型，如：save，add，delete，excel
beforeAction：响应之前调用的 js 函数，return true 继续执行
type = add 类型

```
function beforeAction(grid,button){
    //grid 当前grid对象
    //button 当前点击的button对象
}
```

其它类型

```
function beforeAction(records,grid,button){
    //records 选择的所以行记录
    //grid 当前grid对象
    //button 当前点击的button对象
}
```

afterAction：响应之后调用的 js 函数，return true 继续执行
type = add 类型

```
function afterAction(record,grid,button){
    //record 新建的行记录
    //grid 当前grid对象
    //button 当前点击的button对象
}
```

其它 类型

```
function afterAction(records,resObj,button){
    //record 选择的所有行记录
    //服务器反馈的ResponseObject对象
    //button 当前点击的button对象
}
```

子标签：column 定义 grid 对应的列。
属性：id：唯一标示；
name：对应后台取值名称；

prompt：对应文本描述；
sortable：是否支持排序；
width：对应宽度
editor：对应编辑器（textfield,datefield,numberfield,password，datetimefield,combo,datetimefield,lov,lovcombo）；
upper：是否输入大写 true 为大写，false 为不限制；
required：是否必须；
options：下拉列表对应的数据集，editor 为 combo 时起效。
displayField：显示的列，editor 为 combo 时起效。
valueField：实际值，editor 为 combo 时起效。
mode：下拉框的数据模型是 remote 或 local
lovUrl：lov 对应的页面。
lovWidth：打开窗口宽度。
lovHeight：打开窗口高度。
editable：对应下拉框组件 combo、lov、lovcombo 是否可手动编辑。
align： 对其方式 center、left、right。
sortable：是否可排序。
allowDecimals 允许为负数，对 numberfield 生效。
allowNegative 允许为小数，对 numberfield 生效。
renderer：渲染，调用 js 函数，返回显示的值.

```
function(value, cellmeta, record, row, col, store){
    //value 对应的值，record 对应行记录
}
```

vtype：校验类型 dbfound 内置五种类型。课根据 Ext 接口自定义。

1.alpha //只能输入字母，无法输入其他（如数字，特殊符号等）
2.alphanum//只能输入字母和数字，无法输入其他
3.email//email验证，要求的格式是"usc@sina.com"
4.url//url格式验证，要求的格式是http://www.sina.com
5.ip //ip 地址

4.2 表单容器 form

```
<d:form id="viewForm" labelWidth="80">
    <d:line columnWidth="0.5">
        <d:field  name="function_code" width="150" editor="textfield" prompt="功能编号" />
        <d:field  name="function_des"  width="150" editor="textfield" prompt="功能描述" />
    </d:line>
    <d:line columnWidth="0.5">
        <d:field name="jsp_pager" width="150" editor="textfield" prompt="对应jsp页面" />
        <d:field name="image" width="150" editor="textfield" prompt="图标" />
    </d:line>
    <d:toolBar>
        <d:formButton action="" beforeAction="reset" title="新建" />
        <d:formButton id="save_bt" action="sys/function.execute!update" title="保存"/>
    </d:toolBar>
</d:form>
```

扩展对象函数：

```
function viewForm.setData(json);   //给 formPanel 设置数据，扩展了隐藏域
function viewForm.getData();   //得到 formPanel 的数据，包括隐藏域
function viewForm.reset();     //重置 formPanel
function viewForm.clear();     //清空 formPanel
function $D.setFieldReadOnly(filedId, status); //设置字段是否只读
```

属　性：id：声明一个 form 对象，全局唯一；
labelWidth：输入框说明文字宽度；
title：标题；
fileUpload：是否文件上传 form；
width：宽度，不设置时　自适应；
height：高度；

子标签：field 定义 form 对应的输入框。
属性：id：唯一标示；
name：对应后台取值名称；
prompt：对应文本描述；
width：对应宽度，px 为单位；
anchor：对应宽度百分比，当没有设置 width 时生效。
editor：对应编辑器（combo，textfield，numberfield，datefield，monthfield，datetimefield，htmleditor，textarea，password，file，checkbox，lov，lovcombo）；
lovUrl：lov 对应的页面。
lovWidth：打开窗口宽度。
lovHeight：打开窗口高度。
hidden：是否为隐藏列；
hideLabel：是否为隐藏标签；
upper：是否输入大写 true 为大写，false 为不限制；
required：是否必须；
emptyText：为空时显示内容；
readOnly：是否只读；
options：下拉列表对应的数据集，editor 为 combo 时起效。
displayField：显示的列，editor 为 combo 时起效。
valueField：实际值，editor 为 combo 时起效。
hiddenField：隐藏字段，对应 combo 的实际值。
checkedValue：复选框选中时值是多少，默认为 Y。
allowDecimals：是否允许为小数，对 numberfield 生效。
allowNegative：是否允许为负数，对 numberfield 生效。
vtype：校验类型 dbfound 内置五种类型。课根据 Ext 接口自定义。

1.alpha //只能输入字母，无法输入其他（如数字，特殊符号等）
2.alphanum//只能输入字母和数字，无法输入其他
3.email//email验证，要求的格式是"usc@sina.com"
4.url//url格式验证，要求的格式是http://www.sina.com
5.ip //ip 地址

子标签：formButton 定义 form 下面的控制按钮。

属性：id：唯一标示；
title：对应显示文本；
disabled：是否失效；
action：服务器响应 url；
icon：对应图标；
beforeAction：响应之前调用的 js 函数，return true 继续执行

```
function beforeAction(data,form,button){
    //data form中的数据
    //form 当前form对象
    //button 当前点击的button对象
}
```

afterAction：响应之后调用的 js 函数

```
function afterAction(res,data,form,button){
    //res服务器返回对象
    //record 选择的当前行记录
    //grid 当前grid对象
    //button 当前点击的button对象
}
```

4.3 树 tree

```
<d:dataSet id="treedata" modelName="test/tree"></d:dataSet>
<d:tree id="tree" title="树测试" bindTarget="treedata" idField="id" parentField="pid"
        displayField="text" width="300" height="600" />
```

扩展对象函数：

```
function tree.init();   //初始化树
function tree.refresh();   //刷新树
function tree.getCurrentNode();   //得到当前节点
function tree.getSelectedText();  //在有 checkbox 的情况下，得到选中节点描述。
function tree.getSelectedId();  //在有 checkbox 的情况下，得到选中节点 id。
```

Dataset：数据集，modelName 对应后台的 model 文件。
Tree：父子结构关系的对象，idField 和 parentField 进行关联。
Click：单据响应事件，对应 js 函数。

```
function click(node){
    // node 当前节点的信息
    // node.json 对应dataset的行记录
}
```

4.4 树表 treeGrid

treeGrid 继承 tree，对 grid 部分特性做了的相关实现。

```
<d:treeGrid id="treegrid" title="功能树表" idField="id" parentField="pid" showCheckBox="true" queryUrl="menu.query" height="450">
    <d:toolBar>
        <d:gridButton title="添加模块" beforeAction="add" />
    </d:toolBar>
    <d:columns>
        <d:column name="text" prompt="功能名称" width="150"></d:column>
        <d:column name="id" prompt="功能id" width="100"></d:column>
        <d:column name="url" prompt="对应jsp页面" width="200"></d:column>
        <d:column name="priority" prompt="优先级" width="100"></d:column>
    </d:columns>
</d:treeGrid>
```

扩展对象函数：

```
function tree.init();   //初始化树
function tree.refresh();   //刷新树
function tree.getCurrentNode();   //得到当前节点
function tree.getCurrentNodeData();   //得到当前节点数据，返回 json
function tree.setCurrentNodeData(json);   //设置当前节点数据
function tree.getSelectedText();  //在有 checkbox 的情况下，得到选中节点描述。
function tree.getSelectedId();  //在有 checkbox 的情况下，得到选中节点 id。
```

属　性：　id：声明一个 treegrid 对象，全局唯一；
autoQuery ：自动查询；
queryForm：对应过滤的查询表单；
queryUrl：查询对应的服务器地址，优先级高于 model 的查询地址；
viewForm：行展示 form，展示一行记录；
title：对应标题；
height：高度；
width：宽度，不设置时 屏幕自适应；
idField：tree 的 node id；
parentField: tree node 的父节点 id；
showCheckBox：是否显示选择框，默认 false；

4.5 组件 tabs

```
<d:tabs height="450" width="850">
    <d:tab title="角色定义">
        <d:grid id="roleGrid" model="sys/role" autoQuery="true" pagerSize="10">
            <d:columns>
                <d:column name="create_date" prompt="创建时间" width="120" />
                <d:column name="last_update_date" prompt="最后修改时间" width="120" />
                <d:column name="last_update_user" prompt="最后经手人" width="150" />
```

```
            </d:columns>
        </d:grid>
    </d:tab>
    <d:tab title="百度" url="http://www.baidu.com/"></d:tab>
</d:tabs>
```

使用方法如上例

```
function $D.getTab(tabId); //得到 tab 中 iframe 对象，用于访问 ifrme 的内容
```

4.6 组件 panel

```
<d:panel height="480" title="panel测试">
    <d:tree style="width:22%;position:absolute;" id="menuTree" >
    </d:tree>
    <d:grid style="width:76.5%;position:absolute;left:22.4%" id="roleGrid">
    </d:grid>
</d:panel>
```

Panel 为容器组件，可以在里面放在 form、grid、buttongroup、tree 等组件。

4.7 按钮 button

```
<d:buttonGroup>
    <d:button id="query" title="查询" icon="" click="query" />
    <d:button title="重置" click="reset" />
</d:buttonGroup>
```

使用方法如上例

4.8 菜单 menu

```
<d:menu id="menu">
    <d:menuItem title="添加子节点" icon="DBFoundUI/images/add.gif" click="say"></d:menuItem>
    <d:menuItem title="修改"></d:menuItem>
</d:menu>
```

```
function rootmenu(tree,e){
    menu.showAt([e.getPageX(),e.getPageY()]);
}
```

4.9 数据集 dataStore 与 dataSet

```
<d:dataSet id="roleStore" modelName="sys/role" name="combo" />
<d:dataStore id="useStore" fields="user_name,user_id" url="user.do" />
```

Dataset 和 datastore 都是数据存取的容器。

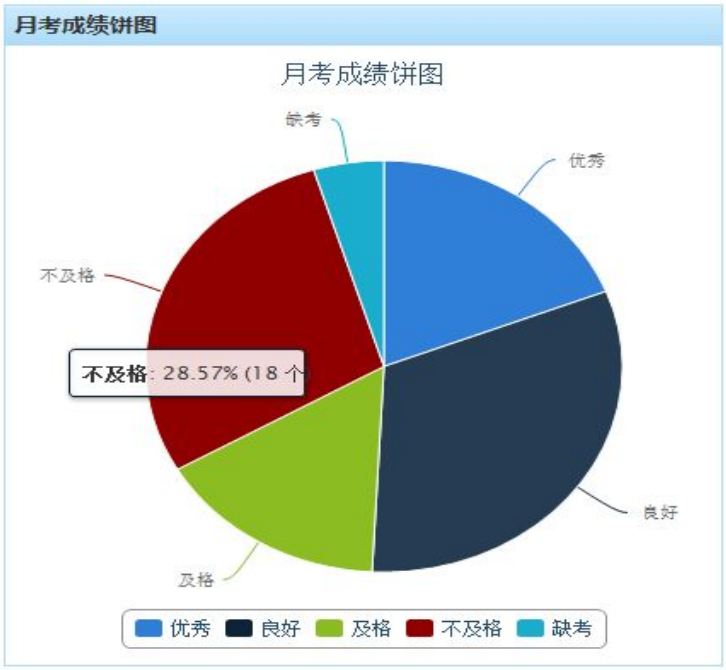
Dataset 能够直接绑定一个 model 文件，进行数据的异步与同步加载。

Datastore 绑定一个 url 连接进行异步赋值，`dataProvideClass` 实现同步赋值，指定一个 java 类进行赋值

`dataProvideClass` 必须实现 StoreDataProvide 接口

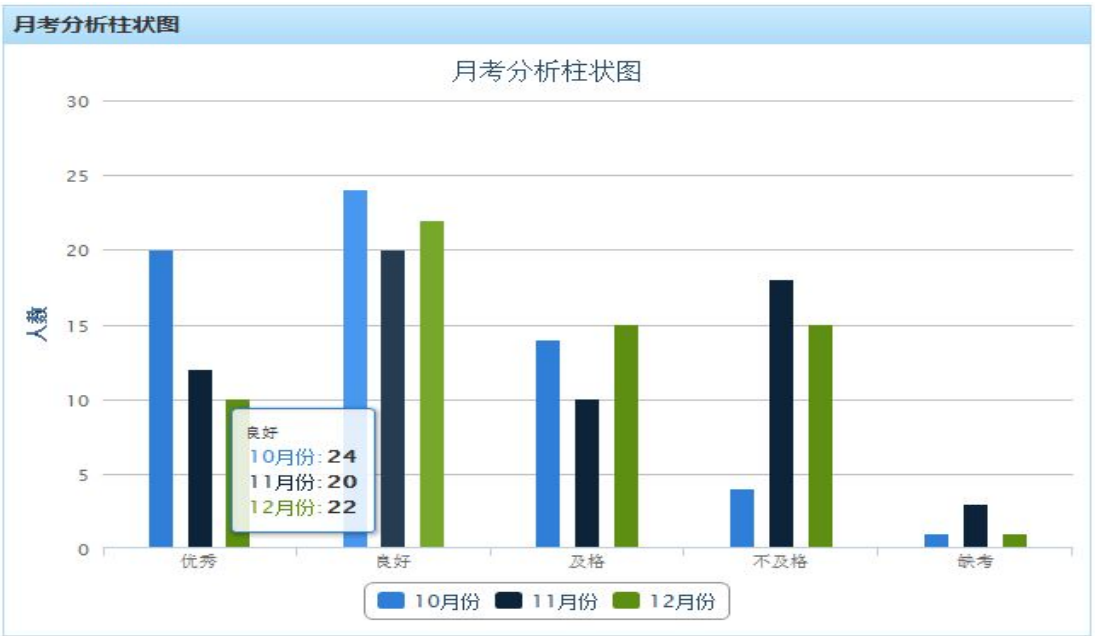
4.10 图像报表 chart

饼图：




```
<d:dataSet id="pieStore" modelName="test/chart" name="pie" />
<d:pieChart title="月考成绩饼图" dataLabel="true" bindTarget="pieStore" height="450"
valueField="totalnum" displayField="grade" />
```

柱状图：



```
<script type="text/javascript">
	var yAxis = [ {
	    name : '10月份',
	    field : 'total_num_10'
	} ,{
	    name : '11月份',
	    field : 'total_num_11'
	} ,{
	    name : '12月份',
	    field : 'total_num_12'
	}];
	var xAxis = {
	    name : '人数',
	    field : 'grade'
	};
</script>
<d:dataSet id="barStore" modelName="test/chart" name="bar" />
<d:barChart title="月考分析柱状图" yAxis="yAxis" xAxis="xAxis" bindTarget="barStore"
height="450" />
```

柱状图：

```
<d:lineChart title="月考分析折线图" yAxis="yAxis" xAxis="xAxis" bindTarget="barStore"
height="350" />
```

4.11 事件 event

目前事件支持在 field、column、grid、tree 组件中用标签定义事件。

例一、Lov 编辑器提交时触发 commit 事件。

```
<d:field name="user_name"  editor="lov" lovUrl="modules/sys/userLov.jsp"  >
    <d:event name="commit" handle="commit"></d:event>
</d:field>
```

例二、树的节点点击时触发

```
<d:tree bindTarget="treedata" idField="id"parentField="pid" >
      <d:event name="click" handle="sayHello"></d:event>
</d:tree>
```

同时我们可以用 js 对 UI 对象添加事件：

```
Ext.get("treedata").on("click",function(){});
```

对于子对象添加事件方式，我们只能用 js 处理，如 grid 的 store 的 load 事件；

```
grid.getStore().on("load",function(){});
```

具体对象对应的有那些事件，我们可以参照 extjs 官方文档。

4.12 DBFound 提示框

一般提示消息 `$D.showMessage(message,fn);`

警告提示消息 `$D.showWarning(message,fn);`

错误提示消息 `$D.showError(message,fn);`

确认提示消息 `$D.showConfirm(message,fn);`

参数 message 为消息，fn 为回调函数。

打开一个模态窗口：`$D.open(id, title, width, height, url, closeFunction);`

4.13 Ajax 数据交互

Dbfound 数据交互，直接支持 json 对象传递。及 param 可以是一个多层次的 json 对象。服务器接收到参数后，将数据转化为 list 的 map 形式。 服务器的数据存储见 第七章。

传递方式有下面两种

方式一：`$D.request(url,param,callback,mask,maskTitle);`

方式二：

```
$D.request({
        url : url,
        param : param,
        mask : mask,
        maskTitle : maskTitle,
        callback : callback
    });

    //url 为请求地址
    //param 为 json 格式参数
    //callback(resObj,response,action);回调函数 resObj 为服务器反馈对象
    //mask 是否需要 mask 锁屏，默认为 false
    //maskTitle 锁屏标题
```

4.14 国际化 i18n

dbfound 提供了国际化标签接口。
在 dbfound 标签内的多语言：采用 *i18n:user.define* 方式获取，如

```
<d:form id="queryForm" title="i18n:user.define" width="750" labelWidth="80">
```

其它区域的多语言的实现方式如下：<d:text value="user.define"/>方式获取，如

```
var a ='<d:text value="user.define"/>';
<div><d:text value="user.define"/></div>
```

4.15 dbfound 布局 style

dbfound 每个组件都提供了 style 属性，来实现布局效果。同时 dbfound ui 组件可以和 div 混用，对应复杂的可以先用 div 布局好，然后填充。

一、style 绝对路径布局。如下代码我们让树宽度为屏幕的 49.9%，左边距为 50%，上边距为 0 像素。

```
<d:tree style="width:49.9%;position:absolute;left:50%;top:0px" id="menuTree">
</d:tree>
```

二、div 绝对路径布局。这样达到和上面同样的效果，一般情况下我们的宽度不要达到 100%，留一点点缓冲区对应不同浏览器的兼容性好一些。

```
<div style="width:49.9%;position:absolute;left:50%;top:0px">
    <d:tree id="menuTree" >
    </d:tree>
</div>
```

三、混合布局。我们先用两个 div 先屏幕划分为左右两个区域。然后分别添入两个 grid。由于组件默认情况下左右边距都为 margin 5px，这样导致中间有了 10px 的间距，所以我们在右边的 grid 设置 marin-left 0px，调整边距。

```
<div class="dbfound-td" style="width:50%;position:absolute;">
    <d:grid id="userGrid0"
    </d:grid>
    <d:grid id="userGrid1" >
    </d:grid>
</div>
<div style="width: 49.9%;position:absolute;left:50%">
    <d:grid id="userGrid2" style="margin-left:0px" >
    </d:grid>
    <d:grid id="userGrid3" style="margin-left:0px" >
    </d:grid>
</div>
```

效果如下：


用户列表
新增
保存
用户编号
用户名
密码
创建日期
最后经手人
1
ADMIN
系统管理员
123456
2013-03-...
ADMIN-系...
第 1 页,共 1 页
每页显示 10 条 本页显示第

五、文件上传与下载

5.1、DBFound 文件上传

1、首先定义一个 form，代码如下

```
<d:form id="uploadform" width="400" labelWidth="70" fileUpload="true" >
      <d:line columnWidth="1">
          <d:field name="file" id="file_cmp" required="true" prompt="附 件" editor="file"/>
      </d:line>
</d:form>
```

2、调用下面 JavaScript 进行 form 提交，代码如下

```
uploadform.form.submit({
            url:'upload.execute!add?pk_value=${param.pk_value}
                &table_name=${param.table_name}',
            method:'post'
});
```

3 、服务器端 model 写法，代码如下

```
<execute name="add">
    <param name="file id" UUID="true" />
    <param name="file_name" dataType="varchar" />
    <param name="file type" dataType="varchar" />
    <param name="file size" dataType="number" />
    <param name="file" dataType="file" />
    <sqls>
        <executeSql>
         <![CDATA[
           insert into sys_upload_file
                (file id,
                 file_name,
                 file_content,
                 file type,
                 file_size,
                 table_name,
                 pk value)
           values (${@file_id},
                 ${@file_name},
                 ${@file},
                 ${@file_type},
                 ${@file_size},
                 ${@table name},
                 ${@pk_value})
         ]]>
        </executeSql>
    </sqls>
</execute>
```

这样就轻松的实现了 将文件上传到数据库。同时 dbfound 支持上传文档到文件夹。只需要设置 param 的 fileSaveType 属性（db 保存到数据库、disk 保存到文件夹，默认 db）。

_name, _size , _type 根据上传的文件的 param，dbfound 自动分析得到。如果上传的 param 的 name 为 file。所以我们就可以根据 file_name, file_type, file_size 得到对应属性。

当 file 保存到文件夹时参数 file 的 value 为对应硬盘上保存的文件名称。

5.2 、DBFound 文件下载

dbfound 文件下载支持从数据库下载 和 从文件系统下载两种方式。
根据 fileSaveType 区分，如果是 db 就从数据库取，如果是 disk 就从文件系统取。默认为 db。
下面为从数据库取文件的 execute 例子

```
<execute>
    <param name="file_id" dataType="varchar" />
    <param name="file_name" dataType="varchar" />
    <param name="content" dataType="file" ioType="out" fileNameParam="file_name" />
    <sqls>
        <querySql>
         <![CDATA[
           SELECT f.file name , f.file content content
             FROM sams.sys upload file f
            WHERE f.file_id = ${@file_id}
         ]]>
        </querySql>
    </sqls>
</execute>
```

下面为从文件系统取的例子

```
<execute name="diskdown">
    <param name="file_name" dataType="varchar" />
    <param name="content" dataType="file" ioType="out" fileSaveType="disk" fileNameParam="file_name" />
    <sqls>
        <querySql>
         <![CDATA[
            select '应用服务器安装报告.doc' file name,
                   'AQEYDSHID.doc' content
            from  dual
         ]]>
        </querySql>
    </sqls>
</execute>
```

参数 content 说明文件夹下 这个文件叫什么名字，参数 file_name 说明下载文件实际的名字。通过设置 `fileNameParam` 属性来实现绑定。

如果在知道文件名字的前提下 dbdound 提供了更简单的下载方式，代码如下

```
<execute name="download">
    <param name="file name" dataType="varchar" value="应用服务器安装报告.doc" />
    <param name="content" dataType="file" ioType="out" fileSaveType="disk" value="AQEYDSHID.doc"
fileNameParam="file name" />
</execute>
```

这样我们就实现了轻松下载。我们根据参数 content 的 value 可以就到对应的文件，然后根据 `fileNameParam` 找到最终的文件名称，从而进行下载。

写好了 model 文件，我们就可以反问 url 进行下载了
通过访问：http://localhost:8080/dbfound/download.execute!download
我们通过名为 download 的 execute 就得到了*应用服务器安装报告.doc* 这个文件。

六、excel 导入与导出

6.1、excel 导入

A、定义 form：

```
<d:form id="uploadform" width="400" labelWidth="70" fileUpload="true" >
    <d:line columnWidth="1">
        <d:field name="file" width="280" required="true" prompt="excel文件" editor="file"/>
    </d:line>
</d:form>
```

B、提交 js 函数与文件上传一致。

```
uploadform.form.submit({
            url:'upload.execute!batch_add',
            method:'post'
});
```

C、后台 model 写法： excelReader 将 excel 内容转化为 list<map>

```
<execute name="batch add">
  <param name="发货单号" dataType="varchar" />
  <param name="单据日期" dataType="varchar" />
  <param name="收发类别" dataType="number" />
  <param name="客户编码" dataType="varchar" />
  <param name="部门编码" dataType="varchar" />
  <param name="file name" dataType="varchar" />
  <param name="file" dataType="file" />
  <sqls>
     <excelReader sourceParam="file" rootPath="exceldatas"/>
     <batchSql sourcePath="exceldatas[0]">
          <executeSql>
           <![CDATA[
             insert into shipments document(
                         shipments_code,
                         document time,
                         take type ,
                         customer_code,
                         unit code)
                 values(${@发货单号},
                        ${@单据日期},
                        ${@收发类别},
                        ${@客户编码},
                        ${@部门编码})
           ]]>
          </executeSql>
      </batchSql>
  </sqls>
</execute>
```

6.2、excel 导出

dbfound excel 导出，只需要在 grid 定义 gridButton 声明 type = ‘excel’；

```
<d:grid id="itemGrid"  queryUrl="report/buyItems.query!count query" >
    <d:toolBar>
         <d:gridButton type="excel" />
    </d:toolBar>
    <d:columns>
        <d:column name="item_code" width="110" prompt="产品编号" />
        <d:column name="item_name" width="360" prompt="产品名称" />
        <d:column name="purchase_number" prompt="数量" width="80" />
        <d:column name="sigle_price" prompt="含税单价" width="100" />
        <d:column name="total_price" width="100" prompt="含税金额" />
    </d:columns>
</d:grid>
```

七、数据交互与存储

dbfound 前后台以 json 格式数据进行交互。前台传过来的数据，后台转化为一个 List<Map>的数据树结构。

dbfound 的数据存储 分为四大区域

1、param （对应前台传过来的参数）。
2、request （对应 http request 中设置的属性）。
3、session （对应 http session 中设置的属性）。
4、outParam （对应 dbfound Model 中的 out 参数，将其返回到客户端）。

案例说明：

一、在 jsp 页面中我们用$D.request 来请求服务器地址。传递了 user 和 role 两个对象到服务器端。

```
function test() {
    var param = {};
    param.user = {code:"10000",name:"黄炯"};
    param.role = {code:"ADMIN",name:"管理员"}
    $D.request("sys/user.do!test",param,function(resObj){
        $D.showMessage(resObj.message);
    });
}
```

二、在服务器端，我们在 thread 中可以得到传输过来的数据。

```
public ResponseObject test(Context context) throws Exception {
        LogUtil.info("path /: "+context.getDatas());
        LogUtil.info("path /param: "+context.getData("param"));
        LogUtil.info("path /param/user: "+context.getData("param.user"));
        LogUtil.info("path /param/user/name: "+context.getData("param.user.name"));

        ResponseObject ro = new ResponseObject();
        ro.setMessage("测试成功啦!! ");
        ro.setSuccess(true);
        return ro;
}
```

打印出来的数据：

```
2013-06-02 16:16:41,270 INFO [dbfound] - path /:
{session={role_id=3, user_code=admin, user_id=2},
 param={role={name=管理员, code=ADMIN}, user={name=黄炯, code=10000}}}

2013-06-02 16:16:41,270 INFO [dbfound] - path /param:
{role={name=管理员, code=ADMIN}, user={name=黄炯, code=10000}}

2013-06-02 16:16:41,270 INFO [dbfound] - path /param/user:
{name=黄炯, code=10000}

2013-06-02 16:16:41,270 INFO [dbfound] - path /param/user/name:
 黄炯
```

三、在服务器端 new 一个 ResponseObject 对象返回到客户端。客户端接收到服务器端的反馈消息。