1.使用Java来动态构建Dorado数据集与组件（Java版本DoradoHelloworld）(T31)

下载

使用Java来动态构建Dorado数据集与组件（Java版本DoradoHelloworld）.doc
Java版本DoradoHelloworld 工程文件.zip

目录

前言

大家已经对Dorado的Helloworld示例非常熟悉了。亲眼见证通过View的简单配置就可以开发出功能完整的页面是一件令人兴奋的事。如果再关注一下Dorado的服务器端编程你会发现另外一片天空。本文是对《dorado5 用户指南》第六章《ViewModel(视图模型)》的代码实践。
由于Dorado在客户端的表现十分出色，所以一些使用者喜欢在那里做更多的事情，操作业务上下文与流转的JavaScript增多，导致客户端效率下降并且代码难以维护。我们建议客户端仅仅保留由于业务定义要求的对Dataset和Control的约束，而将关键的上下文判断与业务计算放在服务器端，甚至在服务器端判断出上下文后可以仅仅初始化与该业务相关的Dataset和Control，降低系统开销。

ViewModel介绍

Dorado的服务器端编程主要集中在ViewModel的实现类中，即使我们没有定义过，也会存在一个默认的实现，在setting.xml中可以看到：

```
<property name="view.defaultViewModel"
 value="com.bstek.dorado.view.impl.DynaViewModel"/>
```

您可以通过继承上面的ViewModel，定义新的方法并覆盖此处配置，达到扩展ViewModel的效果。
首先明确一下我们编写ViewModel实现类的目的。由于ViewModel是对一个业务单元的抽象，管理着与该业务相关的数据集和组件，所以它的实现类的工作重点也是如此。下面是与此相关的一些重要API。

ViewModel的重要方法

void init(int state)

ViewModel的初始化方法。初始化的依据是View的配置文件与参数state，即 "ViewModel的状态"。
View的配置文件是静态的，而state是动态的，根据不同的状态，ViewModel完成不同的初始化动作。

createDataset(type,id)

创建一个ViewDataset，并注册给ViewModel。如果参数id与ViewModel已经存在的某个Dataset的id相同，则创建动作失败并有异常抛出。事实上需要我们手动创建一个Dataset的机会并不多。

initDataset(viewdataset)

ViewModel初始化一个ViewDataset的方法。该方法在createDataset中被调用，是初始化ViewDataset的回调函数，默认的实现体为空，我们可以在这里做动态的初始化工作。

initDatasets()

ViewModel初始化所有已配置ViewDataset的回调函数。默认的实现体为空，我们可以在这里动态初始化多个ViewDataset，甚至添加新的ViewDataset。

createControl(type,id)

创建一个Control，并注册给ViewModel。同样id不能冲突。

initControl(control)

ViewModel初始化一个Control的方法。该方法在createControl中被调用，是初始化Control的回调函数，默认的实现体为空，我们可以在这里做动态的初始化工作。

initControls()

ViewModel初始化所有已配置的不可见Control的回调函数。默认的实现体为空，我们可以在这里动态初始化多个Control，甚至添加新的Control。

doLoadData()

ViewModel为其中的若干个Dataset加载数据时的方法。

doLoadData(viewdataset)

ViewModel为指定的Dataset加载数据时的方法。

doUpdateData(parameters, outParameters)

默认的处理数据提交的方法。简单执行每个被提交Dataset的update()方法。

ViewModel的状态

在Web应用当中，每次用户刷新或请求新的页面时都会产生客户端向服务器端发送请求的操作，Dorado的Web应用也是如此。此外Dorado应用集成了AJAX功能，所以Dorado的客户端会向服务器端发送两种请求：一、页面请求；二、AJAX请求。然而Dorado是一个面向业务的企业级应用框架，根据业务操作又将AJAX请求分为三种：

1. 简单的AJAX请求。客户端向服务器端传递一组参数，并调用一个远程方法，服务器端将执行结果返回给客户端。
2. 加载数据的请求。客户端向服务器端传递一组参数，服务器端根据这些参数返回一组数据给客户端。
3. 提交数据的请求。客户端向服务器端提交一组数据，服务器端加工处理后再将这组数据返回给客户端。

这三种请求分别对应Dorado的三种组件：RPCCommand、QueryCommand、UpdateCommand。因为在服务器端为这些请求提供服务的都是ViewModel，所以ViewModel必须具备区分这些请求的能力。区分结果被转换成了ViewModel的一个属性——"状态"，也就是说我们可以根据不同状态判断出ViewModel正在处理的是客户端的哪种请求。我们接下来会细致的看一下ViewModel在不同状态下是如何处理每一种请求的。

STATE_VIEW

视图状态，当浏览器打开Dorado的JSP时其中ViewModel的状态，发生在用户刷新或浏览Dorado页面时，这时ViewModel会进行一系列动作，最终将生成标准的HTML输出给客户端。那么这些HTML包括哪些内容呢？

1. ViewModel的信息。定义ViewModel的JavaScript。
2. 数据模型的信息。包括定义Dataset和其中数据的JavaScript和XML。
3. 组件和布局的信息。包括定义Control和布局的HTML以及JavaScript。

在输出这些内容之前，ViewModel必须在服务器端做好充分的准备。首先调用init(state)方法，将View配置文件中的信息同步到ViewModel中。包括

1. 自身属性：role、safe、skin等，还有Properties。
2. Dataset信息：
   调用createDataset方法根据View配置文件中Dataset节点初始化Dataset，并注册给ViewModel，然后调用initDataset(viewdataset)方法。通常我们不需要关注createDataset方法，而将对Dataset动态初始化的工作放到initDataset(viewdataset)中。由于View配置文件中可能有多个Dataset节点，所以上面的方法会被多次调用。所以initDataset(viewdataset)的代码中需要判断viewdataset的id属性。当所有ViewDataset初始化结束后执行initDatasets()方法。
3. Control信息：为了提高服务器端效率Dorado提供了叫做"组件懒加载"的机制，即只初始化那些在JSP标签中出现的可见组件，如果一个被定义的可见组件的标签没有在JSP中出现过，则该组件不会被初始化也不会被输出到客户端。然而所有的不可见组件是不存在懒加载的，因为无法判断是否被使用了，所以不可见组件都会被初始化。这样就导致了这两种组件被初始化的时间点是不同的，可见组件是在解析标签时初始化的，不可见组件是在初始化完所有的ViewDataset后被初始化的。

然后进行加载数据的动作：执行doLoadData()方法，所有autoLoadData属性不是false的ViewDataset都会去加载数据。为每个ViewDataset加载数据的方法是doLoadData(viewdataset)，其中会调用viewdataset的load()方法。下图是上文提到的方法调用顺序。

init(state)	state==STATE_VIEW
createDataset	创建ViewDataset，可能执行多次
initDataset(viewdataset)	初始化viewdatset的回调函数
initDatasets()	初始化全部viewdatset的回调函数
createControl	创建不可见Control，可能执行多次
initControl(control)	初始化不可见control的回调函数
initControls()	初始化全部不可见control的回调函数
doLoadData()	加载数据，仅执行一次

doLoadData(viewdataset)	ViewDataset加载数据，可能执行多次
createControl	创建可见Control，可能执行多次
initControl(control)	初始化可见Control的回调函数

图1.2.1

STATE_SERVICE

服务状态，当一个已打开视图执行远程方法调用或局部数据刷新时的状态。通常是由于客户端执行了下面的代码：

1. Dataset.flushData(); 显性执行局部数据刷新。
2. QueryCommand.execute(); 隐式执行局部数据刷新。
3. DataPilot或PagePilot引起的翻页动作。隐式执行局部数据刷新。
4. ExcelCommand.execute();并且dataMode=server-all或server-current-page，导出记录。
5. RpcCommand.execute(); 显性执行远程方法调用。

前四种情况的本质都是客户端向服务器端发送加载数据的AJAX请求，服务器端的ViewModel具有相同的动作。这时ViewModel的动作与STATE_VIEW时的相似，但更简单。
第一步、初始化工作。调用init(state)方法，将View配置文件中的信息同步到ViewModel中。包括：

1. 自身属性：role、safe、skin等，还有Properties。
2. Dataset信息：仅执行一次createDataset和initDataset方法 。

第二步、加载数据： 仅执行一次doLoadData(viewdataset)方法。
方法的调用顺序如下表：

init(state)	state==STATE_SERVICE
initDatasets()	初始化全部viewdatset的回调函数
createDataset	创建viewdataset，仅一次
initDataset(viewdataset)	初始化viewdatset的回调函数
doLoadData()	
doLoadData(viewdataset)	viewdataset加载数据，仅一次

图1.2.2
第五种情况，即简单的RPC请求，是一种轻量级的AJAX请求，服务器端的ViewModel的初始化工作最少，因为这时不需要初始化任何的Dataset和Control，所以ViewModel初始化自身的简单属性后就可以执行被请求的方法了。

init(state)	state=STATE_SERVICE
simpleRpc(parameters, outParameters)	执行被请求的方法

STATE_UPDATING

提交状态。ViewModel处理客户端提交的数据，通常是由于客户端执行了UpdateCommand.execute()语句。
因为一次性可以提交多个Dataset，所以服务器端的ViewModel需要初始化这些ViewDataset并填充客户端提交过来的数据，仍然使用createDataset方法进行创建，initDataset方法也同样会被调用。然后执行处理提交请求的方法，默认为doUpdateData方法。方法的调用顺序如下表：

init(state)	state=STATE_UPDATING
initDatasets()	
createDataset	创建客户端提交的viewdataset，可能多次
initDataset(viewdataset)	初始化viewdataset的回调方法

doUpdateData ()	默认的处理提交请求的方法

STATE_REPORT

报表状态。ViewModel为Dorado报表提供服务。

STATE_DESIGN

设计时状态。ViewModel处理Studio或Eclipse Plugin请求时，例如：Dataset自动生成字段。用户不需要关注。

Java版Helloworld

截图

功能点

建立view标签与ViewModel的联系

更多的时候我们会利用View配置文件间接建立view标签与ViewModel的联系，方法是在jsp中的view标签中定义config属性，例如：

```
<d:View config="view.employee.EmployeeConfig">
```

再利用view配置文件的clazz属性，例如：

```
<view clazz="view.employee.EmployeeConfigViewModel">
```

在本示例中我们使用一种直接的方式，直接配置view标签的clazz属性，例如：

```
<d:View clazz="view.employee.EmployeeViewModel">
```

创建datasetEmployee

如果使用view配置文件，定义一个Dataset是很轻松的事情，简单的配置不需要编写任何代码，Dorado已经为我们处理了所有的繁琐和细节问题，那么如果我们想用代码的方式做这件事情，就需要承担这些工作。下面是在ViewModel中创建datasetEmployee的相关方法：

```
/**
     * 创建View版本的datasetEmployee
     *
     * @return
     * @throws Exception
     */
    private ViewDataset creatDatasetEmployee() throws Exception {
        AutoSqlDataset innerDatasetEmployee = getInnerDatasetEmployee();
        FieldSet fieldset = innerDatasetEmployee.fieldSet();
        Field field;
        ViewField viewField;
        ViewDataset datasetEmployee = this.createDataset("custom",
                "datasetEmployee");
        datasetEmployee.setPageSize(10);
        for (int i = 0, count = fieldset.getCount(); i < count; i++) {
            field = fieldset.getField(info);
            viewField = (ViewField) datasetEmployee.addField(field.getName());
            viewField.setDataType(field.getDataType());
            viewField.setLabel(field.getLabel());
        }
        viewField = (ViewField) datasetEmployee.getField("EMPLOYEE_ID");
        RequiredValidator validator = new RequiredValidator();
        validator.setErrorMessage("请输入[员工编号]信息");
        viewField.addValidator(validator);
        if (this.getState() == ViewModel.STATE_VIEW) {
            viewField = (ViewField) datasetEmployee.getField("SEX");
            if (viewField != null) {
                viewField.setDropDown("dropdownSex");
            }
            viewField = (ViewField) datasetEmployee.getField("DEPT_ID");
            if (viewField != null) {
                viewField.setDropDown("dropdownDept");
            }
            viewField = (ViewField) datasetEmployee.getField("BIRTHDAY");
            if (viewField != null) {
                viewField.setFormat("yyyy年MM月dd日");
            }
            EventHandler eventHandler = new EventHandler("beforeFlushData");
            eventHandler.setScript("var ps = dataset.parameters();"
                    + "var id = ps.getValue('id');"
                    + "if(id && id.charAt(0)!='%')"
                    + "{ps.setValue('id','%'id.toUpperCase()'%');};"
                    + "var name = ps.getValue('name');"
                    + "if(name && name.charAt(0)!='%')"
                    + "{ps.setValue('name','%'name'%');}");
            datasetEmployee.addEventHandler(eventHandler);
        }
```

```
    return datasetEmployee;
}
/**
 * 创建AutoSql版本的datasetEmployee
 *
 * @return
 */
private AutoSqlDataset getInnerDatasetEmployee() {
    if (innerDatasetEmployee != null) {
        return innerDatasetEmployee;
    }
    innerDatasetEmployee = new AutoSqlDataset();
    innerDatasetEmployee.setId("innerDatasetEmployee");
    innerDatasetEmployee.setDataSource("doradosample");
    innerDatasetEmployee.setOriginTable("EMPLOYEE");
    innerDatasetEmployee.setKeyFields("EMPLOYEE_ID");
    AutoDBField f = (AutoDBField) innerDatasetEmployee
            .addField("EMPLOYEE_ID");
    f.setOriginField("EMPLOYEE_ID");
    f.setDataType(DataType.STRING);
    f.setLabel("员工编号");
    f = (AutoDBField) innerDatasetEmployee.addField("EMPLOYEE_NAME");
    f.setOriginField("EMPLOYEE_NAME");
    f.setDataType(DataType.STRING);
    f.setLabel("员工姓名");
    f = (AutoDBField) innerDatasetEmployee.addField("SEX");
    f.setOriginField("SEX");
    f.setDataType(DataType.BOOLEAN);
    f.setLabel("性别");
    f = (AutoDBField) innerDatasetEmployee.addField("DEPT_ID");
    f.setOriginField("DEPT_ID");
    f.setDataType(DataType.STRING);
    f.setLabel("部门");
    f = (AutoDBField) innerDatasetEmployee.addField("BIRTHDAY");
    f.setOriginField("BIRTHDAY");
    f.setDataType(DataType.DATE);
    f.setSqlDataType(java.sql.Types.DATE);
    f.setLabel("出生日期");
    f = (AutoDBField) innerDatasetEmployee.addField("SALARY");
    f.setOriginField("SALARY");
    f.setDataType(DataType.DOUBLE);
    f.setSqlDataType(java.sql.Types.DOUBLE);
    f.setLabel("薪水");
    f = (AutoDBField) innerDatasetEmployee.addField("DEGREE");
    f.setOriginField("DEGREE");
    f.setDataType(DataType.STRING);
    f.setLabel("学历");
    innerDatasetEmployee.addBaseMatchRule("EMPLOYEE_ID", "like", ":id")
            .setEscapeEnabled(true);
    innerDatasetEmployee.addBaseMatchRule("EMPLOYEE_NAME", "like", ":name")
            .setEscapeEnabled(true);
    innerDatasetEmployee.addBaseMatchRule("SEX", "=", ":sex")
            .setEscapeEnabled(true);
    innerDatasetEmployee.addBaseMatchRule("DEPT_ID", "=", ":dept")
```

```
            .setEscapeEnabled(true);
        return innerDatasetEmployee;
    }
```

在上面的private AutoSqlDataset getInnerDatasetEmployee()方法中我们看到了如何创建AutoSqlDataset实例，以及设置属性和添加字段与匹配规则的代码。注意这里创建的AutoSqlDataset并没有注册到ViewModel中。
我们将更多的精力放到private ViewDataset creatDatasetEmployee()方法上。这里我们展示了如果使用createDataset方法创建并注册CustomDataset的代码：

```
ViewDataset datasetEmployee = this.createDataset("custom",  "datasetEmployee");
```

如何为ViewDataset添加字段：

```
viewField = (ViewField) datasetEmployee.addField(field.getName());
```

如何为ViewField添加验证器：

```
viewField = (ViewField) datasetEmployee.getField("EMPLOYEE_ID");
RequiredValidator validator = new RequiredValidator();
validator.setErrorMessage("请输入[员工编号]信息");
viewField.addValidator(validator);
```

如何为ViewField设置下拉框：

```
viewField = (ViewField) datasetEmployee.getField("SEX");
if (viewField != null) {
 viewField.setDropDown("dropdownSex");
}
```

如何为ViewDataset添加事件：

```
EventHandler eventHandler = new EventHandler("beforeFlushData");
 eventHandler.setScript(
 "var ps = dataset.parameters();"
 + "var id = ps.getValue('id');"
 + "if(id && id.charAt(0)!='%')"
 + "{ps.setValue('id','%'id.toUpperCase()'%');};"
 + "var name = ps.getValue('name');"
 + "if(name && name.charAt(0)!='%')"
 + "{ps.setValue('name','%'name'%');}");
 datasetEmployee.addEventHandler(eventHandler);
```

因为Field的dropDown属性和Dataset的EventHandler对于客户端来说是有意义的，对于服务器端来说没有任何意义，所以在设置这些属性的前边我们增加了对ViewModel状态的判断：

```
if (this.getState() == ViewModel.STATE_VIEW)
```

当然这属于精益求精的方式，即使没有这个判断也不会对服务器端造成不良影响。
还有一件需要澄清的问题，我们要在哪里调用private ViewDataset creatDatasetEmployee()？答案是在initDatasets()方法里。代码如下：

```
@Override
    protected void initDatasets() throws Exception {
        switch (this.getState()) {
        case ViewModel.STATE_VIEW:
            createDatasetQuery();
            creatDatasetEmployee();
            break;
        case ViewModel.STATE_SERVICE:
            creatDatasetEmployee();
            break;
        case ViewModel.STATE_UPDATING:
            creatDatasetEmployee();
            break;
        default:
            break;
        }
    }
```

显然我们在三种状态中都创建了datasetEmployee。你可以尝试把STATE_UPDATING状态下的creatDatasetEmployee()语句屏蔽掉，你会发现程序仍然可以运行，奇怪吗？

创建datasetQuery

为了提供查询功能我们需要创建datasetQuery，代码如下：

```
private ViewDataset createDatasetQuery() throws Exception {
        ViewDataset datasetQuery = this.createDataset("form", "datasetQuery");
        ViewField field;
        field = (ViewField) datasetQuery.addField("ID");
        field.setLabel("员工编号");
        field = (ViewField) datasetQuery.addField("NAME");
        field.setLabel("员工姓名");
        field = (ViewField) datasetQuery.addField("SEX");
        field.setLabel("性别");
        field.setDropDown("dropdownSex");
        field = (ViewField) datasetQuery.addField("DEPT");
        field.setLabel("部门");
        field.setDropDown("dropdownDept");
        return datasetQuery;
    }
```

这里的代码要简单的多，并且只有在STATE_VIEW状态时datasetQuery才有意义。

创建formQuery

创建formQuery的代码如下：

```
private AutoForm createFormQuery() throws Exception {
        AutoForm formQuery = (AutoForm) this.createControl("AutoForm",
                "formQuery");
        formQuery.setDataset("datasetQuery");
        FormGroup group = formQuery.addGroup("group0");
        group.setLabelWidth("60");
        group.setTitle("员工查询");
        FieldSet fieldset = formQuery.getDatasetInstance().fieldSet();
        String fieldname;
        for (int i = 0, count = fieldset.getCount(); i < count; i++) {
            fieldname = fieldset.getField(info).getName();
            group.addElement("TextEditor", fieldname).setField(fieldname);
        }
        return formQuery;
    }
```

上面的展示了如何在AutoForm中添加Group，以及在Group中添加Element的代码，在这里我们能第一次使用到了createControl方法。

创建buttonQuery和queryCommandEmployee

因为buttonQuery与queryCommandEmployee是相互依存的，所以他们的创建动作被封装在一个方法中，代码如下：

```
private Button createButtonQuery() throws Exception {
        QueryCommand queryCommandEmployee = (QueryCommand) this.createControl(
                "QueryCommand", "queryCommandEmployee");
        queryCommandEmployee.setConditionDataset("datasetQuery");
        queryCommandEmployee.setQueryDataset("datasetEmployee");
        Button buttonQuery = (Button) this.createControl("Button",
                "buttonQuery");
        buttonQuery.setValue("查询");
        buttonQuery.setWidth("70");
        buttonQuery.setCommand("queryCommandEmployee");
        return buttonQuery;
    }
```

创建buttonSave和commandSave

由于相同的原因创建buttonSave与commandSave的动作封装在一个方法中了，代码如下：

```
  private Button createButtonSave() throws Exception {
        Button buttonSave = (Button) this.createControl("Button", "buttonSave");
        buttonSave.setCommand("commandSave");
        buttonSave.setValue("保存");
        buttonSave.setWidth("70");
        UpdateCommand commandSave = (UpdateCommand) this.createControl(
                "UpdateCommand", "commandSave");
        commandSave.setMethod("doSaveEmployee");
        DatasetInfo info = commandSave.addDatasetInfo("datasetEmployee");
        info.setSubmitScope(DatasetInfo.ALL_CHANGE);
        return buttonSave;
    }
```

上面展示了如何向UpdateCommand中添加Dataset和设置提交范围的代码。

创建dropdownSex

为了将datasetEmployee的sex字段翻译成"男"或"女"，我们需要创建dropdownSex。代码如下：

```
private DropDown createDropdownSex() throws Exception {
    ListDropDown dropdownSex = (ListDropDown) this.createControl(
            "ListDropDown", "dropdownSex");
    dropdownSex.setMapValue(true);
    dropdownSex.setFixed(true);
    dropdownSex.setAutoDropDown(true);
    dropdownSex.addItem("", "<空>");
    dropdownSex.addItem("true", "男");
    dropdownSex.addItem("false", "女");
    return dropdownSex;
 }
```

是不是很简单，还记得如何将dropdownSex绑定到sex字段上吗？

创建dropdownDept

为了将datasetEmployee的dept_id字段翻译成部门名称我们需要创建dropdownDept，代码如下：

```
private DropDown createDropdownDept() throws Exception {
    DatasetDropDown dropdownDept = (DatasetDropDown) this.createControl(
            "DatasetDropDown", "dropdownDept");
    dropdownDept.setAutoDropDown(true);
    dropdownDept.setDataset("datasetDept");
    dropdownDept.setFixed(true);
    dropdownDept.setMapValue(true);
    dropdownDept.setValueField("DEPT_ID");
    dropdownDept.setLabelField("DEPT_NAME");
    dropdownDept.setStartWithEmptyRecord(true);
    AutoSqlDataset autoDatasetDept = new AutoSqlDataset();
    autoDatasetDept.setOriginTable("DEPT");
    AutoDBField f = (AutoDBField) autoDatasetDept.addField("DEPT_ID");
    f.setOriginField("DEPT_ID");
    f = (AutoDBField) autoDatasetDept.addField("DEPT_NAME");
    f.setOriginField("DEPT_NAME");
    autoDatasetDept.load();
    ViewDataset datasetDept = this.createDataset("custom", "datasetDept");
    datasetDept.addField("DEPT_ID");
    datasetDept.addField("DEPT_NAME");
    datasetDept.setRecordSet(autoDatasetDept.recordSet());
    return dropdownDept;
 }
```

因为datasetDept与dropdownDept是相互依存的，只有在两者同时存在时才有意义，所以将二者的创建动作封装到了一个方法中，中间还使用了一个临时的AutoSqlDataset。

创建tableEmployee

为了展示datasetEmployee中的数据，我们需要创建tableEmployee，代码如下：

```
private DataTable createTableEmployee() throws Exception {
        DataTable tableEmployee = (DataTable) this.createControl("DataTable",
                "tableEmployee");
        tableEmployee.setDataset("datasetEmployee");
        tableEmployee.setHeight("100%");
        tableEmployee.setWidth("100%");
        tableEmployee.setShowHScrollBar(false);
        EditableColumn column;
        ViewField viewfield;
        String fieldname;
        ViewDataset datasetEmployee = this.getDataset("datasetEmployee");
        FieldSet fieldset = datasetEmployee.fieldSet();
        for (int i = 0, count = fieldset.getCount(); i < count; i++) {
            viewfield = (ViewField) fieldset.getField(info);
            fieldname = viewfield.getName();
            column = (EditableColumn) tableEmployee.addColumn(fieldname);
            column.setField(fieldname);
            column.setLabel(viewfield.getLabel());
        }
        column = (EditableColumn) tableEmployee.getColumn("EMPLOYEE_ID");
        if (column != null) {
            column.setAlign("left");
            column.setWidth(110);
        }
        column = (EditableColumn) tableEmployee.getColumn("EMPLOYEE_NAME");
        if (column != null) {
            column.setAlign("left");
            column.setWidth(60);
        }
        column = (EditableColumn) tableEmployee.getColumn("SEX");
        if (column != null) {
            column.setAlign("center");
            column.setWidth(30);
        }
        return tableEmployee;
    }
```

上面的代码展示了如何向DataTable中添加Column，以及如果定义Column的属性。

附录

Dataset VS ViewDataset

Dataset是Dorado的核心概念之一，抽象了数据集的概念，其核心是对数据集合的定义，包括：

FieldSet	字段集合
RecordSet	记录集合
ParameterSet	参数集合
PossibleRecordCount	记录总数
PageSize	每页记录数
PageIndex	当前页索引

PageCount	记录页总数

ViewDataset是Dataset的扩展定义，可以将其理解为"视图状态的Dataset"，比Dataset多了一些视图（客户端）方面的概念：

MasterLink	主（从）关系
ReadOnly	只读
AutoLoadPage	自动加载分页数据
AutoLoadData	自动加载数据
Async	异步加载数据
InsertOnEmpty	自动添加空记录
EventHandler	事件捕捉器

目前版本(080306)的Dataset如下：

com.bstek.dorado.data.db.SqlDataset
com.bstek.dorado.data.db.AutoSqlDataset
com.bstek.dorado.data.edo.DODataset
com.bstek.dorado.data.custom.CustomDataset
org.marmot.view.dorado.MarmotDataset

目前版本(080306)的ViewDataset如下：

com.bstek.dorado.view.data.DatasetReference
com.bstek.dorado.view.data.FormDataset
com.bstek.dorado.view.data.DatasetWrapper
com.bstek.dorado.view.data.DODataset
com.bstek.dorado.view.data.CustomDataset
com.bstek.dorado.view.data.ViewDatasetAdapter
com.bstek.dorado.view.control.dropdown.DynamicDropDownDataset

注： 在ViewModel中通过getDataset方法得到的都是ViewDataset，作为参数传递到DataListener的方法中的是Dataset。

Field VS ViewField

Field就是字段，主要属性如下：

Name	名字
Label	显示名称
DataType	数据类型
Nullable	可否为空
DefaultValue	默认值

ViewField比Field多了一些视图（客户端）属性，如下：

Validator	验证器
ReadOnly	只读
ValueProtected	值保护
Format	格式
Visible	可见

EditorType	编辑类型
ToolTip	提示信息
DropDown	下拉框
SupportsSum	支持合计