**iRINDev开发规范V1.0**

### 接口规范

1.使用http方法进行分类操作，API资源均以名词体现

|  |  |
| --- | --- |
| **http方法** | **资源对应操作** |
| GET | SELECT |
| POST | INSERT |
| PUT | UPDATE |
| DELETE | DELETE |

2.接口相应参考状态码（是否需要自行封装，或http已代为执行）

|  |  |  |  |
| --- | --- | --- | --- |
| **状态码** | **http方法** | **返回值** | **状态描述** |
| 200 | GET、PUT | 资源 | 操作成功 |
| 201 | POST | 资源、状态数据 | 对象创建成功 |
| 202 | POST、PUT、DELETE | N/A | 请求已经被接受 |
| 204 | DELETE、PUT | N/A | 操作已经执行成功 |
| 301 | GET | 错误页面的link | 对象已永久移走 |
| 303 | GET | 重定向页面link | 重定向 |
| 400 | GET、POST、PUT、DELETE | 错误提示（消息） | 参数列表错误（缺少、格式不匹配） |
| 401 | GET、POST、PUT、DELETE | 错误提示（消息） | 未授权 |
| 403 | GET、POST、PUT、DELETE | 错误提示（消息） | 访问受限、授权过期 |
| 404 | GET、POST、PUT、DELETE | 错误提示（消息） | 资源、服务找不到 |
| 405 | GET、POST、PUT、DELETE | 错误提示（消息） | 不允许的http方法 |
| 409 | GET、POST、PUT、DELETE | 错误提示（消息） | 资源冲突或者资源被锁定 |
| 415 | GET、POST、PUT、DELETE | 错误提示（消息） | 不支持的数据类型 |
| 429 | GET、POST、PUT、DELETE | 错误提示（消息） | 请求过多被限制 |
| 500 | GET、POST、PUT、DELETE | 错误提示（消息） | 系统内部错误 |
| 501 | GET、POST、PUT、DELETE | 错误提示（消息） | 接口未实现 |

3.API版本分为两级：Vx1.x2.其中x1为主功能的版本号，x2为次功能的版本号。每级功能版本号，最多为两位数，版本初始为V1.0。接口版本号放在API字符串首要位置，如：API/V1.0/tickets。

5.API在书写形式是：版本号/主资源(/关联资源/约束/参数集描述)？参数1&参数2……， 如：

GET API/V1.0/dept/parent?id=1234

7.针对过滤、排序，搜索采用“状态名词=状态”的方式来传递信息，如：

* GET API/V1.0/tickets?ticketsq=recently\_closed&state=open&sort=priority #找到最近关闭，状态为open，以priority来排序(数字默认大->小，日期默认近->远)的tickets资源。

9.API字符串均为小写。

10.API返回资源信息均以Map形式体现。

11.分页参数的形式体现在API字符串中，如：

GET API/V1.0/tickets?page=2&limit=100 #以100条数据为一页进行分页，返回第二页数据，page和limit必须成对出现，且如无此参数则默认为返回全部资源条目。

### 编码规范

1. 方法名采用驼峰式。
2. 尽量用StringBuffer来生成字符串；
3. 用logger(sf4j)完成日志记录，不要用System.out.print()。
4. 进行数学计算时，要先进行类型转换再计算，如：float twoThirds = 2f/3
5. Stream、Buffer类资源一定要用try catch捕捉异常，并在finally里进行关闭。
6. 所有可能为null的对象，必须在使用前进行空对象异常判断，并抛出异常。
7. 不同类型类，不能使用equals比较。
8. 三层级以上判断使用switch。
9. 不得在循环里循环调用微服务方法。
10. 循环中禁止使用Thread.sleep()，防止死锁，用wait()代替。
11. 尽量将异常抛出，少用try catch。
12. Try catch写在循环外。
13. 如出现异常，一定要终止异常线程，如：

try {

while (true) {

// do stuff

}

}catch (InterruptedException e) {

LOGGER.log(Level.WARN, "Interrupted!", e);

// Restore interrupted state...

Thread.currentThread().interrupt();

}

1. 在条件语句中，每个分支条件对应的执行块不能相同。
2. 进行数学计算时，一定保证数学逻辑的先决条件。
3. Finally执行块中不能写return。
4. 异常不要用printStackTrace打印，采用LOGGER.log("context", e)代替；
5. IP地址写在可配置的文件中，不予许程序中写死；
6. 创建枚举类型ENUM时，该类型一定以常量的形式进行编码，属性值必须均为私有，且不创建setter方法，采用构造函数的形式初始化；
7. Public static修饰符必须有final伴随；
8. 类属性必须为私有；
9. 后端所有和REST API接口的controller都放在rest包里，前端service层类名以PR结尾放在service包里。
10. 不违反22条的情况下，controller/biz/vo类，类名后面跟controller/biz/vo 如：PlanAndTaskFlowSerivce；

### 界面规范

#### 界面设计规范

1. **通用基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 默认内容区背景色 | #ffffff |
| 默认边框色 | #b1b1b1 |
| 默认边框圆角 | 0px |
| 默认控件背景色 | #f0f0f0 |
| 默认文字大小 | 10pt |
| 默认文本色 | #000000 |
| 默认高亮文本色 | #15428b |
| 默认无效文本色 | #a7a7a7 |
| 默认图标集 | Dorado7 font-awesome |
| 默认图标色透明度 | 0.7 |
| 拖拽参考线颜色 | #ff8040 |
| 默认控件高亮色 | #d8e9fd |
| 默认控件高亮文字颜色 | #000000 |
| 默认数据高亮色 | #a3cdff |
| 默认数据高亮文字颜色 | #000000 |
| 默认控件立体度 | 0 |
| 默认无效控件透明度 | 40 |
| 模态蒙版透明度 | 20 |
| 阴影颜色 | #bababa |
| 阴影尺寸 | 4px |
| 滚动条颜色 | hsva(0, 0%, 0%, 0.94) |

1. **按钮基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 按钮颜色 | #f0f0f0 |
| 按钮边框色 | #cacaca |
| 激活按钮颜色 | #fdfdfd |
| 按钮文字颜色 | #000000 |
| 高亮按钮颜色 | #d8e9fd |
| 高亮按钮边框色 | #8fc0f9 |
| 激活高亮按钮颜色 | #f0f7fe |
| 高亮按钮文字颜色 | #000000 |
| 警告按钮颜色 | #ffabab |
| 警告按钮边框色 | #ff5e5e |
| 激活警告按钮颜色 | #ffc4c4 |
| 警告按钮文字颜色 | #000000 |
| 按钮立体度 | 0 |
| 按钮边框色差 | 1 |

1. **输入框基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 输入框文字色 | #000000 |
| 输入框边框色 | #b1b1b1 |
| 输入框背景色 | #ffffff |
| 输入框背景装饰色 | #ffffff |
| 焦点输入框边框色 | #5097ec |
| 焦点输入框背景色 | #ffffff |
| 焦点输入框背景装饰色 | #f0f7ff |
| 警告输入框背景色 | #faefd5 |
| 错误输入框背景色 | #fce0e0 |
| 只读输入框背景色 | #fbfbfb |
| 输入框立体度 | 0 |
| 下拉按钮宽度 | 18px |
| 下拉按钮颜色 | #f0f0f0 |
| 下拉按钮边框 | Solid(实线) |
| 激活下拉按钮颜色 | #a3cdff |

1. **复选框和单选框基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 复选框基底色 | #ffffff |
| 复选框边框色 | #b1b1b1 |
| 复选框选中块颜色 | #214d82 |
| 激活复选框基底色 | #f6faff |
| 激活复选框边框色 | #3b7ece |
| 激活复选框选中块颜色 | #214d82 |
| 复选框边框尺寸 | 20px |
| 复选框立体度 | 0 |

1. **提示框基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 提示背景色 | #e5f1ff |
| 提示边框色 | #4b9eff |
| 提示关闭按钮色 | #4b9eff |
| 提示文字色 | #000000 |

1. **进度条基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 进度条颜色 | #a3cdff |
| 进度条背景色 | #ffffff |
| 进度条边框色 | #b1b1b1 |
| 进度条立体度 | 0 |

1. **菜单基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 菜单背景色 | #ffffff |
| 菜单文字色 | #000000 |
| 菜单边框色 | #b1b1b1 |
| 菜单装饰色1 | #fbfbfb |
| 菜单装饰色2 | #fbfbfb |
| 激活菜单项颜色 | #d8e9fd |
| 激活菜单项文字色 | #000000 |
| 激活菜单项边框色 | #70a6e6 |
| 菜单项立体度 | 0 |

1. **多页标签基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 标签页颜色 | #f8f8f8 |
| 标签页边框色 | #c5c5c5 |
| 标签页文字色 | #000000 |
| 悬停标签页颜色 | #f0f0f0 |
| 悬停标签页边框色 | #bdbdbd |
| 悬停标签页文字色 | #000000 |
| 当前标签页颜色 | #d8e9fd |
| 当前标签页边框色 | #77b2f8 |
| 当前标签页文字色 | #000000 |
| 标签页边框圆角 | 0px |
| 标签页基线宽度 | 1px |
| 标签页立体度 | 0 |

1. **面板基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 标题栏颜色 | #f0f0f0 |
| 标题栏文字色 | #000000 |
| 标题栏装饰线颜色 | #b1b1b1 |
| 标题栏立体度 | 0 |

1. **工具栏基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 工具栏颜色 | #f0f0f0 |
| 工具栏边框色 | #e3e3e3 |
| 工具栏文字色 | #000000 |
| 工具栏立体度 | 0 |

1. **手风琴基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 栏目颜色 | #f0f0f0 |
| 栏目边框颜色 | #cacaca |
| 栏目文字色 | #000000 |
| 激活栏目颜色 | #d8e9fd |
| 激活栏目边框颜色 | #8fc0f9 |
| 激活栏目文字色 | #000000 |
| 手风琴立体度 | 0 |

1. **分组框基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 分组框文字色 | #15428b |
| 分组框边框色 | #15428b |

1. **分割面板基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 分割栏颜色 | #f0f0f0 |

1. **对话框基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 对话框边框色 | #cacaca |
| 对话框边框透明度 | 50 |
| 对话框边框圆角 | 0px |
| 对话框边框宽度 | 4px |
| 对话框边框立体度 | 0 |
| 对话框标题栏颜色 | #f0f0f0 |
| 对话框标题栏文字色 | #000000 |
| 对话框标题栏立体度 | 0 |
| 激活对话框边框色 | #aeaeae |
| 激活对话框边框透明度 | 50 |
| 激活对话框标题栏颜色 | #f0f0f0 |
| 激活对话框标题文字色 | #000000 |
| 激活对话框标题立体度 | 0 |

1. **日历基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 日历标题颜色 | #f0f0f0 |
| 日历标题文字色 | #000000 |
| 日历装饰颜色 | #f6faff |
| 日历立体度 | 0 |

1. **列表型控件基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 列表行颜色 | #ffffff |
| 列表当前行颜色 | #a3cdff |
| 列表当前行文字色 | #000000 |
| 列表悬停行颜色 | #e8e8e8 |
| 表格奇数行颜色 | #ffffff |
| 表格模拟行颜色 | #f0f0f0 |
| 列表当前行立体度 | 0 |
| 表格标题栏颜色 | #f0f0f0 |
| 表格标题文字色 | #000000 |
| 表格标题栏悬停颜色 | #f8f8f8 |
| 表格过滤栏颜色 | #e4e9ec |
| 列表标题栏立体度 | 0 |
| 表格分组标题文字颜色 | #fb8b0d |
| 表格横线颜色 | #f8f8f8 |
| 表格纵线颜色 | 透明 |

1. **数据块基础设计规范**

|  |  |
| --- | --- |
| 显示对象 | 值 |
| 悬停数据块颜色 | #e8e8e8 |
| 当前数据块颜色 | #a3cdff |
| 当前数据块文字色 | #000000 |
| 数据块边框色 | #f8f8f8 |
| 数据块立体度 | 0 |

1. **界面通用设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 屏幕自适应分辨率1366\*768 |
| 规范2 | 界面Z向层次不宜超过3层 |
| 规范3 | 默认窗口设置下，不应出现水平、垂直滚动条 |
| 规范4 | 页面上方必须有当前页面所在菜单位置的说明文字 |
| 规范5 | 整个系统界面必须采用统一样式的icon图标 |

1. **一般编辑界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 只读文本框和可输入文本框在颜色上必须进行区分，可以输入的文本框以白色为背景，只读的文本框以灰色为背景 |
| 规范2 | 输入框样式要统一 |

1. **弹出框界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 查询弹出框按钮为“查询”“重置”“关闭”  编辑弹出框按钮为“确定”“取消”  保存弹出框按钮为“确定”“取消”  取消按钮点击后会清空输入内容缓存并关闭弹出框 |
| 规范2 | 弹出框右上角均无“X” |

1. **登录框界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 用户名、密码文字应上下对齐（“密码”中间加空格调整宽度） |
| 规范2 | 输入框上下对齐 |
| 规范3 | 在输入框下方设置“登录”、“重置”按钮 |

1. **绑定下拉框默认值界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 当输入框绑定的下拉框不存在默认值时，在输入框中显示淡灰色“请选择”字样 |
| 规范2 | 当输入框绑定的下拉框存在默认值时，输入框中淡灰色显示默认值，展开下拉框时，下拉框内容中默认值对应项默认选中 |

1. **表单必填值界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 表单必填项必须以红色\*或其他符号标识出来，符号位置为输入框与对应文字之间 |
| 规范2 | 表单输入框左侧label靠右对齐 |
| 规范3 | Label和输入框中必须有中文“：” |

1. **有效控件界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 有效的控件以正常颜色显示，无效的控件以灰色显示 |

1. **提示语界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 提示信息中标点符号统一为全角符号 |
| 规范2 | 提示信息如有主语，统一为“您” |
| 规范3 | 复杂的操作或需要后续动作的操作成功后均应给予提示信息 |
| 规范4 | 提示信息窗口不宜太长，宽度不能超过当前屏幕宽度的1/2，当超过此比例时，可视具体情况进行换行 |
| 规范5 | 当按钮为图片按钮时，光标停留需给予浮动提示信息 |
| 规范6 | 提示信息用js动态添加 |
| 规范7 | 视情况，保存功能的按钮加tip进行提示，提示为“快捷键Ctrl+S” |

1. **键盘支持界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 支持回车键登录 |
| 规范2 | 表单支持Tab键移动光标焦点，表格支持回车移动光标焦点 |
| 规范3 | 视情况，保存支持快捷键“Ctrl+S” |

1. **非弹出查询框界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 所有查询输入框都应使用fieldset作为容器 |
| 规范2 | 查询按钮有两个“查询”、“重置”，同样放在fieldset容器中，在容器右侧给出100px的宽度用于从上到下放置按钮。 |

1. **导航工具栏界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 导航工具栏在布局上靠左对齐，且只用pages和pagesize |

1. **输入限制界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 只允许输入数字的输入框控制不允许输入其它字符，而不是输入非法字符后再给予提示 |
| 规范2 | 只允许输入日期、时间的输入框应格式化，而不是输入非法值后再给予提示甚至没有提示 |
| 规范3 | 当输入的内容达到了字段长度限制时，需要控制不允许再输入，而不是保存后自动截断或保存时在给予提示 |
| 规范4 | 对非法字符的控制，限制不可以输入或提交时给予提示 |
| 规范5 | 字段中文放在公共model里，而字段的校验放在每个view自己的model里面 |

1. **列表界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 列表内容中低于7个字符应居中，7个及高于7个字符的居左，数字居中，时间居中 |
| 规范2 | 列表中，如果列数过多，屏幕无法完全显示，默认锁定前3列 |
| 规范3 | 列表表头水平及垂直居中对齐 |
| 规范4 | 列表中选中的记录必须和未选中的记录明显区分 |
| 规范5 | 列表中的数字必须靠右显示 |
| 规范6 | 金额数字采用千分位分隔符，分隔符为西文逗号“,” |
| 规范7 | 如果没有选中列表的项，在点击相关功能操作按钮时，必须有提示“没有选中，请选择！” |
| 规范8 | 列表中如果有的内容显示不完，应在鼠标放上去时，采用tip或其他形式显示全部内容 |

1. **按钮界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 无特殊情况下，每种状态的按钮颜色及响应方式应统一 |
| 规范2 | 按钮上如有只有两个字，则这两个字之间需要一个空格 |
| 规范3 | 按钮之间没有空格，间距由css定义 |
| 规范4 | 所有的按钮必须有图标 |

1. **系统屏幕窗体界面设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 系统屏幕默认支持分辨率为1366\*768 |
| 规范2 | 弹出框大小以显示全部表单内容为宜 |

1. **防错和出错界面通用设计规范**

|  |  |
| --- | --- |
| 要求项 | 描述 |
| 规范1 | 对输入的数据应进行校验 |
| 规范2 | 在某些情况下不适用的菜单和按钮，应设置为“失效”状态(灰色无响应或隐藏) |
| 规范3 | 执行破坏性操作之前，应当获得用户确认，如删除、更新之前。 |
| 规范4 | 尽量为操作提供undo功能，用户可以撤销前一步的操作 |
| 规范5 | 如果发生意外和错误，应当及时给出警告信息和错误信息，提醒用户做出正确的处理 |

#### 界面编码规范

1. **控件命名**
   1. **命名规则**

大部分控件都有id属性，而有些控件还有子控件，其中不能单独存在的子控件没有id属性，有name属性，下面要讨论的命名规范对这两个属性是一样的。

基本规则如下：

1. **命名规范符合java变量名的命名规则，如下的一些命名方式都是不规范**
   * 中文命名，如：部门
   * 带特殊符号，如：dept$id, dept^id, dept@id
   * 带横线或点，如：dept.id, dept-id
   * 数字命名，如：20131016
2. **首字母小写,除了第一个单词,所有单词首字母大写**
3. **要采用有意义的命名方式，可以通过名字就能看出是一个什么样的控件,如**
   * dataTypeProduct
   * dataSetProduct
   * dataGridProduct
   * buttonAdd
   * actionUpdate
4. **要采用统一的命名规则，如下的命名就是不规范的：**
   * 页面A的新增按钮为buttonAdd,页面B的新增按钮命名为buttonNew
   * 页面A的新增按钮为buttonAdd,页面B的新增按钮命名为btnAdd
   1. **命名规范示例**
5. **示例1：完整命名**
   * dataTypeProduct
   * dataSetProduct
   * dataGridProduct
   * buttonAdd
6. **示例2：简写命名**
   * dtProduct
   * dsProduct
   * dgProduct
   * btnAdd
7. **示例3：DataType的命名**

DataType可以用前缀词的方式命名，如：

* + dataTypeProduct(完整命名)
  + dataTypeCategory(完整命名)
  + dtProduct(简写命名)
  + dtCategory(简写命名)

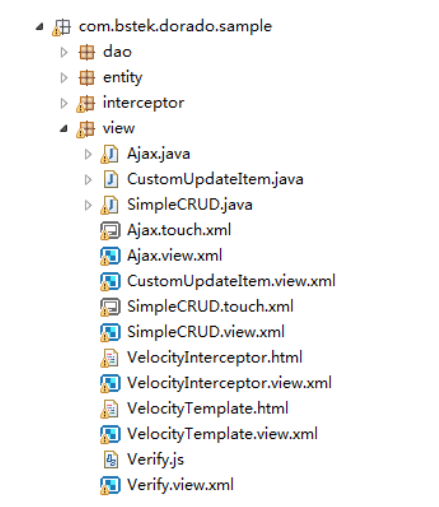
但由于DataType本身是一个模型对象，并没有数据，与Java类的定义有很多相似性，因此在很多时候我们会采用Java类的命名方式，例如:

* + Product.java对应的DataType的名称为Product
  + Category.java对应的DataType的名称为Category
  1. **控件命名简写对照表**

|  |  |  |
| --- | --- | --- |
| 控件名称 | 简写 | 完整名称 |
| Accordion | accord | accordion |
| BlockView | bview | blockView |
| Button | btn | button |
| CardBook | card | cardbook |
| CheckBox | chk | checkbox |
| Container | container | container |
| Control | control | control |
| DataSet | ds | dataSet |
| DataType | dt | dataType推荐用Java类的类名的方式命名，首字母大写，如： Employee、 Dept、 Product |
| DataPilot | pilot | datapilot |
| DateSetDropDown | dsdd | datasetDropDown |
| DatePicker | datepicker | datePicker |
| DateTimeSpinner | dtspinner | datetimeSpinner |
| Dialog | dialog | dialog |
| FieldSet | fset | fieldset |
| FloatContainer | container或fcontainer | floatContainer |
| FloatPanel | panel或fpanel | floatPanel |
| Grid | grid | grid |
| GroupBox | gbox | groupbox |
| HtmlContainer | container或hcontainer | htmlContainer |
| IFrame | iframe | iframe |
| Image | img | image |
| Label | label | label |
| Link | link | link |
| ListBox | listbox | listbox |
| ListDropDown | listdd | listDropDown |
| Menu | menu | menu |
| NumberSpinner | spinner或nspinner | numberSpinner |
| Panel | panel | panel |
| PasswordEditor | pwdeditor | passwordEditor |
| ProgressBar | prgbar | progressbar |
| RadioGroup | radiogroup | radioGroup |
| SimpleButton | simplebtn | simpleButton |
| SimpleIconButton | simplebtn或simpleiconbtn | simpleIconButton |
| Slider | slider | slider |
| SplitPanel | sp或splitpanel | splitPanel |
| SubViewHolder | subviewholder | subViewHolder |
| TabBar | tabbar | tabbar |
| TabColumn | column | tabColumn |
| TabControl | tabcontrol | tabControl |
| TextEditor | editor | textEditor |
| Tip | tip | tip |
| ToolBar | toolbar | toolbar |
| Tree | tree | tree |
| TreeGrid | grid或tgrid | treeGrid |
| Trigger | trigger | trigger |
| VerticalTabControl | tabcontrol或vtabcontrol | veritcalTabControl |
| YearMonthDropDown | ymdd | yearMonthDropDown |
| YearMonthPicker | ympicker | yearMonthPicker |
| YearPicker | ypicker | yearPicker |
| MonthPicker | mpicker | monthPicker |

1. **view.xml和touch.xml**
   1. **目录管理**

这两种类型的文件都需要被编译到classes目录下，推荐管理方式是将这些视图文件和java文件放在一起，参考目录结构如：

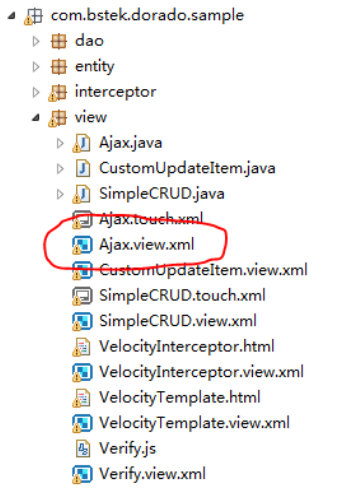


通常都将view.xml和touch.xml文件统一到一个目录下便于后期管理。

* 1. **页面访问**

1. **默认访问模式**

如下图的Ajax.view.xml文件：



可以通过如下两种方式访问这个页面：

http://localhost:8080/sample-center/com.bstek.dorado.sample.Ajax.d

http://localhost:8080/sample-center/com/bstek/dorado/sample/Ajax.d

但是不推荐第二种方式，第二种方式由于相对路径的原因，不利于当前页面访问系统内的其他资源(css或js文件)或其它的视图页面(view.xml,touch.xml)推荐采用第一种方式。

1. **简化URL字符串**

另外我们还可以通过dorado中提供的view.root的参数配置简化URL字符串，默认的配置如下：

view.root=classpath:

如果我们将上面的配置调整为：

view.root=classpath:com.bstek.dorado.sample

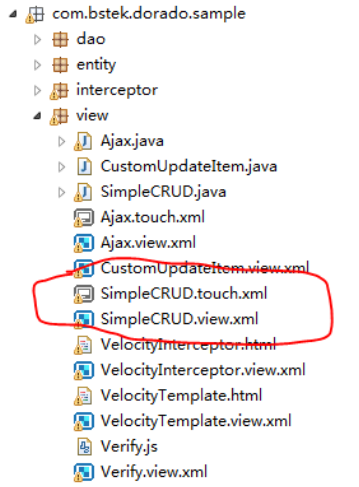
修改后，我们就可以直接通过如下的URL访问这个View文件了:

http://localhost:8080/sample-center/Ajax.d

这个变量配置在doradohome的configure.properties中。

1. **关于touch.xml**

如果在相同目录下,有同名的视图文件(view.xml和touch.xml)，如下图的SimpleCRUD.view.xml和SimpleCRUD.touch.d:



如果是PC设备访问，则系统会自动解析SimpleCRUD.view.xml并输出，如果是Touch设备访问，则系统自动解析SimpleCRUD.touch.xml并输出，如果Touch设备访问的URL不存在对应的touch.xml，则系统默认还是解析对应资源文件的.view.xml文件并输出。

因此如果系统中要做touch设备的开发，可以考虑将.touch.xml文件与.view.xml文件放在同一位置。当然有的项目有自身的特殊管理需求，可能将所有的touch文件统一管理在另一个目录也可。

1. **Model.xml**
   1. **基本用途**

model.xml文件用于统一管理系统中全局的DataType和DataProvider等对象，其中最常用的是DataType的管理。带来的好处列举如下：

* + 统一定义字段的中文标题，业务页面引用即可，无需重复定义，减少开发工作量
  + 统一定义数据类型字段的显示格式，这样不同页面如果引用相同的数据模型时就有相同的数据格式
  + 统一定义数据的默认值
  + 统一处理数据字典类的code和label的转换，如：性别，在职状态，工程师级别等

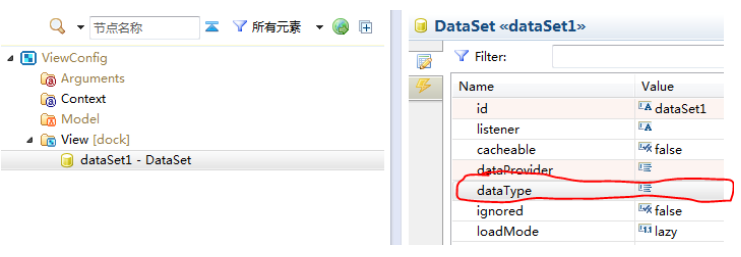
总之，通过model.xml文件我们可以统一系统中数据模型的定义，提高开发效率和统一开发效果。

model.xml文件的存放路径可以由configure.properties配置文件指定：

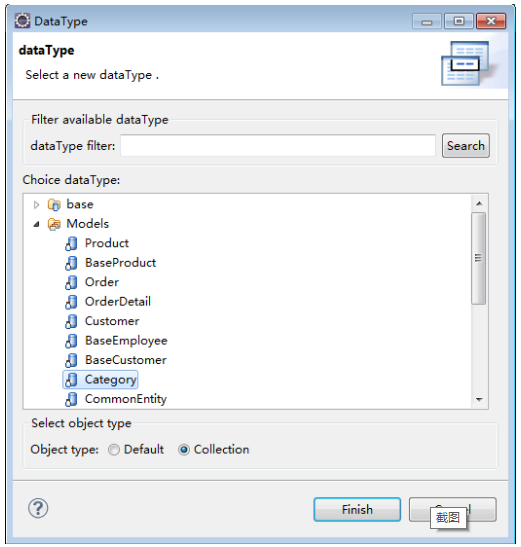
model.root=classpath\*:models

* 1. **使用方法**

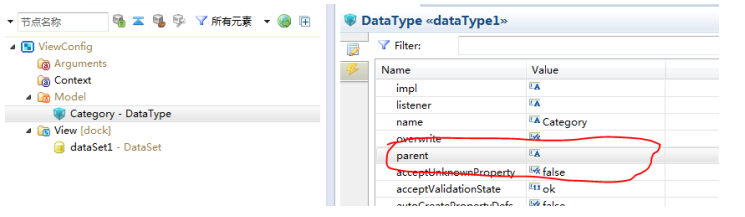
当我们在model.xml中定义好全局的DataType后，在开发视图文件时，可以直接引用model.xml中的DataType，如下图我们设置DataSet的dataType属性时：

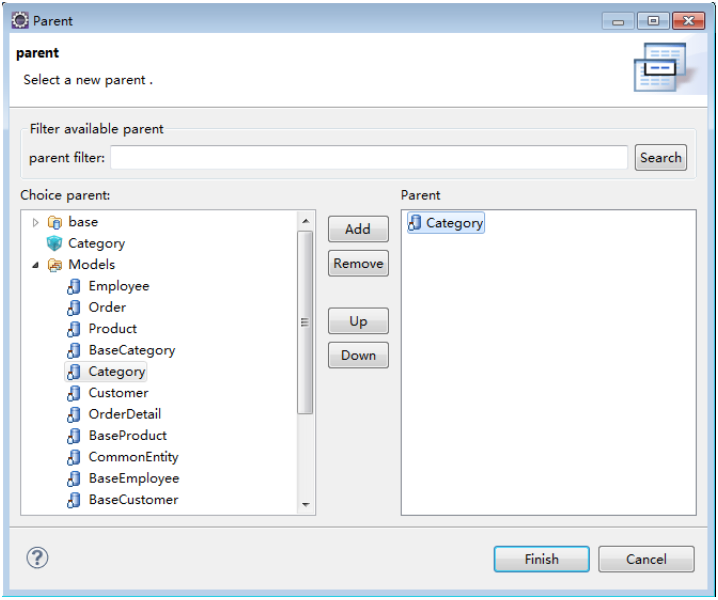


编辑时，对应的DataType的设计向导窗口中就可以选择定义在model.xml中的全局DataType对象：

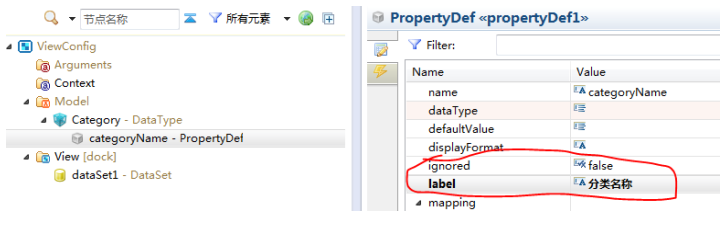


这样业务开发人员就不需要在View中重复定义这个DataType，大大提高开发效率，如果在本页面这个DataType有个性化设定，例如某一个字段的中文标题，在当前页面和数据模型不一样。这其实是一种页面个性化的需求，则开发人员可以在当前页面的Model节点中添加一个新的DataType,并设置这个DataType的parent为model.xml中对应的全局的DataType对象，如下：





并修改字段对应的PropertyDef的label属性，进行页面个性化设置：



* 1. **设计规范**

1. **基本设计规范**
   * 通常而言每一个POJO都应该在model.xml中定义一个对应的DataType，并通过matchType与之关联，matchType可以带给我们的一个好处是，当展现层接收到一个POJO的时候，即使你没有指定这个POJO对应的DataType，dorado中的数据模型处理机制就可以通过model.xml中POJO与DataType的一一映射关系而自动的找到对应的DataType做处理，这有利于简化系统代码的编写，后面阐述DataType设计规则的时我们再结合具体的应用场景说明matchType的作用
   * DataType的matchType的规定：必须一一对应，不能将不同的DataType关联到同一个POJO上或将多个POJO绑定到不同的DataType上，如果存在这种需求就用creationType规避
   * model.xml文件可以有多个,可根据项目需要设计
   * model.xml文件需要统一管理，放置在一个统一的目录下，对于大型开发团队来说model.xml文件应该交给一个人专门管理变更，业务页面引用model.ml中的DataType对象，做一些个性化的设定处理
   * 通常而言我们不将数据模型的关系定义在model.xml中，也就是不在model.xml中建立立体结构的数据模型，都是平面化的数据模型，如有需则应该在业务页面的视图模型中定义
2. **关于对象系统的维护**

上述设计规范中的最后一个是关于对象关系维护是否设计在model.xml中的一个基本约定。我们列举一个例子详细说明。如下结构的POJO对象：

public class Category implements Serializable {

private static final long serialVersionUID = 6076304611179489256L;

private long id;

private Category parent;

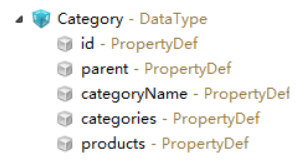
private String categoryName;

private String description;

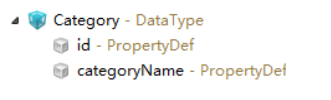
private Collection<Category> categories;

private Collection<Product> products;

在model.xml中设计DataType的时候不要定义对象类型的子属性，如下图的定义是不推荐的：



而是应该只定义简单属性：



而最终在View层面，开发人员根据需要添加对应的对象类型的子属性。这么设计的原因说明如下：

如果在model.xml文件中就设计好了对象关系，而与此对象模型相关的数据提供者(DataProvider)如果是返回一个完整数据包，例如我们获取一个指定的Categroy对象的时候，同时返回了对应的parent,categories,products等相关对象的所有数据。由于这个模型会被公用，而对应的数据服务的提供者不发生变化的时候，只要引用这个模型的页面都会获得所有的数据，页面Ajax数据输出时就会将所有的数据输出到客户端，而有些页面可能根本就不关心其中的子对象，或不关心其中的部分子对象。这种情况下不仅会造成网络资源的浪费，还会产生页面处理性能下降(页面数据量增大从而增加了浏览器的运算压力)。

当然这个规则并不是绝对的，经验丰富的开发人员可以根据页面的需要，为相同的模型在不同的页面中提供不同的数据提供者(DataProvider)，确保没有冗余数据输出的情况下，是可以将对象关系模型维护在model.xml文件中的。但是如果你无法很好的控制团队的开发能力，那么model.xml中不维护对象关系是最佳选择。

1. **运行模式**

Dorado提供了运行模式的管理，简单的说就是在doradohome文件夹下，针对configure.properties配置文件和context.xml文件提供不同运行态的管理。运行态管理是通过configure.properties的core.runMode参数控制的：

core.runMode=debug

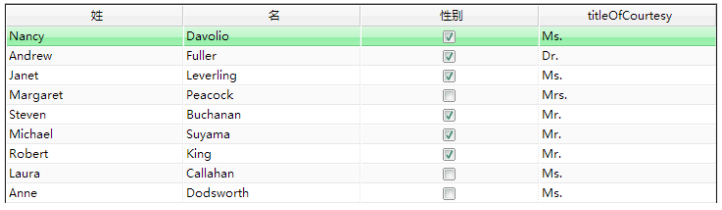
如果设置为debug，则系统运行时会自动加载configure-debug.properties和context-debug.xml配置文件，并与configure.properties以及context.xml文件中的相关配置合并，并且含debug关键子的文件中的配置优先级更高，会覆盖不含debug关键字的文件中的配置。而在发布时，如果我们清空core.runMode的值，或者配置为production，则系统就不会加载configure-debug.properties和context-debug.xml文件，不会做合并处理。利用这种机制，我们可以将系统中有这种运行模式区别的参数都归类整理到这两个不同的文件中，并最终通过core.runMode参数控制运行模式，这样就方便很多。举一个例子：dorado提供了这么一个参数：

view.outputPrettyJson=true

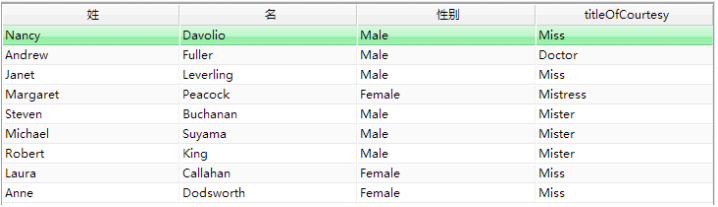
这个参数的含义是：访问页面的时候，视图输出的页面内容中的JSON是否采用比较规范的格式化输出风格。一般情况下在开发时我们为了更好的调试系统，我们希望它能输出规范化的格式文件，但是在生产环境下我们更希望其采用压缩格式，不仅可以增加安全性还可以提高输出的效率。这是两个不同的需求，利用dorado提供的运行模式管理，我们可以在configure.properties设置view.outputPrettyJson为false,而在configuredebug.properties文件中将这个参数设置为true。在开发时我们设置core-runMode为debug,而正式运行时调整为production模式就可以。这种开发模式的使用将有利于简化项目发布的复杂性和避免一些运行态差异带来的低级错误。

1. **DataType设计**
2. **区分字段翻译还是业务关联**
   * 简述

在页面开发和设计时，我们经常会遇到一些大对象，这些对象动则几十个到几百个字段，其中很多字段都是保存了相关业务对象的code，在编辑的时候我们要显示相关业务对象的label.如下图中的“性别”和“titleOfCourtest”字段：



数据库中保存的是code，但是我们希望展现给用户的时候显示对应label，这样更容易看懂：



dorado开发这种code转label的工作有两种实现方式：

* 方式一：通过mapping实现，实现时开发人员将mapping的值穷举，并输出到客户端，同时和对应的字段做绑定设置，这样在页面初始化过程中，就可以自动将code和mapping中的值比对，找到对应的label后并显示；
* 方式二：通过调整数据的模型实现，类似的设计有POJO转VO，新建一个VO对象，在VO对象中添加POJO中code对应的label字段，并在查询初始化的时候同时初始化label的值，并最终将整个VO对象输出到客户端，客户端就隐藏code字段，而是直接通过label字段来显示值；当然在dorado的模型设计中支持虚拟属性，VO的创建不是必须的，可以基于虚拟属性添加对应的label属性。
  + 用mapping还是修改数据模型

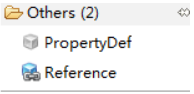
在业务开发的过程中，这两种方式如何区分呢？区分规则：

* 面向对象的关系来区分：如果相互关联的是两个业务对象，则一定用方式二。例如：淘宝订单中的用户和商家，用户和商品，商家和商品等
* 数据量来区分：code和label在数据库中有专门的表，可以单独维护，并且对应表的数据量是几十个数量级的就可以用mapping实现，如果数据量是几千个数量级的就应该通过调整数据模型实现，如果是几百个数量级的要根据页面实际情况综合判断
* 静态动态区分：一般来说数据量很少，且在整个系统的运行期很少会发生变化的数据，则可以用mapping实现。如：性别，员工上班状态，流程审批状态等

之所以要慎重选择转换方式的原因：于两种转换方式所采用的开发手段有很大区别，并且相对来说mapping的开发更为简洁，很多开发人员在未区分的情况下将上面应该通过数据模型转换实现的需求通过mapping方式实现，并且在测试阶段可能没有任何问题(理论上可以实现相同的功能)，但是一到正式运行环境就发现速度很忙。究其原因就是正式环境下由于数据库更完整，mapping穷举的数量很大，而mapping运行机制中要求将mapping中穷举出来的所有数据都输出到客户端，有时候这个数据量可能是几千条，有时候可能是几十万条，这对于网络和浏览器来说都是不可承受的。因此在设计的时候就需要注意到这个规则，我们在很多管理不善的项目中都发现项目上线之后再匆忙的将mapping实现修改为修改数据模型实现的应急开发。

1. **子对象**

DataType下可以添加两种类型的属性对象：

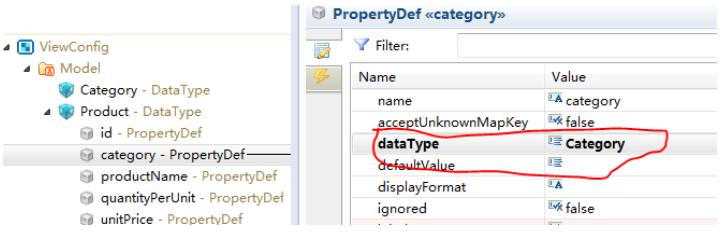


* PropertyDef：可定义简单类型或对象类型的数据，也可以是Collection类型的数据；
* Reference：对象类型的数据，也可以是Collection类型的数据，简单类型的数据不推荐用Reference定义

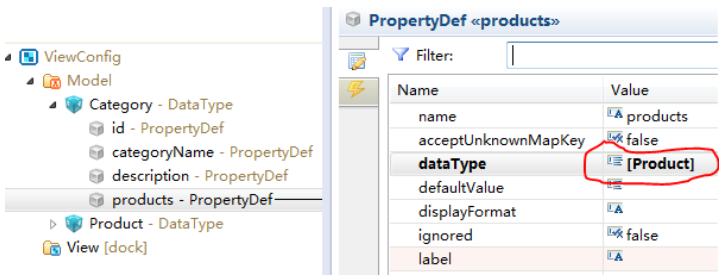
PropertyDef一般用来定义POJO中的简单类型的子属性或子对象，而Reference一般用来定义对象类型的数据或对象集合类型的数据。其中Reference对象有自己独立的数据装载器，可以实现延迟数据装载，而PropertyDef中的数据是随着当前的Entity对象直接下载的。

* + PropertyDef

一般用于定义简单类型的数据，如：String, int, Integer, long, Long,char,Date,Float,float等；如果是对象类型的数据，则我们还需要进一步通过其dataType属性定义内部对象的对象结构。如下图(Product中含category属性)：



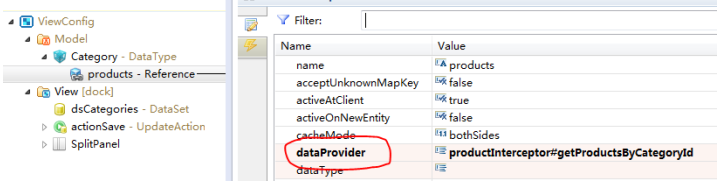
如果是集合类的数据，则也需要定义dataType属性，且外面还要包含中括号，表示是一个集合：



上面的对象类型的PropertyDef或集合类型的PropertyDef并不是一定要设置dataType属性的，事实上当你从DataProvider放回对应的数据对象的时候，如果DataProvider判断到其中包含的子属性是一个对象或对象集合的时候，就会从model.xml的配置中根据其中DataType的matchType属性找到对应的POJO的对应的DataType，并用这种DataType作为当前PropertyDef的默认dataType属性。关于matchType的说明已经在上文的03. modal.xml中提到过，这儿就是一个具体应用场景。

* + Reference

一般用于定义POJO中对象类型的属性或集合类型的属性。同样我们也需要为Reference定义dataType属性，以便于DataSet可以知道这个Reference属性的内部数据结构。另外同样根据model.xml的基本设计规范，这个dataType在满足基本设计规范的情况下是可以省略的。看起来Reference与PropertyDef都可以定义子属性为对象或对象集合。不同的地方是，它可以定义自身的数据提供者(DataProvider)，如：



Reference中的数据是异步加载的，如上图中的Category对象有一个名称为products的Reference，这就意味着当你在客户端获取到一个Category数据对象的时候，其products中的值可能还没有加载到客户端，只有当你尝试着访问它的时候，它才会触发异步的数据加载动作，如：category.get(“products”);

* + PropertyDef还是Reference？

根据上面的描述，做为开发人员应该可以清楚的做如下判断：

当某一个POJO中的某一个属性为简单数据类型的时候，我们就一定用PropertyDef定义。

但是如果这个子属性是一个POJO或一个对象集合的时候，我们该怎么选择呢？

我们在DataType的设计时，对自己提出以下几个问题：

* 得到子属性的值是不是一个非常耗时的工作，例如：对应的SQL查询非常耗时；

需要调用第三方系统的接口才能获得的数据，如：WebService,工作流接口，规则引擎等；

* 子属性的数据量是不是非常大，例如：子属性是一个数据集合，有几千条数据；
* 这个子属性如果是一个集合，是不是还要支持翻页操作；
* 在页面操作过程中，应为目的的不同，是不是某些操作场景下可以忽略这个子属性的，如：多标签页场景，用户一般情况下可能只操作其中的某几个标签页，而另外几个标签页中的数据很少操作，而这个子属性的数据恰好是在这几个不常用的标签页对象中。同样适合这种分析的场景有：一开始隐藏起来的Dialog或某些Fieldset或GroupBox中显示的子属性。

如果以上问题的任意一个你根据业务场景的分析所得到的答案是确定的，那么这个子属性就应该用Reference来设计。

Reference中的数据是异步加载的，采用这样的设计后，即使这个数据没有加载到客户端，也不会对用户的操作带来任何影响，同时这种设计也有利于页面快速的展现给系统的操作人员，从而提高更好的用户体验。

1. **不支持循环引用的数据实体**

DataType可以用于定义复杂数据对象，以及定义对象关系。但是对象关系定义时，要避免嵌套逻辑关系的定义，因为最终当一个数据实体被输出到客户端时，Java对象会根据DataType的定义决定输出那些属性到客户端，而如果你的DataType的定义中形成了嵌套的闭环关系，会导致这个对象无法被正常的转换为一个合适的JSON对象输出到客户端。为避免这种情况Dorado会在服务器端输出时提起做这种检查，一旦发现这种问题就会直接抛出如下的异常:

ERROR:com.bstek.dorado.view.output.DataOutputter - java.lang.IllegalArgumentException:不支持循环引用的数据实体

举例：Category和Propduct两个Java对象是一对多的关系，Category中含products属性，Product又包含Category对象。

Category c = new Category();

Product p = new Product();

p.setCategory(c);

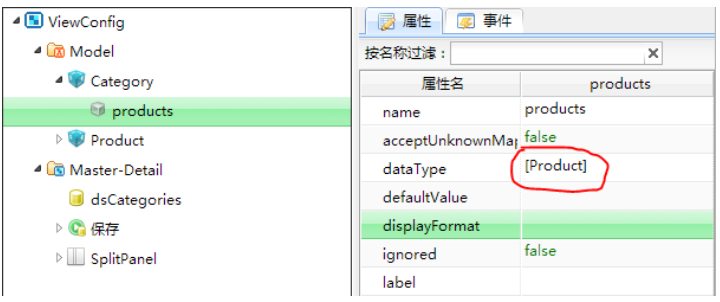
c.getProducts().add(p);

这个时候上面的Category和Product是相互引用的关系，而这个时候直接转Json是会出错的。

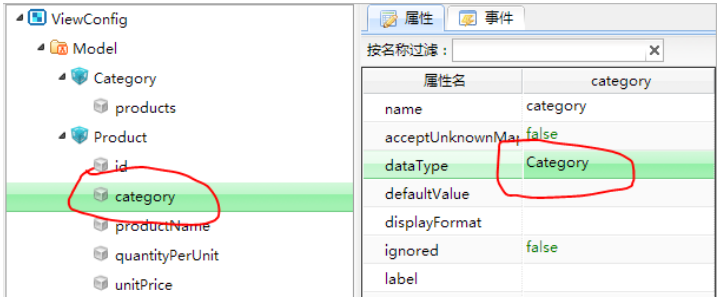
当然在日常的开发中我们可能并不需要输出这种相互引用的嵌套关系。我们而采取一些技巧避免这种问题，如输出Product信息的时候我们就不输出Product中的category属性了。

在Dorado中由于我们是通过DataType来定义Java对象的结构的，通常我们用PropertyDef来定义一个对象的属性，如果你通过PropertyDef定义对象的属性，一旦你将一个Java对象通过Dorado的Ajax技术返回到前台的时候，Dorado就会将Java对象对应的这个属性的值转换为Json输出到前台。由于这个规则，如果你返回的是一个具有相互应用关系的两个不同对象，而恰好DataType设计时，又正好通过PropertyDef定义导致了DataType的相互引用，这个时候就会出现上面的“不支持循环引用的数据实体”的错误。

以上面的Category和Product对象为例，看下面的DataType配置，就会导致错误。Category下有一个名称为products的PropertyDef，他DataType是Product集合类型的：

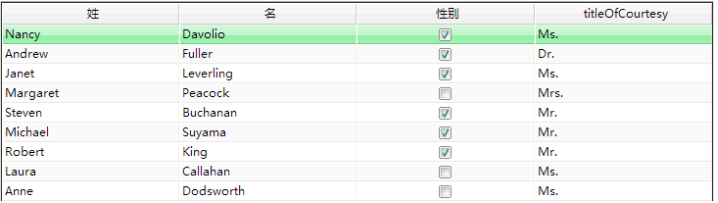


而对应名称为Product的DataType的内部又含有category对象的定义，并且dataType属性为Category：



1. **Boolean和boolean的区别**

虽然这个差别与Java中Boolean和boolean的差别一样，但是在展现层开发的时候很多人还是会忽略这两种类型的差别，所以再强调一下。如下图中的“性别”数据：

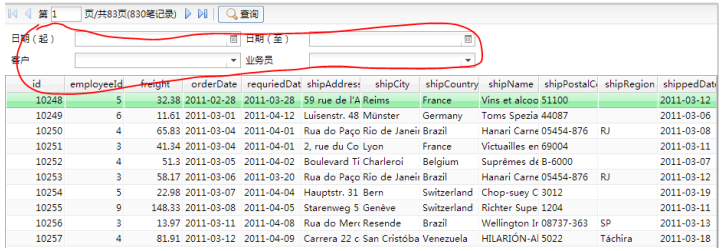


复选框选中代表true,不选中不一定是false，还有可能是null，只是在Java或JS运算中null的运算结果也是false。因此当一个属性允许存在null的情况下就要用对象类型定义DataType，类似的DataType还有：

* int和Integer
* float和Float
* double和Double

1. **查询条件设计注意事项**

一个典型的查询页面中存在大量的这种页面：



一般dorado开发人员都会将查询条件输入区域利用AutoForm控件协助开发，虽然AutoForm可以在不绑定DataSet的情况下使用，但是由于很多Dorado控件以及数据控制都与数据模型相关，因此对于这种查询条件类型的数据，我们都建议专门创建一个DataType描述所有的查询条件，并将AutoForm的dataType与之关联，最后再设置AutoForm的createPrivateDataSet属性为true(也可以直接添加一个DataSet，并将AutoForm绑定到这个DataSet上),使得AutoForm内部的数据最终通过DataSet管理起来。

于此类似的，如果不是通过AutoForm定义查询条件对应的展示控件，而是通过多个TextEditor等控件定义的界面，我们依然建议你为这些查询条件定义DataType,并在页面上创建一个DataSet，并最终都将这些编辑框数据绑定到这个专门存储查询条件的DataSet中。

这种编程方式有利于后期的维护，同时数据逻辑的处理可以和业务数据逻辑的处理保持一致，使系统有统一的编码风格。

1. **PropertyDef的默认值**

业务页面开发时，经常有会有默认值的设置需求，由于dorado统一的采用数据模型管理机制，默认值的设置与传统不基于数据模型的网页开发技术有所差别，如下的是基于传统网页技术设置默认值的方式：

<form action="form\_action.jsp" method="get">

First name: <input type="text" name="fname" />

Last name: <input type="text" name="lname" />

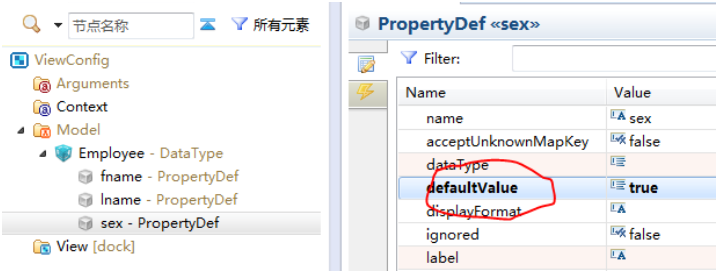
<input type="submit" value="Submit" />

</form>

而在dorado中如果你直接设置一个绑带了DataSet或Entity的TextEdtior控件对象的text值是没有意义的，因为数据绑带的关系它会忽略自身的text属性。下面我们看看基于数据绑带机制下的默认值实现办法。

* + DefaultValue属性
* 原理说明

TextEditor类型的文本编辑器的数据模型都是通过DataType的PropertyDef描述的，PropertyDef提供了一个defaultValue属性，用于定义默认值：



但是这是基于数据模型的设置方式，只有当你创建一个新的Entity对象的时候这个默认值才起作用，这样说你可能有点难以理解，但是我用Java类的定义来描述，你可能就容易理解了，如下的Java类的定义：

public class Employee {

private String fname;

private String lname;

private boolean sex = true;

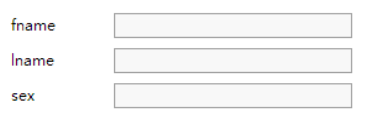
...

}

其中的sex属性的设置，只有你new Employee()的时候才会起作用。PropertyDef的defaultValue的运行机制和这个一样。

* 补充知识：理解当前记录

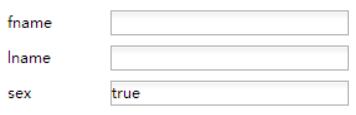
由于数据绑带机制,如果当前绑带的DataSet中没有数据，则编辑框会展现灰化禁用标示：



如果这个时候想要使当前编辑框可编辑，就需要创建一个新的Entity对象，假设当前AutoForm绑定的DataSet为dataetEmployee,则可以通过如下的代码添加一个新的Entity对象：

dataSetEmployee.insert();

insert方法执行后，编辑框就变成可执行状态，并且PropertyDef的defaultValue也发挥作用了，如下的sex字段：



* + insert({})方法

上述范例中我们其实也可以通过如下的代码实现默认值的处理：

var entity = dataSetEmployee.insert();

entity.set("sex", true);

如果有多个属性要设置默认值，可以利用set方法对多个属性一次性的赋值：

var entity = dataSetEmployee.insert();

entity.set({

"fname":"defaultFName",

"lname":"defaultLName",

"sex":true

});

但有时候也可以通过insert方法实现更简洁的写法：

dataSetEmployee.insert({"sex":true});

下面是多个属性的赋值：

dataSetEmployee.insert({

"fname":"defaultFName",

"lname":"defaultLName",

"sex":true

});

当你有多个属性需要设置默认值的时候， 不推荐使用下面的赋值方式：

var entity = dataSetEmployee.insert();

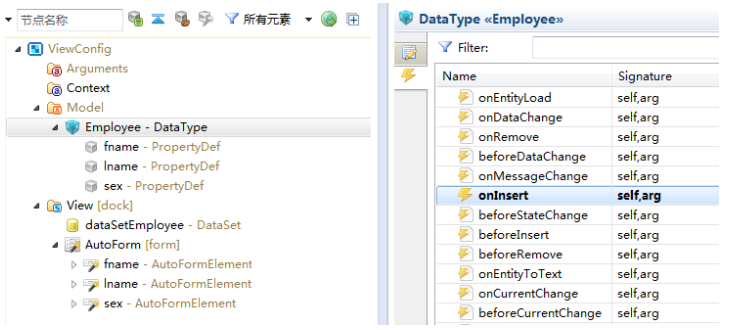
entity.set("fname", "defaultFName");

entity.set("lname", "defaultLName");

entity.set("sex", true);

* + onInsert事件

除了上面介绍的几种设置默认值的方法，还有一种方法是利用数据模型的事件触发机制，当我们新增一个Entity对象的时候，该Entity对象所属的DataType对象就会触发onInsert事件：



并且该事件的arg中有一个entity参数被传入，该参数表示是当前新增后的Entity对象，因此默认值的处理，我们也可以通过该事件实现：

// @Bind @Employee.onInsert

!function(self, arg){

arg.entity.set({

"fname":"defaultFName",

"lname":"defaultLName",

"sex":true

});

}

### 注释规范

1. 源文件头注释

/\*\*

\* @description …….

\* @history <author> <time> <version > <desc>

\* @author jacky

\* @create 2018-04-15 8:48

\*/

说明：Description一项描述本文件的内容、功能、内部各部分之间的关系及本文件与其它文件关系等。History是修改历史记录列表，每条修改记录应包括修改日期、修改者及修改内容简述。

1. 函数头注释

/\*\*

\*

\* @param params

\* @return

\*/

1. 条件判断语句必须在每一次判断是做出注释说明。
2. Swagger仅对接口类进行注释。
3. Swagger的@Api注释整个接口类，value参数和tags参数必须填写。
4. Swagger的@APIOperation注释接口中的方法，value和notes必须填写。
5. Swagger的@ApiParam注释接口方法中的参数，value和required必须填写。
6. 如需要对pojo等实体类进行注释说明 使用Swagger的@ApiModel，value和description必须填写。
7. 如需要对pojo等实体类属性进行注释说明，使用Swagger的@ApiModelProperty，value和required必须填写。