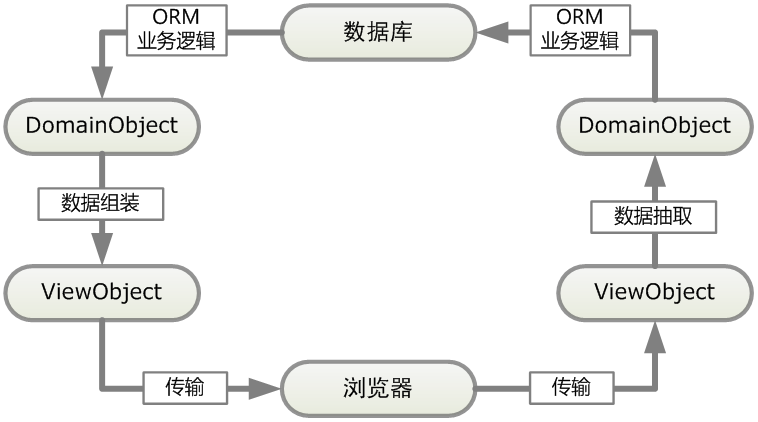
## 1、立体数据模型

立体数据模型是Dorado7中全新的、最为核心的概念。

### 1）什么叫立体数据模型

Dorado7主张数据和表现的分离，将界面上的数据抽象为专门的数据模型，称为一个独立的子系统。

从数据流转的角度分析WEB应用中的数据处理过程



分析从数据库到浏览器：

* 首先从数据开始，例如使用Hibernate，那么这些数据就会被ORM处理机制转换为通过一堆JavaBean描述的对象，称为领域对象模型（DomainObject）；
* 之后领域对象可能会经过业务逻辑的组装，重新拼装或改写，变成一种全新的格式（也可能形式上没有变化，但是作用上已经发生改变），这些对象为前台提供服务，称为视图数据对象（ViewObject）；
* 之后视图数据对象通过HTTP传输，被response到浏览器中，成为浏览器前端所表达的数据。

分析从浏览器到数据库：

ViewObject输出到浏览器之后，这些数据被展示给客户，由用户做各种操作，可能改写其中的部分信息，最后被提交到服务端，并被转换为ViewObject。之后服务端要抽取这些ViewObject中的信息组装为ORM中所需要的领域对象，在持久化到数据库中。

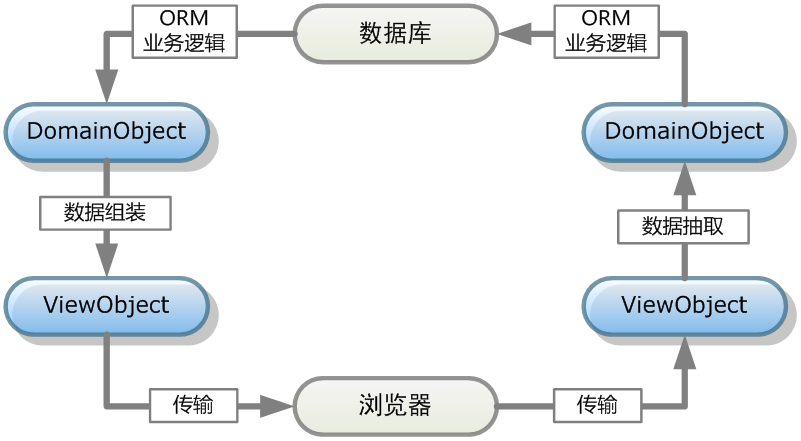
平面数据与立体数据

概念：

* 平面数据：通过键值来表达数据与数据间的关系。例如关系型数据库中的数据。每一条数据都是独立的，它们之间的关系是通过外键值指向另一个数据的主键值。
* 立体数据：通过引用来表达数据与数据间的关系。例如通过Hibernate获得的领域模型数据。如我们拿到一个部门对象，部门对象内还包含所有的雇员对象。

以数据载体形式的角度再度观察

根据这平面数据和立体数据的概念，将前面WEB应用中的数据处理过程的图，进行着色区分这两种数据，其中的立体数据被标以蓝色。



|  |  |  |
| --- | --- | --- |
| 节点 | 数据类型 | 说明 |
| 数据库 | 平面数据 | 一般采用关系型数据库 |
| DomainObject | 立体数据 | 经过ORM转换后的立体数据 |
| ViewObject | 立体数据 | 经过一定形变，但还是立体数据 |
| 浏览器 | 平面数据 | 已经被溶解到Form的input控件中，比二维表的平面数据结构还糟一些 |
| ViewObject | 立体数据 | 转换Form中的input数据，变形为立体数据 |
| DomainObject | 立体数据 |  |

整个过程充斥着平面到立体、立体到平面的恶性循环，整个过程非常不流畅。这也是传统WEB开发中相对比较麻烦和复杂的一个重要原因。在关系数据库到DomainObject端的相互转换我们尚可以依靠Hibernate这些相对成熟的ORM工具帮助我们以较小的代价完成转换工作，但Java端到浏览器端立体到平面，平面到立体的转换过程却需要我们完全靠自己的手工完成。

对数据形式进行一些归纳

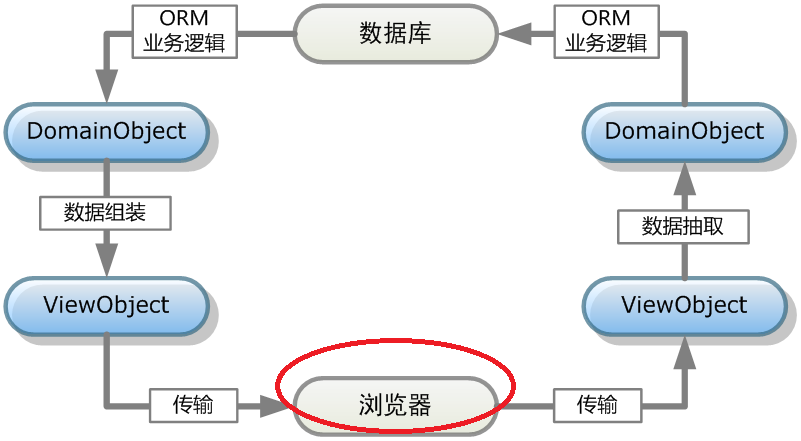
* 数据的本质是立体的，现实生活中的检验我们知道对象之间都是引用的关系；
* 但是保存在数据库中的数据是平面的，关系型数据库的好处是有利于存储和检索；
* ORM把数据库中的平面数据映射成立体的；
* XML、JSON数据可以方便的描述较简单立体数据，它可以描述数据与数据之间的关系，但有局限性：它可以描述树状的数据，但是无法描述网状的数据，所幸日常开发中很少遇到网状的页面展现形式和表达式；
* 网页Form中的数据是平面的；
* 在进行运算时，处理立体数据比处理平面数据容易；
* Dorado5的DataSet是平面的

### 2）为什么需要引入立体数据模型

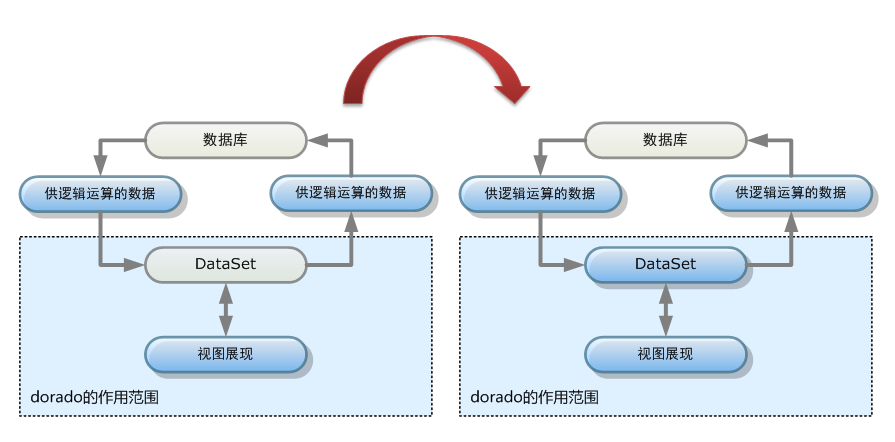
引入立体数据的原因：

* 数据本质是立体的，因此当界面越能展现和处理这种立体关系时，它对用户友好程度就越高；
* Hibernate（ORM）的使用经验告诉我们。平面数据和立体数据进行互转时需要付出很大的代价。

因此，我们需要在Client端提供立体数据模型以提高界面的友好度、同时改善应用数据处理过程的流畅性。即将红框中的部分变为立体数据模型：



总结如下：



从左边平面DataSet到右面立体DataSet

### 3）怎么实现立体数据模型

数据载体：

* 用于封装和承载数据的对象，通常由数据实体和数据实体集合这两种对象构成。
  + 数据实体是指一个拥有1到n个属性的对象。在Java中通常表现为POJO Bean或Map；在JavaScript中通常表现为JSON对象。
  + 数据实体集合用于管理0到n个相似的数据实体。在Java中通常表现为List/Set；在JavaScript中通常表现为Array。

数据集：

* 数据集是指那些根据业务逻辑或编程的需要提取出来的一堆数据。
* 一个数据集的所有数据往往在逻辑上具有很强的相关性。比如当我们开发一个部门信息的维护界面时，会提取出某部门及其中所有雇员的信息作为一个数据集。

Dorado7中的数据载体：

Dorado7在Server端不提供专用的数据载体对象，而是直接使用POJO Bean/Map以及List/Set来作为数据载体。

* 出于状态管理等需要Dorado7会动态代理（Cglib或Javassist）对原始的数据载体进行一个功能增强，只是大多数情况下开发人员不需要也不会注意到这一点。
* 很大程度简化了Dorado7与用户的业务逻辑层对接的难度。Dorado7在Client端提供了专用的数据载体Entity/EntityList，用法类似于JSON/Array。
* 之所以要在Client端提供了专用的数据载体，是因为Dorado7需要为数据载体添加很多诸如异步数据加载、状态管理、翻页、管理当前记录、数据校验等功能。

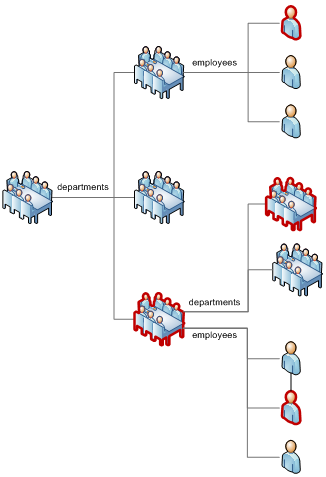
Dorado7的数据集——DataSet：

DataSet主要的生命周期在Client端。主要用途就是为其管理的那堆数据赋予一个ID，以便于数据感知控件能够找到这些数据。另外DataSet不再局限于二维表结构数据，可以支持非常自由的数据形式：

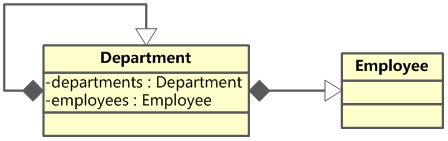
* 数据实体的集合
* 单个的数据实体
* 数据实体构成的树状结构
* 甚至是一个简单的字符串

### 4）稍复杂的数据集场景

人事组织架构的场景：



部门组织数，顶级节点下包含3个子部门，其中第一个子部门下有3名员工，第三个部门下又包含了2个子部门，另外这个部门内还有3名员工。Dorado7用一个DataSet就实现了：



（其中Department拥有一个自关联关系，另外还有一个Employee的聚合关系。）

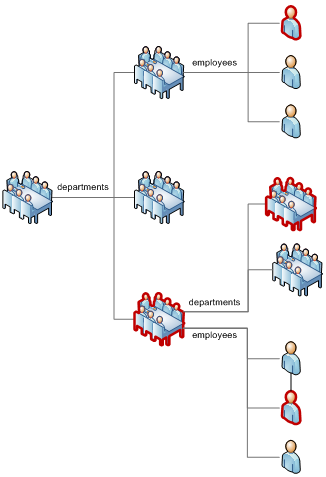
### 5）立体数据模型带来的新问题

问题1：如何对模型中的数据实体进行进一步的描述？

例如上面的部门对象，其中既有子部门又有员工集合，无法简单的按照平面数据模型的方式定义其中各个属性的校验规则、数据类型、显示格式等。

解决办法：引入DataType

问题2：如何实现与数据感知控件的绑定？



如果页面上有一个Grid，用来显示员工信息，但是上图中如何让Grid显示第一个Department下的员工，或者显示第三个部门下的员工。

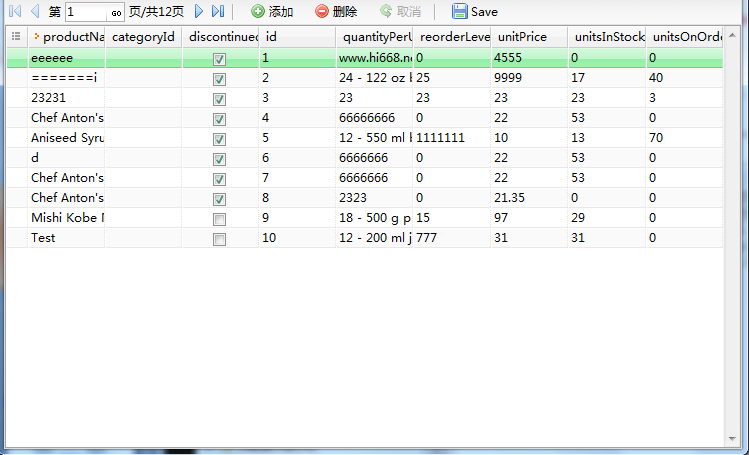
解决：引入DataPath

问题3：如何管理当前记录？

Dorado7中由于采用立体数据结构，如上面的组织结构图中的第三个部门中，相对部门来说当前部门是指红色框中的部门，而当前员工是指下面红框中的员工。一个DataSet拥有多个当前记录，其中的每一个集合都需要管理各自的当前记录。

## 2、实做CRUD

产品列表的维护界面，属于典型的CRUD界面：



* 单击添加按钮新增；
* 单击删除按钮删除当前选中行的记录；
* 单击Grid中的某一行记录，直接在Grid中编辑；
* 编辑完成后通过Save按钮完成记录提交保存的工作；
* 翻页标签，进行数据翻页；

## 3、DataProvider和DataResolver

@DataProvider

public Collection<Product> getAll() {

return productDao.getAll();

}

@DataResolver

@Transactional

public void saveAll(Collection<Product> products) {

productDao.persistEntities(products);

}

@DataProvider和@DataResolver会自动将当前方法对应的服务，根据一系列约定的规则，在运行时注册成全局的DataProvider和DataResolver放在DataProviderManager的对象中。

通常情况下我们所说的DataProvider和DataResolver并不是简单的表示某个具体的方法。而是两种Dorado7中特有的对象。这两个对象一般情况下我们并不需要专门声明，通过annotation定义就会自动的创建，如果我们还希望手工的定义DataProvider或DataResolver。也可以直接利用IDE在.model.xml文件或当前.view.xml的Model节点下定义。

## 4、DataType

### 1）对象结构分

DataType用来描述所有的数据类型，既包括String、int这样的简单类型，也包括数据实体这样的复杂类型。分为以下三类：

* 简单DataType：如String、int、Boolean、Date
* 聚合DataType：如List、Set、Array
* 实体DataType：如POJO Bean、Map

由于Dorado默认已经声明好了常用的简单类型和聚合类型。因此绝大多数情况下，开发需要声明的只是业务系统中的数据实体类型。

### 2）按定义位置分

DataType还可分为全局和私有两种。私有的DataType声明在每一个.view.xml中，另外全局DataType必须定义在.model.xml文件中。

### 3）其他特性

* 支持继承：包括多重继承（即同时继承多个DataType）。迭代式的继承：A继承B，B继承C
* 在View引用DataType时，Dorado首先根据名称查找是否存在匹配的私有DataType，如果没有再继续查找全局DataType。
* [SlEmployee]：表示以集合的方式引用一个名为SlEmployee的DataType。
* 全局DataType与JavaBean的matchType关系，要求一一对应，不允许多个DataType对应同一个JavaBean。这便在于系统知道对象类型，但不知道DataType的时候，系统可以根据前面的对应关系自动的找到相关的DataType，并将数据转换为Dorado中的数据。

## 5、实做主从维护界面

## 6、DataPath

可以认为它是XML中的Xpath技术。主要用于描述如何提取、挖掘立体数据模型中的数据的表达式。

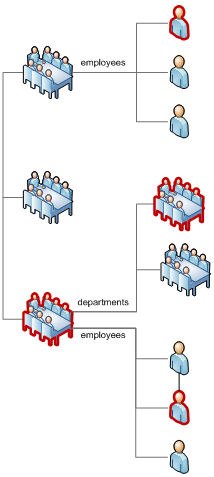
基本语法：

属性名1(参数1,参数2,...)[逻辑表达式1,逻辑表达式2,...].属性名2(参数1,...)[逻辑表达式1,...]...

除基本语法外，DataPath支持用户为期扩展语法——自定义片段。

范例

立体数据模型如下：



### 下面为针对上述立体数据模型定义一些DataPath：

|  |  |
| --- | --- |
| 表达式 | 含义 |
| null | 相当于直接返回被查询的数据，也就是整个集合 |
| \* | 同上 |
| [#current] | 表示顶层集合中的当前Department，也就是第三个Department |
| # | 同上 |
| [#dirty] | 表示顶层集合中所有在客户端被改变过的（包含被删除的）Department |
| #.employees | 表示顶层集合中当前Department中所有的Employee，也就是第三个Department对应的员工列表 |
| #.#employees | 表示顶层集合中当前Department中的当前Employee，也就是第三个Department对应的员工列表中的第二个员工 |
| departments(repeat)或者  departments(R) | 表示所有Department的集合,它表示循环整棵树，将所有的Department作为一个数组返回 |
| .departments(R) | 表示除顶层Department外所有其它Department的集合 |
| #departments(R) | 表示各层当前Department的集合 |
| #departments(leaf) | 表示最末端的当前Department。即通过不断的尝试获取当前Department中的当前Department，直到最末端那个Department |
| departments(R).employees[@.  get("salary")>5000 && @.get("sex")=="male"] | 表示所有Department中的薪水高于5000的男性Employee |
| departments(R).employees.id | 表示返回所有Employee对象的id属性值的集合(DataPath并不总是返回实体对象，如本例是将值的集合返回) |

## 7、数据操作常用技巧

### 1）Server端

EntityUtils工具类和虚拟实体属性

Dorado以数据模型驱动开发，对于数据实体的设计，Dorado在Server端不提供专门的数据实体对象，而是直接由POJO对象或Collection集合定义。

但当我们在Dorado编程的过程中希望给POJO添加一些额外属性，或者还想给它添加一个特别的数据校验器，或者想添加一个状态体现其是新增、修改还是删除的动作。对于这些特性我们都无法直接通过POJO本身解决。为此，Dorado特别提供了一个工具类EntityUtils，用它来实现这种特殊处理，使得我们一个实体Bean具有上述的各种特性。

问题1：如何设置虚拟属性

Category category = new Category();

category = EntityUtils.toEntity(category);

//注意Category.java中没有prop1,prop2,prop3等属性

EntityUtils.setValue(category, "prop1", "value1");

EntityUtils.setValue(category, "prop2", 3);

EntityUtils.setValue(category, "prop3", true);

String prop1 = EntityUtils.getString(category, "prop1");

int prop2 = EntityUtils.getInt(category, "prop2");

boolean prop3 = EntityUtils.getBoolean(category, "prop3");

问题2：如何存取数据实体的状态

由于采用数据模型开发机制，且很多情况下客户端会对数据实体做各种复杂的增删改的多次操作，但是当我们把这个数据实体交付给业务逻辑层的时候我们需要知道这个数据实体的状态，可以通过如下方式进行状态管理：

//该代码

Category category = new Category();

category = EntityUtils.toEntity(category);

EntityUtils.setState(category, EntityState.MODIFIED);

if (EntityState.NEW==EntityUtils.getState(category)){

//TODO

}

以上代码中toEntity的代码并不一定需要使用，视场景而定，例如一般来说如果我们在Resolver中获取到Dorado提交上来的数据，dorado会自动帮我们做好转换的工作。

|  |  |
| --- | --- |
| 数据实体的状态 | 含义 |
| EntityState.NONE | 原始状态 |
| EntityState.NEW | 新增数据实体 |
| EntityState.MOVED | 被移动过的数据实体，专指树控件节点移动引起 |
| EntityState.MODIFIED | 被编辑过的数据实体 |
| EntityState.DELETED | 做过删除标记的数据实体 |

问题3：如何批量的获取所有被删除，或被修改的记录

public void saveAll(Collection<Category> categories) {

for (Category category : categories) {

EntityState state = categoryDao.persistEntity(category);

if (EntityState.isVisible(state)) {

Collection<Product> products = category.getProducts();

for (Product product:EntityUtils.getIterable(products, FilterType.MODIFIED, Product.class)){

//TODO

}

for (Product product:EntityUtils.getIterable(products, FilterType.DELETED, Product.class)){

//TODO

}

for (Product product:EntityUtils.getIterable(products, FilterType.NEW, Product.class)){

//TODO

}

}

}

}

问题4：某个数据实体在客户端被编辑过，如何获取原始值

例如产品中的productName属性从"Aniseed Syrup"改为"Chef Anton's Gumbo Mix"，unitPrice从3600.00调整为3200.00，而提交到服务器端后，我们获得的是Product的实体对象，通过它的getProductName()方法只能拿到"Chef Anton's Gumbo Mix"，如果我们还希望取得该属性的原始值，方法为：

public void saveAll(Collection<Category> categories) {

for (Category category : categories) {

EntityState state = categoryDao.persistEntity(category);

if (EntityState.isVisible(state)) {

Collection<Product> products = category.getProducts();

if (products != null) {

for (Product product : products) {

if (EntityState.MODIFIED.equals(EntityUtils

.getState(product))) {

String oldString =

EntityUtils.getOldString(product, "productName");

//获取原始的productName值

float unitPrice = EntityUtils.getOldFloat(product, "unitPrice");//获取原始的unitPrice值

//TODO

}

}

productService.saveAll(products);

}

}

}

}

### 2）Client端

Dorado在客户端使用的数据载体，除了JSON和Array之外，也提供了Dorado7中专用的数据载体Entity/EntityList。简单说Entity/EntityList是JSON和Array的一个封装，主要提供异步数据装载、状态管理、翻页、管理当前记录、数据校验等功能。

Entity

Entity对象在一般情况下都是由DataProvider自动创建的，手动创建（在设计好DataType之后）方式如下：

var employee = {

$dataType : "Employee",

id : "0001",

name : "John",

sex : true

}

与之前控件的JSON创建方式非常类似，就是一个标准的JSON声明，另外需要通过$dataType指定DataType的类型。创建好的Entity对象就有一些特性：

* 状态管理

employee.setState(dorado.Entity.STATE\_DELETED);

* 虚拟属性和oldValue的访问

employee.set("name", "ANLIN");

alert(employee.getOldData().name);//oldData()方法返回数据实体内部用于保存原有属性值的JSON对象

alert(employee.get("sex"));

我们知道现在Dorado7中的数据载体支持立体数据模型，JSON可以包含一个立体数据模型的数据，通过DataType我们可以完成立体数据的描述，但是怎么通过Entity对象将相关的数据加入呢，这种Java对象间的应用关系，我们就可以通过两个关键方法createBrother()和createChild()，以树为例：createBrother是用于创建当前节点的平行节点，而createChild是用于创建当前节点的子节点。

getMessages ( String [property] ) 和 setMessages( String [property],

String|Object|[String]|[Object] [messages] )

返回或设置当前数据实体关联的额外信息的数组。

在一般情况下这里的信息都对应了数据校验的验证结果。返回的对象是一个JSON数组，JSON结构说明：

* state {String} 信息级别。取值范围包括：info、ok、warn、error。默认值为error。
* text {String} 信息内容。

由于一个PropertyDef可能会添加多个数据校验器，数据校验器的验证结果都可以通过getMessages(String property)进行存取。因此这个方法返回的是一个JSON数组。

如果不指定property参数，返回的JSON数组表示当前实体对象的信息集合，如果指定property属性，则表示指定属性的信息集合。如指定属性的数据校验结果。

entity.setMessages("desc",{state:"info", text:"这是备注信息的提示信息

"});

entity.setMessages("salary",{state:"warn", text:"薪水必须大于上海市最低薪资"});

entity.setMessages("sex",{state:"error", text:"性别字段非空"});

以上三行代码会使相应编辑框上出现不同的编辑提示框，提示信息为text中的内容。entity.setMessages(“A”)或entity.setMessages(“A”,undefined)表示清空掉验证状态。

EntityList

向集合中插入一个实体数据。

entitylist.insert({

id : "0001",

name : "John",

sex : true

},"begin");

insertMode参数为可选项，有四种取值：

|  |  |
| --- | --- |
| 模式 | 含义 |
| begin | 在集合的起始位置插入 |
| before | 在refEntity参数指定的数据实体之前插入 |
| after | 在refEntity参数指定的数据实体之后插入 |
| end | 在集合的末尾插入 |

如果不指定insertMode，系统默认为end。

从集合中删除一个数据实体。

entitylist.remove();//对当前的实体对象添加删除标记

entitylist.remove(null, true);//彻底删除当前的实体对象

true用于控制是否做彻底删除，如果做了彻底删除，则你无法再通过JS访问到它，同理你也无法在DataResolver提交到服务器端的时候取到这个对象。

遍历当前集合的所有实体对象。

// 将每一个集合元素的name属性连接成为一个字符串

var names = "";

var entityList = ...

entityList.each(function(entity){

names += data.get("name");

});

要点：

* 已经打上删除标记的实体对象使用each遍历的时候也能访问到；
* 如果EntityList是一个支持分页的集合，则会遍历所有已经加载过的分页中的实体对象，注意这儿说的是已经加载过的，未加载过的分页不会去遍历。如果想遍历未加载数据的分页建议采用iterator()方法。

each是EntityList提供的一个简单易用的遍历函数，如果希望对遍历规则做更多的控制，我们就需要通过iterator方法，举例:

问题一:不希望对已经做过删除标记的数据进行遍历

解决的办法代码如：

// 将每一个集合元素的name属性连接成为一个字符串

var names = "";

var entityList = ...

var iterator = entityList.iterator(false);

iterator.first();

while(iterator.hasNext()){

names += iterator.next().get("name");

}

问题二：采用了分页技术，但是希望对未加载的页也进行数据遍历。

// 将每一个集合元素的name属性连接成为一个字符串

var names = "";

var entityList = ...

var iterator = entityList.iterator({includeUnloadPage:true});

iterator.first();

while(iterator.hasNext()){

names += iterator.next().get("name");

}

includeUnloadPage会触发DataProvider处理机制引起AJAX数据请求，使用时要注意性能问题。

问题三：希望从指定位置开始进行遍历

// 将每一个集合元素的name属性连接成为一个字符串

var names = "";

var entityList = ...

var iterator = entityList.iterator({nextIndex:20});

iterator.first();

while(iterator.hasNext()){

names += iterator.next().get("name");

}

问题四：希望从指定页开始遍历

// 将每一个集合元素的name属性连接成为一个字符串

var names = "";

var entityList = ...

var iterator = entityList.iterator({pageNo:5});

iterator.first();

while(iterator.hasNext()){

names += iterator.next().get("name");

}

问题五：只遍历当前页

// 将每一个集合元素的name属性连接成为一个字符串

var names = "";

var entityList = ...

var iterator = entityList.iterator({currentPage:true});

iterator.first();

while(iterator.hasNext()){

names += iterator.next().get("name");

}

### 3）DataSet.getData和DataSet.queryData

getData默认只返回DataPath命中的第一笔数据，可能是Entity，也可能是EntityList。

queryData返回DataPath完整的执行结果，且默认只返回Entity（只在使用#等确定只可能有最多一个命中数据实体的DataPath时）或Entity的数组。

getDate范例1：

//获取产品分类为2的所有产品

var entityList = ds.getData(“[@.get(‘categoryId’)==2]”);

范例2：参数：[path]、[options]

//清除DataSet中原有的数据并重新提取数据后，再查询产品分类为2的所有产品

var entityList = ds.getData(“[@.get(‘categoryId’)==2]”,{flush:true});

范例3：

//在已经加载到客户端的数据中查询产品分类为2的所有产品

var entityList = ds.getData(“[@.get(‘categoryId’)==2]”,{loadMode:”never”});

queryData范例1：

//获取产品分类为2的所有产品

var entityArr = ds.queryData(“[@.get(‘categoryId’)==2]”);

范例2：

//清除DataSet中原有的数据并重新提取数据后，再查询产品分类2的所有产品

var entityArr = ds.queryData(“[@.get(‘categoryId’)==2]”,{flush:true});

范例3：

//在已加载到客户端的数据中查询产品分类为2的所有产品

var entityArr = ds.queryData(“[@.get(‘categoryId’)==2]”,{loadMode:”never”});