**数据统计管理文档**

此项目主要是用来理清etl依赖关系，主要是从5张表里面进行数据的提取和处理。项目功能主要分两块：表层级详细和表层级关系。

主要依靠图这个数据结构来对各个节点(表)及其依赖关系进行存储。NubiaetlGraphHandler：用来处理5层，生成图的类。主要的函数是init()生成图。过程是这样的：分别遍历这5张表，然后遍历每张表的每一行的每一个sql。发现数据库名和表名，然后为每个table来new一个node出来，并添加进图，然后根据from来进行依赖关系的发现，然后添加到图的，以边的形式来展示。

表层级详细主要是靠NubiaetlGraphHandler来生成图，然后把相关数据传到前端，进行显示。

表层级关系ArborjsGraphController前端主要依靠Arbor 这个插件来进行显示<http://arborjs.org/halfviz/#/a-new-hope>

相关的git：<https://github.com/samizdatco/arbor>

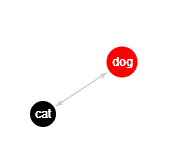
通过这种格式来进行展示

101->100

100->101

100 {color:red,shape:dot,label:dog}

101 {color:black,shape:dot,label:cat}



分为其他四层，因为报表层太多了，把报表层进行一个区分。

对于其他四层基础层，每深一层都要显示之前那一层。就是显示是递增的。

细微的技术点：

public List<Object[]> reader(String tableName) {  
 List<Object[]> result = (List<Object[]>) dao.execute(new SessionCallback() {  
 @Override  
 public Object doInSession(Session session) {  
 SQLQuery sqlquery = session.createSQLQuery("SELECT \* FROM " + tableName);  
 List<Object[]> result = sqlquery.addScalar("title", StandardBasicTypes.*STRING*)  
 .addScalar("description", StandardBasicTypes.*TEXT*)  
 .addScalar("template", StandardBasicTypes.*TEXT*).list();  
 return result;  
 }  
 });

1. 通过execute(new SessionCallback(){});重写doInSession来执行我自己的特定方法。

sess.createSQLQuery("SELECT ID, NAME, BIRTHDATE FROM CATS").list();

它们都将返回一个Object数组(Object[])组成的List，数组每个元素都是CATS表的一个字段值。Hibernate会使用ResultSetMetadata来判定返回的标量值的实际顺序和类型。

如果要避免过多的使用ResultSetMetadata,或者只是为了更加明确的指名返回值，可以使用addScalar()。

sess.createSQLQuery("SELECT \* FROM CATS").addScalar("ID", Hibernate.LONG).addScalar("NAME", Hibernate.STRING).addScalar("BIRTHDATE", Hibernate.DATE)

这个查询指定了:

* SQL查询字符串
* 要返回的字段和类型

它仍然会返回Object数组,但是此时不再使用ResultSetMetdata,而是明确的将ID,NAME和BIRTHDATE按照Long,String和Short类型从resultset中取出。同时，也指明了就算query是使用\*来查询的，可能获得超过列出的这三个字段，也仅仅会返回这三个字段。

详解请看<http://www.cnblogs.com/biGpython/archive/2012/03/26/2417926.html>

1. Collections.unmodifiableSet()方法返回指定set的不可修改视图。

注册注解处理器

• 方式一:bean

<bean class="org.springframework.beans.factory.annotation.

AutowiredAnnotationBeanPostProcessor"/>

• 方式二: 命名空间<context:annotation-config />

<context:annotationconfig /> 将隐式地向Spring 容器注册AutowiredAnnotationBeanPostProcessor、CommonAnnotationBeanPostProcessor 、 PersistenceAnnotationBeanPostProcessor 以及RequiredAnnotationBeanPostProcessor 这4 个BeanPostProcessor 。

• 方式三: 命名空间<context:component-scan />

如果要使注解工作，则必须配置component-scan ，实际上不需要再配置annotation-config。

base-package 属性指定了需要扫描的类包，类包及其递归子包中所有的类都会被处理。还允许定义过滤器将基包下的某些类纳入或排除。

开启mvc注解然后分别扫描到DAO、service、controller层。

<mvc:annotation-driven/>  
<context:component-scan base-package="controller" />。。。。