如何创建知识库对象

1. 说明

新增加的扩展插件中如果需要往知识库中增加库表时，可以使用Smartbi所包含的Hibernate机制运行。

2. 操作步骤

首先使用升级类自动创建库表，然后按以下步骤进行之后就可以直接在module中操作创建的库表了：

1、增加一个与数据库表映射的POJO实体类

（1）在实体类中添加Annotation：

```
/*
 *
 */
@javax.persistence.Entity
@javax.persistence.Table(name = "t_user")
@org.hibernate.annotations.Cache(usage = org.hibernate.annotations.CacheConcurrencyStrategy.READ_WRITE,
region = "POJO")
```

⚠ 注意：这三行声明该类是一个知识库映射类表名是t_user。在Smartbi中，建议表名以【模块名缩写+下划线开头，即：t_、t_ap_、tr_】等。

在POJO类中，Annotation应该标注在类名定义、getter方法上，而不要直接定义在属性上。常用的Annotation有：

名称	说明	备注
@Entity	本类为POJO，与数据库中的表映射	
@Table	属性name 声明数据库中的表名	在BOF框架中，表名、字段名应该为全小写； 表名格式为【t + 项目类型 + _ + 表名】，如 to_cube，tr_report。
@IdClass	指定使用复合主键，主键类使用	指定复合主键对应的类，主键类的属性名称与主类的属性名称全部相同，主类中属性均声明@Id。具体请参看UserConfig等实现。
@Embeddable	配合@IdClass指定本类为主键类	
@NamedQueries	声明多个预编译的HQL语句	
@NamedQuery	声明预编译HQL语句	name语句名称 query为HQL语法的查询，可使用参数，允许跨表查询。
@Cache	使用缓存提高访问效率	策略应为READ_WRITE，区域为POJO。
@Id	声明属性为主键	
@Column	属性name声明表中字段名称	字段名应为c_开头，字段名必须全小写。
@Type	属性type声明字段类型	默认不需要增加此标注，只在以下两种情况需要： • 当字段类型为CLOB时，将此字段声明了类型 smartbi.repository.StringClobType以修正Oracle中CLOB操作可能出现的问题； • 当字段类型为Varchar并且需要将此字段映射成Java中的枚举类型时 @Type(type = "org.hibernate.type.EnumType", parameters = { @Parameter(name = org.hibernate.type.EnumType.ENUM, value = "smartbi.managereport.util.RuleType"), @Parameter(name = org.hibernate.type.EnumType.TYPE, value = Types.VARCHAR + "") }) 格式，RuleType为枚举类型的实现类。

@Transient	声明没有映射知识库字段	
@ManyToOne	声明多对一关系，需配合@JoinColumn使用	fetch一般为LAZY。
@JoinColumn	声明多对一中的字段名称	指定【多】这一方中的字段名称，对应【一】的主键值。
@OneToMany	声明一对多关系，通常由mappedBy指定由【多】方控制	方法声明应该使用泛型：public List<ResourceTreeNode> getChildNodes() mappedBy：指定对应【多】方中属性名称，注意这里是由对象getter来判断而不是类的成员变量。参考HQL语句中的getter/setter说明。
@OrderBy	指定一对多关系中，返回的列表排序方式	
@ManyToMany	指定多对多关系，需配合@JoinTable	一般声明fetch为LAZY，cascade为ALL，其中一方增加@JoinTable，另一方由mappedBy指定第一方的属性名称。
@JoinTable	指定多对多关系中使用到的中间表	name指定中间表名 joinColumns指定本方@JoinColumn inverseJoinColumns对方@JoinColumn。
@JoinColumn	配合@JoinTable多对多关系	name指定本表中的字段名 referencedColumnName中间表字段名。

（2）映射中中还可以使用NamedQuery注解声明HQL查询语句：

```
@NamedQueries({
    @NamedQuery(name = "User.getByName", query = "from User u where u.name = ?"),
    @NamedQuery(name = "User.getByAlias", query = "from User u where u.alias = ?"),
    @NamedQuery(name = "User.likeName", query = "from User u where (u.name like ? or u.alias like ?) and u.
id <> ? and u.id <> ? ")
})
```

（3）每个getter方法前声明Column注解，字段名建议使用c_开头。

（4）当使用CLOB类型时，应该声明@Type(type = "smartbi.repository.StringClobType")，以便修正Oracle中CLOB操作可能出现的问题。

（5）当需要使用多对一、一对多、多对多映射时，需要额外添加@OneToMany(fetch = FetchType.LAZY, mappedBy = "user")和@Cache(usage = CacheConcurrencyStrategy.READ_WRITE, region = "POJO")和@JoinTable这样的注解。

（6）当某些getter方法不需要作为映射到知识库字段时，需要添加@Transient注解。

2、在组件的激活方法public void activate()中调用daoModule.addPOJOClass(class)方法注册新增的实体类。

```
public void activate() {
    daoModule.addPOJOClass(CreatingKnowledgeBaseObjects_A.class);
    daoModule.addPOJOClass(CreatingKnowledgeBaseObjects_B.class);
}
```

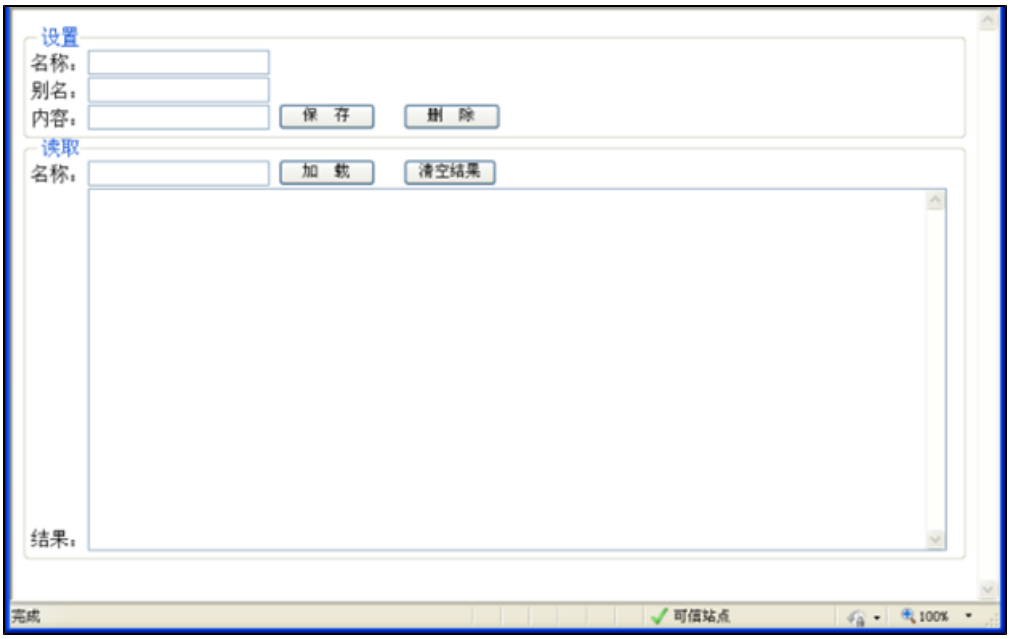
3、建议新增一个继承 smartbi.repository.AbstractDAO<T, PK>的类（其中T为实体类、PK为实体类的主键），例如public class ResourceTreeNodeDAO extends AbstractDAO<ResourceTreeNode, String>，这样可以简化加载、更新、删除、查询实体类的操作。

extensions.list | CreatingKnowledgeBaseObjects_Module.java | CreatingKnowledgeBaseObjects_A_DAO.java

```java
package bof.ext.CreatingKnowledgeBaseObjects.repository;

import smartbi.repository.AbstractDAO;


public class CreatingKnowledgeBaseObjects_A_DAO extends AbstractDAO<CreatingKnowledgeBaseObjects_A, String> {
    CreatingKnowledgeBaseObjects_A_DAO() {
        super(CreatingKnowledgeBaseObjects_Module.getInstance().getDaoModule());
    }

    public CreatingKnowledgeBaseObjects_A getByName(String name) {
        return findByNamedQueryUnique("CreatingKnowledgeBaseObjects_A.getByName", name);
    }

    public CreatingKnowledgeBaseObjects_A getByAlias(String alias) {
        return findByNamedQueryUnique("CreatingKnowledgeBaseObjects_A.getByAlias", alias);
    }
}

```

3. 示例说明

在升级类"UpgradeTask_New.java"中创建了两个知识库表，分别与"CreatingKnowledgeBaseObjects_A.java"和"CreatingKnowledgeBaseObjects_B.java"中的描述相对应，"CreatingKnowledgeBaseObjects_A.java" 和 "CreatingKnowledgeBaseObjects_B.java"是一对多的关系。另外"CreatingKnowledgeBaseObjects_Module.java" 中定义了3个测试方法"setTestInfo、getTestInfo、deleteTestInfo"，用于往上述两个知识库表中"添加、读取、删除"数据，测试链接为http://localhost:18080/smartbi/vision/test/CreatingKnowledgeBaseObjects.htm，测试界面1如下：



示例代码下载：CreatingKnowledgeBaseObjects.rar