



国际物流云商系统第二天

一. 回顾

1. 学习项目的方法和心态
2. 项目背景
3. 项目需求的获取（界面原型法）
4. 使用 PD 实现 UML 的 UseCase 图（用例图描述需求）
5. 系统框架的搭建

二. 数据库设计

1.数据库建模

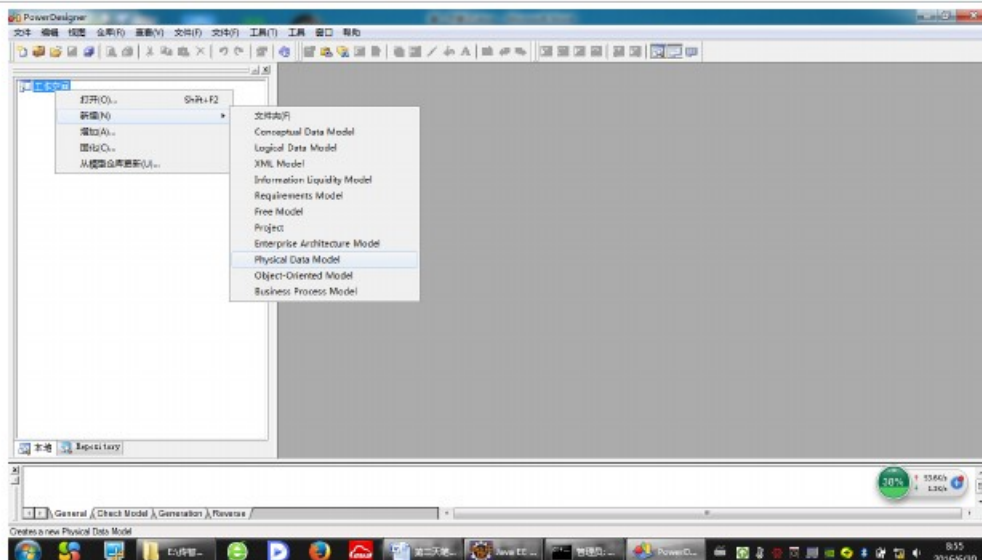
建立数据库的模型（实体与实体之间的关系在数据库中如何表现，体现到关系型数据库中就是主外键的关系）

数据库建模的工具：PD

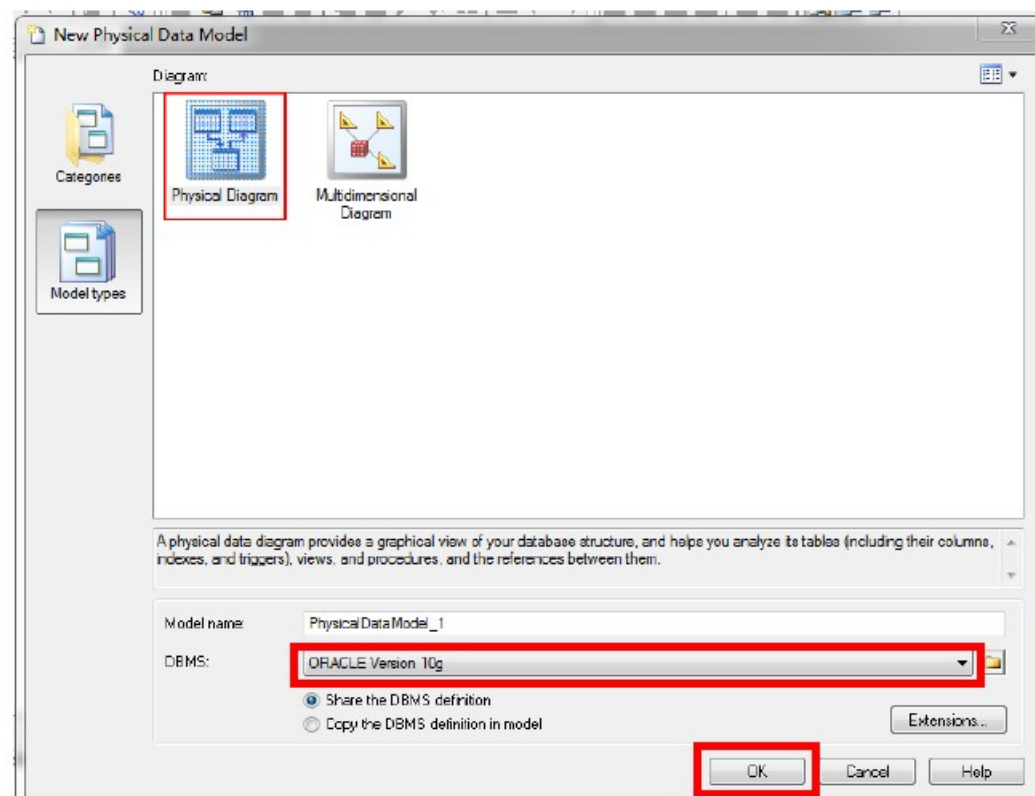
如何使用 PD 来实现数据库建模？

它的优势在于，不用再使用 `create table` 去创建表结构，它会让数据库设计人员只关注如何来进行数据库建模，将来的建表语句，可以自动生成。

1. 进入 PD，并打开如下窗口：



2. 选择物理数据模型 (Physical Data Model)



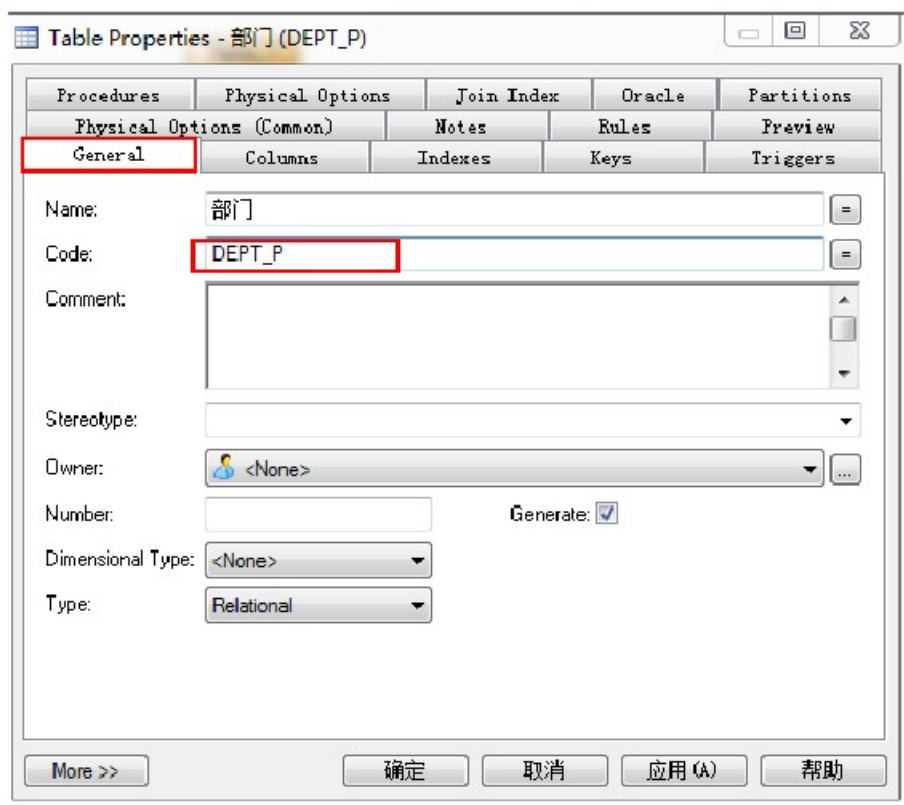
3.点 OK 后，进入



4.进行数据库设计时，原则：

表名：一定使用汉字对应的英文，一般根据业务需要在表名后面加上:_业务后缀。

如，部门表，写成 DEPT_P(表名全大写)



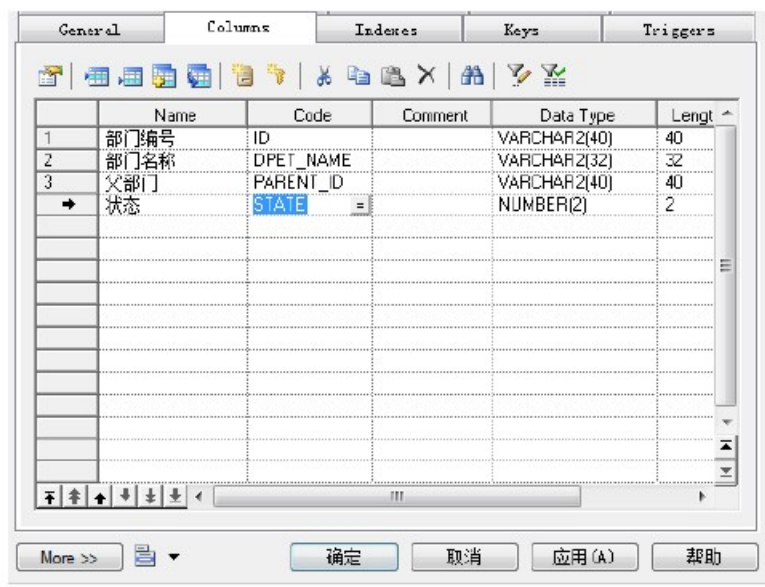
列的确定：与当前对象相关的属性并且与本次系统开发也相关的属性


比如：部门有部门编号，部门名称，父部门的编号，状态

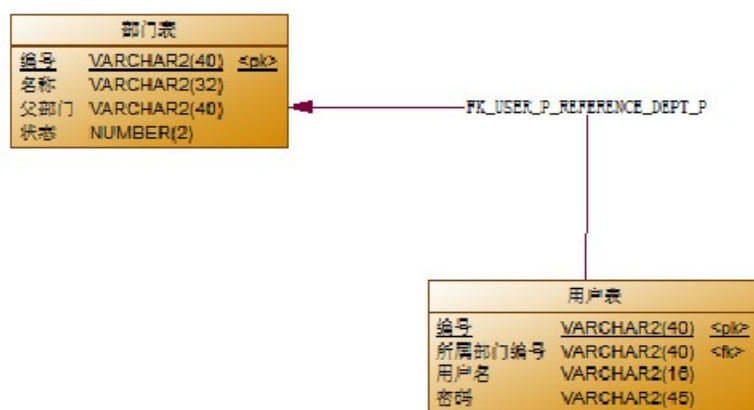
列名的确定：ID, DEPT_NAME,PARENT_ID,STATE

列的数据类型：varchar2 varchar2 varchar2, NUMBER

列的长度：50 32 50 2

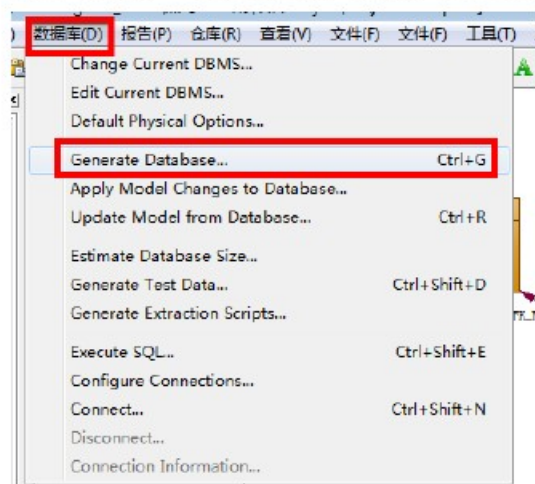


外键的创建，，如果主键直接为 ID 会出现问题，所以一般会在主键前面加上表名作为前缀。

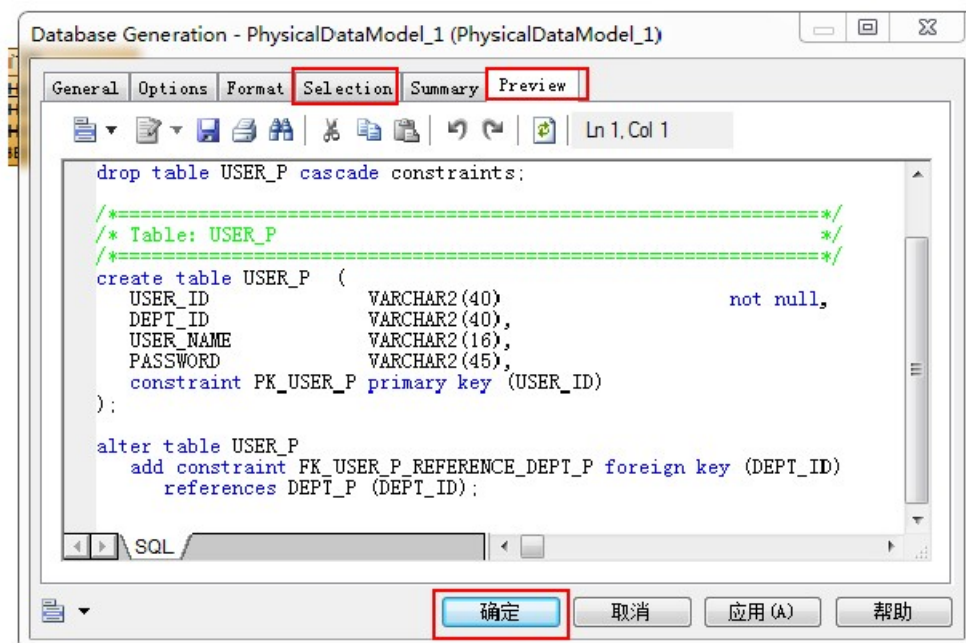


表结构的获取：

菜单“数据库”---生成数据库表结构（Generator DataBase）



得到下面的界面



项目总体数据库设计图如下

2.ORACLE 数据库的回顾

数据库-----实例-----表空间-----用户-----授权

作为一个新项目，所使用的表结构应当放在一个指定的表空间中。最好就是一个项目对应一个表空间。

创建表空间：

```
create tablespace itheima
datafile 'D:\oracletablespace\itheima.dbf'
size 50M
autoextend on
next 10M
```

创建用户：

```
create user itheima
identified by orcl
default tablespace itheima
```

授权：

```
grant dba to itheima
```


grant connect,resource to itheima

三. 运行流程分析

1.index.jsp 页面的加载过程

The screenshot illustrates the initial loading of the web application. Key components include:

- JavaScript:** A script that sets `window.location.href = 'login';` to redirect the user.
- Struts Configuration:**
 - `<action name="Login" method="login" class="loginAction">` defines the login process.
 - `<result name="success">/WEB-INF/pages/home/fmain.jsp</result>` specifies the page to load after a successful login.
 - `<bean id="loginAction" class="cn.itcast.jk.action.LoginAction" scope="prototype">` defines the `loginAction` bean.
- Frameset:** A `<frameset>` tag that defines the layout with frames like `top_frame`, `leftFrame`, and `main`.

2.顶部菜单点击后，左侧及中间区域的页面会发生改变？

This screenshot details the dynamic page loading mechanism triggered by a menu click:

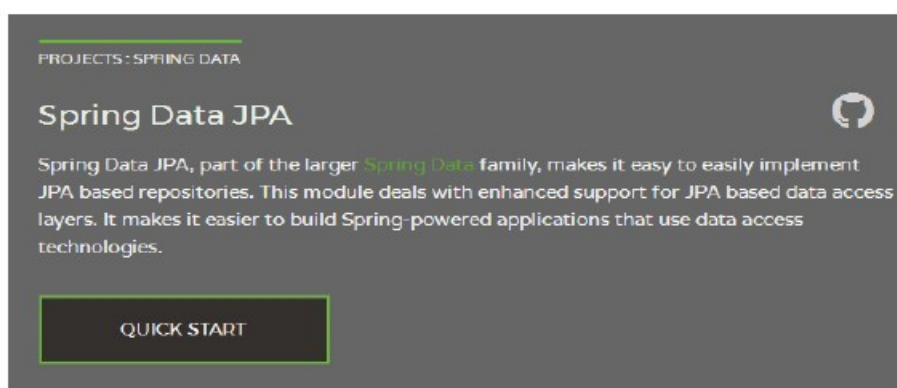
- Menu Click:** A click on the `sysadmin` menu item triggers the `toModule` function.
- JavaScript Logic:** The `toModule(moduleName)` function updates the `location.href` of the `top.leftFrame` and the `main` frame to load the appropriate content.
- Struts Result:** The `<result name="tomain">/WEB-INF/pages/${moduleName}/main.jsp</result>` configuration maps the `tomain` action result to the `main.jsp` page for the selected module.

3.左侧菜单点击后，中间区域显示结果

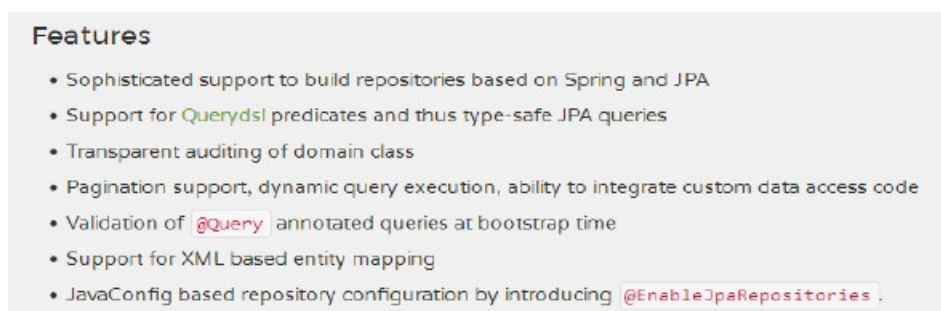
```
'target="main" id="aa_1">部门管理</a></li>
```

四. 使用 Spring Data JPA 实现 DAO 层的构建

1.spring data JPA 是什么

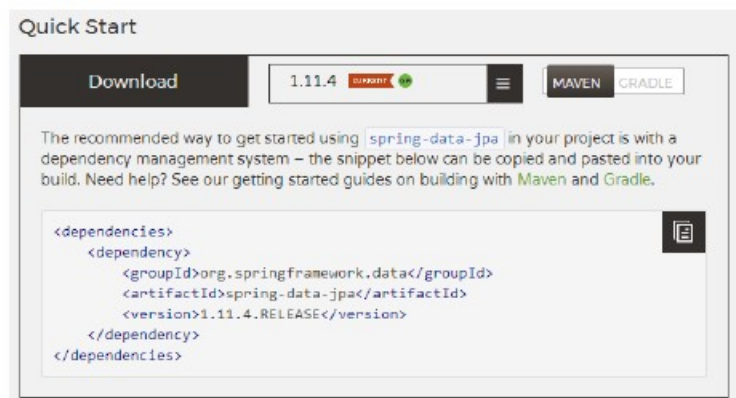


2. Spring data JPA 的特征





3.Spring Data JPA 的坐标引入



4.Spring Data JPA 的配置

- applicationContext.xml 中配置 Spring Data JPA 的 entityManageFactory

```
<bean id="entityManagerFactory"
      class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="dataSource" ref="dataSource"></property>
  <property name="packagesToScan" value="cn.itcast.domain"></property>
  <property name="persistenceProvider">
    <bean class="org.hibernate.jpa.HibernatePersistenceProvider"></bean>
  </property>
  <!-- JPA提供商适配器 -->
  <property name="jpaVendorAdapter">
    <bean
      class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
      <property name="generateDdl" value="false"></property>
      <property name="database" value="ORACLE"></property>
      <property name="databasePlatform"
        value="org.hibernate.dialect.Oracle10gDialect"></property>
      <property name="showSql" value="true"></property>
    </bean>
  </property>
  <property name="jpaDialect">
    <bean
      class="org.springframework.orm.jpa.vendor.HibernateJpaDialect"></bean>
  </property>
  <property name="jpaPropertyMap">
    <map>
      <entry key="hibernate.query.substitutions" value="true 1, false 0" />
      <entry key="hibernate.default_batch_fetch_size" value="16" />
    </map>
  </property>
</bean>
```




```

        <entry key="hibernate.max_fetch_depth" value="2" />
        <entry key="hibernate.enable_lazy_load_no_trans"
value="true"></entry>
        <entry key="hibernate.generate_statistics" value="true" />
        <entry key="hibernate.bytecode.use_reflection_optimizer"
        value="true" />
        <entry key="hibernate.cache.use_second_level_cache" value="false" />
        <entry key="hibernate.cache.use_query_cache" value="false" />
    </map>
</property>
</bean>

```

- 添加 spring data jpa 配置所需的约束

```

xmlns:jpa=http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa
http://www.springframework.org/schema/data/jpa/spring-jpa.xsd

```

- 添加 spring data JPA 事务管理器 JPATransactionManager

```

<!-- JPA事务管理器 -->
<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
    <property name="entityManagerFactory" ref="entityManagerFactory"></property>
</bean>

```

- 添加 spring data jpa 的配置

```

<jpa.repositories base-package="cn.itcast.dao"
    entity-manager-factory-ref="entityManagerFactory"
    transaction-manager-ref="transactionManager"></jpa.repositories>

```

base-package 代表扫描的 dao 包

entity-manager-factory-ref 代表所引用的 EntityManagerFactoryBean

transaction-manager-ref 代表所引用的 JPA 事务管理器

5.使用 Spring Data JPA 实现 DAO

根据 spring data jpa 的规范，在编写自定义的 DAO 接口时需要继承 spring data JPA 的 JpaRepository<T, Serializable>。

编写部门 DeptDao 的接口如下：

```

public interface DeptDao extends JpaRepository<Dept, Serializable>{
}

```

这样我们就编写好了 DeptDao 接口，在使用 spring data jpa 时持久层只需要编写接口，不使用有任何的持久层的实现。

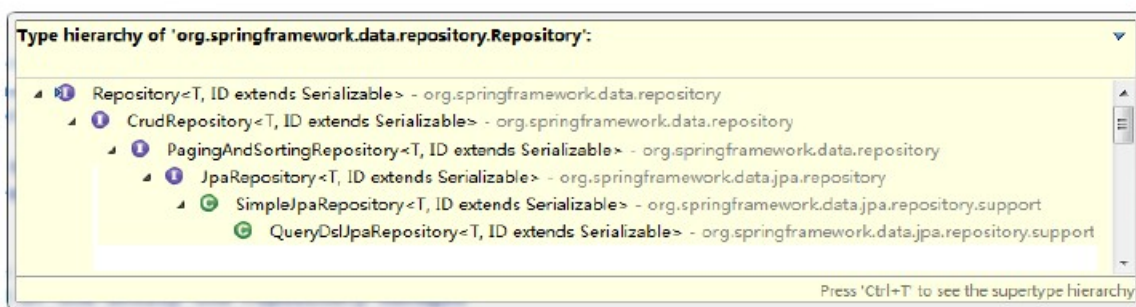


编写 Dept 的业务层，DeptServiceImpl 调用 DeptDao 中的方法。使用单元测试验证 DeptDao 中的方法。可以测试更多的方法

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration("classpath:applicationContext.xml")
public class DeptServiceTest {
    @Autowired
    private DeptService deptService;

    @Test
    public void testQuery(){
        Dept obj = deptService.get("100");
        System.out.println(obj);
    }
}
```

6.spring data jpa 的常用接口分析



在使用 spring data jpa 时，一般我们都是继承 JpaRepository 这个接口，这样我们就继承了父接口中的很多方法，而这些方法只是一些声明，还没有具体实现，那它是如何实现的呢？

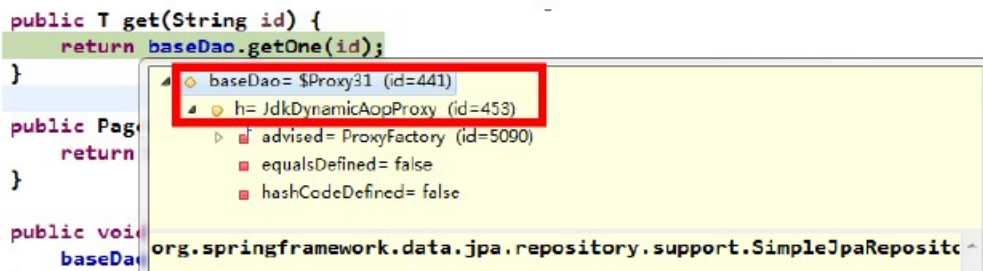
7.源码解析 spring data Jpa 实现过程

SimpleJpaRepository 类的源码分析，我们通过 SimpleJpaRepository 类的分析，可以知道 spring data jpa 也只是对于 JPA 的一层封装，并通过代理方式实现 DAO 的动态生成。

- 生成代理子类的实现过程

JdkDynamicAopProxy.invoke(Object, Method, Object[]) line: 155

通过分析得知，在调用 DeptDao 之前会通过 JdkDynamicAopProxy 类来动态生成 DeptDao 接口的实现类，再通过 Spring 的 IOC 实现的注值，这样就动态生成了一个 DeptDao 的实现对象，这种方式完全是采用 JDK 的动态代理方式来实现的。



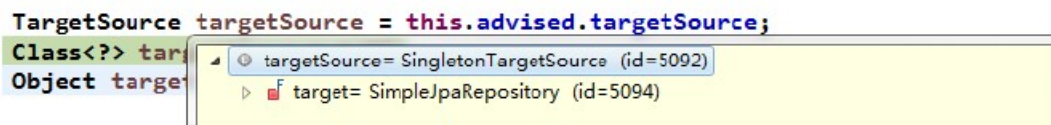
通过注入的值实质是 DeptDao 的对象，而这个对象又是通过 JDK 动态代理生成的。

- 代理类对象中方法的调用分析

当再次调用 baseDao 的 getOne()方法时，其实本质就是调用 DeptDao 的方法，而 DeptDao 只是一个接口，它的实现类也只是动态生成的。

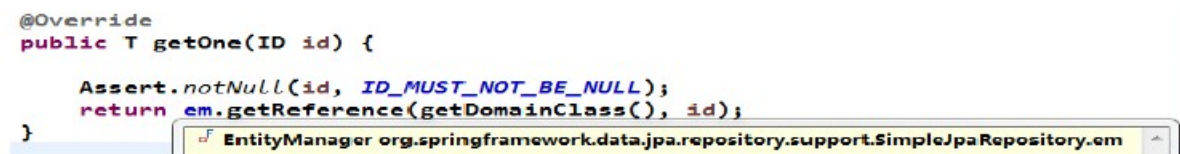
再上一个层次，那么 getOne()方法又是调用谁的？是怎么调用的？

它还是通过 JDK 动态代理实现了 getOne()方法的调用，整个过程中必然会有一个最本质的 Oracle 公司的 JPA 最原生的 JPA 代码，那就是 targetSource 所指向的目标类，真正的 target 是 SimpleJpaRepository 类。



- SimpleJpaRepository 类的分析

通过源码分析，找到了 em 对象，它就是 EntityManager 类型，而它是 JPA 原生的实现方式，所以得到结论 spring data JPA 只是对标准的 JPA 进行了进一步封装，更加了 DAO 的开发，可以看出规范更重要。



五.Spring data jpa 的方法定义

1.使用接口中的方法

在继承的 JpaRepository 接口时，它里面就定义了若干个方法，这样我们就可以直接使用这些方法了，这些方法能满足我们的基本 CRUD 要求。



- ☒ ☒ ^A deleteAllInBatch()
- ☒ ☒ ^A deleteInBatch(Iterable<Dept>)
- ☒ ☒ ^A findAll()
- ☒ ☒ ^A findAll(Iterable<Serializable>)
- ☒ ☒ ^A findAll(Sort)
- ☒ ☒ ^A flush()
- ☒ ☒ ^A getOne(Serializable)
- ☒ ☒ ^A save(Iterable<S>) <S>
- ☒ ☒ ^A saveAndFlush(S) <S>

2.使用方法命名规则进行匹配

使用 Spring data jpa 查询时，只要符合规定就可以直接定义方法，将来 Spring data jpa 会进行方法名的字符串拆解，并自动生成 JPQL 进行查询，这样就可以免去程序员的工作。

//2.使用方法命名规范实现查询

```
public Dept findByDeptName(String deptName);
```

3.使用 JPQL 实现

JPQL: Java Persistence Query Language，基于首次在 EJB2.0 中引入的 EJB 查询语言(EJB QL)，Java 持久化查询语言 (JPQL) 是一种可移植查询语言，旨在以面向对象的方式实现数据操作，可以编译成所有主流数据库服务器上的 SQL。

使用 JPQL 的查询语句来实现各种查询，JPQL 的语法只要我们学过 Hibernate 的 HQL，那么使用 JPQL 的写法几乎一致。

//3.使用JPQL实现

```
@Query(value="from Dept where deptName=?1 and state=?2")  
public Dept findABC(String deptName,Integer s);
```

4.使用 SQL 语句查询

Spring data jpa 同样也支持使用普通 sql 语句进行查询，如下：

//4.使用SQL

```
@Query(value="select * from DEPT_P where dept_name=:deptName and state=:s",nativeQuery=true)  
public List<Object[]> findSQL(@Param("deptName") String deptName,@Param("s") Integer s);
```

六.系统架构技术

在实际项目的框架过程中，架构师通常会通过一些工具类或基础父类的抽取，从而实现项目基础体系的构建，这样构建的目标无非就是达到更加高效的复用。



1. 软件复用技术

- 变量复用
它是最小单元的复用
- 方法复用
它是将一组逻辑单元先封装成方法，再通过频繁调用，从而代码复用
- 模块复用
它是将一些操作抽取成一个模块，在需要的时候就引入进来，直接调用，如我们的 Maven 模块的复用
- 工程复用
在多个系统整合时，经常会调用其它系统的某个部分来完成自己的功能，此时就要实现多个系统之间的功能接口调用，通常我们可以 SOA 技术来实现。

2.BaseAction 的抽取及复用

```

*
* 使用BaseAction的两个理由
* 1.现在编写Action类时，依赖的是自己的API,将来框架升级改造时，只要修改BaseAction就可以了
*
* 2.可以在BaseAction中抽取一些公共的操作方法
*
*/

//通过RequestAware, SessionAware, ApplicationAware实行接口获得request,session,application对象，action中就可直接调用
//背后是servletConfig拦截器实现的

public class BaseAction extends ActionSupport implements RequestAware, SessionAware, ApplicationAware{

```

八. Dept 的 CRUD

1.实现部门的分页查询

```

/**
 * 分页查询部门列表
 */
@Action(value = "deptAction_list", results = {
    @Result(name = "list", location = "/WEB-INF/pages/sysadmin/dept/jDeptList.jsp") })
public String list() throws Exception {
    //1.实现分页查询 只查询当前可用的部门 state=1
    Specification<Dept> spec = new Specification<Dept>() {
        public Predicate toPredicate(Root<Dept> root, CriteriaQuery<?> query, CriteriaBuilder cb) {
            return cb.equal(root.get("state").as(Integer.class), 1);
        }
    };
    //设置Pageable的取值
    Pageable pageable = new PageRequest(page.getPageNo()-1, page.getPageSize());
    org.springframework.data.domain.Page<Dept> spage = deptService.findPage(spec, pageable);

    //实现分页组件的转换
    super.parsePage(page, spage);
    page.setUrl("deptAction_list");

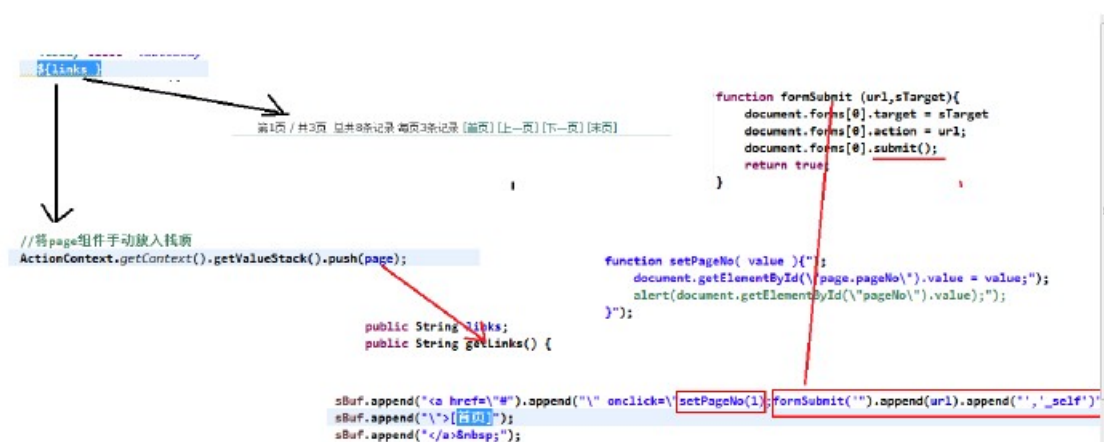
    //page压入到值栈中
    super.push(page);

    return "list";
}

```




分页的实现



九. 实现部门的 CUD 操作(作业)

删除:

1. 递归删除父子部门
2. 删除父项时，子项不删除，因为子项可以进入其它 部门

先更新子项所属的父部门，这样当前这个父部门下就没有子部门，直接删除父部门

3. 结合状态字段，假删除（本质是修改状态字段），递归修改