学号 E21514048 专业 15软件工程 姓名 吕永杰

实验日期 **2018.04.12** 教师签字 成绩

实验报告

【实验名称】 抽象工厂+单例模式

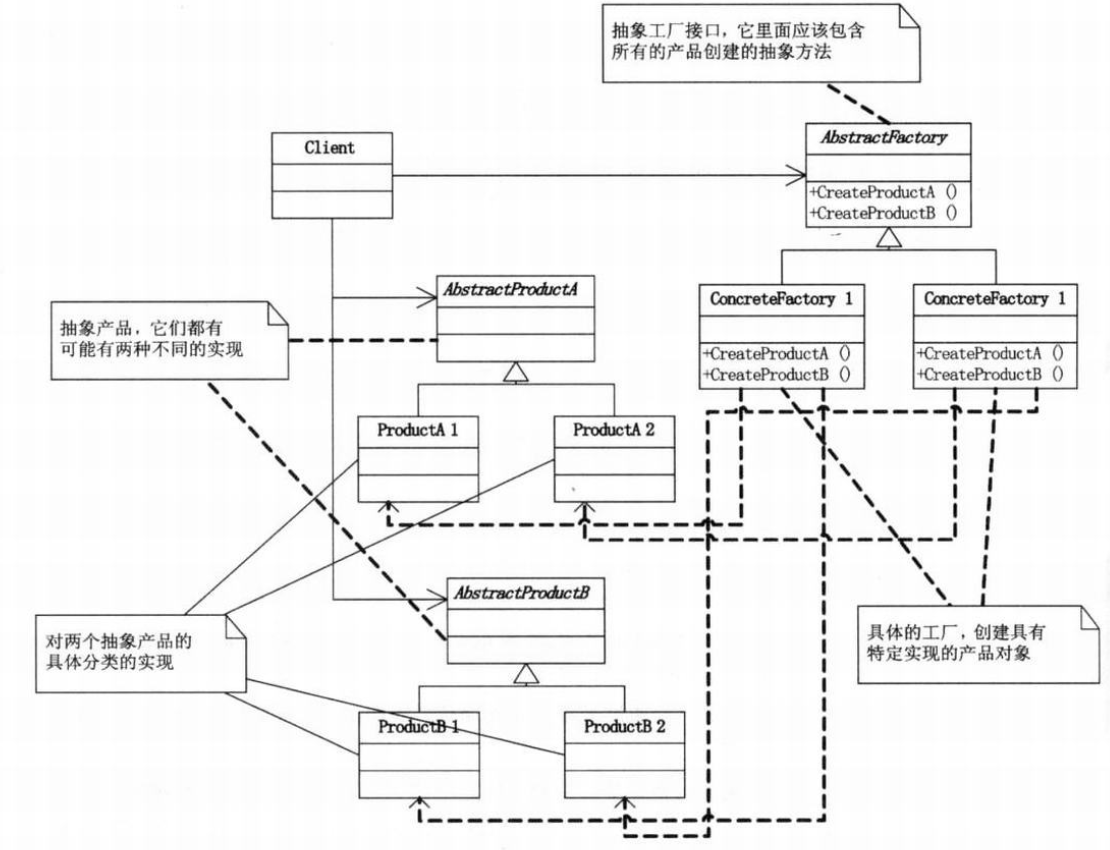
【实验目的】

掌握抽象工厂模式和单例模式的基本使用，优势以及它的原理

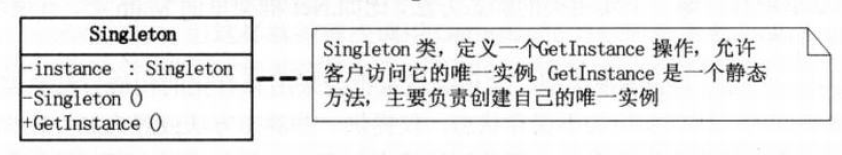
【实验原理】

1.抽象工厂模式提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。

抽象工厂模式UML如下：



2.单件模式UML如下：



将构造方法定义为私有的，阻止了外部程序实例化该类，但可以在该类内部写一个静态的public方法GetInstance()，这个方法的目的就是返回一个类实例，并在此方法中，去做是否有实例化的判断。

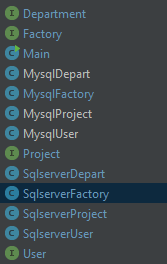
【实验内容】

问题：该公司数据库拥有三张表，分别是用户表、部门表和项目表，每张表的操作都支持查询和添加功能。数据库支持mysql和sqlserver两种。结合抽象工厂模式和单件模式给出该系统的模拟代码。

以User为例，画出下面的UML图



代码中的所有类：



详细的代码见最后的附录。

【小结或讨论】

通过本次实验，对抽象工厂模式和单件模式有了更深刻的理解。在本次实验中，抽象工厂模式将数据库的选择交给了客户端，通过多抽象工厂接口的实现来建立具体的工厂（如SqlserverFactory），具体的工厂去创建对应的产品（如SqlserverUser），然后对具体的产品进行一系列的操作，这就是我对抽象工厂的理解。在本次实验中还用到了单例模式，主要用在具体的工厂上面，确保程序中具体的工厂实例只有一个。

**附代码：**

**Factory:**

*/\*\*  
 \** ***@Description:*** *抽象工厂接口，包含创建各种表的方法  
 \** ***@Date:*** *Create in 19:02 2018/4/12  
 \*/*public interface Factory {  
 User createUser();  
 Department createDepartment();  
 Project createProject();  
}

**SqlserverFactory:**

*/\*\*  
 \** ***@Description:*** *\** ***@Date:*** *Create in 19:04 2018/4/12  
 \*/*public class SqlserverFactory implements Factory {  
 @Override  
 public User createUser() {  
 return new SqlserverUser();  
 }  
  
 @Override  
 public Department createDepartment() {  
 return new SqlserverDepart();  
 }  
  
 @Override  
 public Project createProject() {  
 return new SqlserverProject();  
 }  
  
 // 单例模式  
 private static SqlserverFactory *instance*=null;  
 public static SqlserverFactory getInstance(){  
 if(*instance*==null){  
 *instance*=new SqlserverFactory();  
 System.*out*.println("SqlserverFactory实例化成功！");  
 }else{  
 System.*out*.println("SqlserverFactory实例已经存在！");  
 }  
 return *instance*;  
 }  
 private SqlserverFactory(){}  
}

**MysqlFactory:**

*/\*\*  
 \** ***@Description:*** *\** ***@Date:*** *Create in 19:13 2018/4/12  
 \*/*public class MysqlFactory implements Factory {  
 @Override  
 public User createUser() {  
 return new MysqlUser();  
 }  
  
 @Override  
 public Department createDepartment() {  
 return new MysqlDepart();  
 }  
  
 @Override  
 public Project createProject() {  
 return new MysqlProject();  
 }  
  
 // 单例模式  
 private static MysqlFactory *instance*=null;  
 public static MysqlFactory getInstance(){  
 if(*instance*==null){  
 *instance*=new MysqlFactory();  
 System.*out*.println("MysqlFactory实例化成功！");  
 }else{  
 System.*out*.println("MysqlFactory实例已经存在！");  
 }  
 return *instance*;  
 }  
 private MysqlFactory(){}  
}

**Department：**

*/\*\*  
 \** ***@Description:*** *部门接口，包含对部门表的操作方法  
 \** ***@Date:*** *Create in 19:06 2018/4/12  
 \*/*public interface Department {  
 void insert();  
 Department getDepartment(int id);  
}

**SqlserverDepart:**

*/\*\*  
 \** ***@Description:*** *\** ***@Date:*** *Create in 19:10 2018/4/12  
 \*/*public class SqlserverDepart implements Department {  
 @Override  
 public void insert() {  
 System.*out*.println("插入一个部门--Sqlserver");  
 }  
  
 @Override  
 public Department getDepartment(int id) {  
 System.*out*.println("得到一个部门--Sqlserver");  
 return null;  
 }  
}

**MysqlDepart:**

*/\*\*  
 \** ***@Description:*** *\** ***@Date:*** *Create in 19:15 2018/4/12  
 \*/*public class MysqlDepart implements Department {  
 @Override  
 public void insert() {  
 System.*out*.println("增加一个部门--Mysql");  
 }  
  
 @Override  
 public Department getDepartment(int id) {  
 System.*out*.println("得到一个部门--Mysql");  
 return null;  
 }  
}

**Project:**

*/\*\*  
 \** ***@Description:*** *项目表  
 \** ***@Date:*** *Create in 19:06 2018/4/12  
 \*/*public interface Project {  
 void insert();  
 Project getProject(int id);  
}

**SqlserverProject:**

*/\*\*  
 \** ***@Description:*** *\** ***@Date:*** *Create in 19:11 2018/4/12  
 \*/*public class SqlserverProject implements Project {  
 @Override  
 public void insert() {  
 System.*out*.println("增加一个项目--Sqlserver");  
 }  
  
 @Override  
 public Project getProject(int id) {  
 System.*out*.println("得到一个项目--Sqlserver");  
 return null;  
 }  
}

**MysqlProject:**

*/\*\*  
 \** ***@Description:*** *\** ***@Date:*** *Create in 19:16 2018/4/12  
 \*/*public class MysqlProject implements Project {  
 @Override  
 public void insert() {  
 System.*out*.println("增加一个项目--Mysql");  
 }  
  
 @Override  
 public Project getProject(int id) {  
 System.*out*.println("得到一个项目--Mysql");  
 return null;  
 }  
}

**User:**

*/\*\*  
 \** ***@Description:*** *用户表  
 \** ***@Date:*** *Create in 19:05 2018/4/12  
 \*/*public interface User {  
 void insert();  
 User getUser(int id);  
}

**SqlserverUser:**

*/\*\*  
 \** ***@Description:*** *\** ***@Date:*** *Create in 19:09 2018/4/12  
 \*/*public class SqlserverUser implements User {  
 @Override  
 public void insert() {  
 System.*out*.println("增加一个用户--Sqlserver");  
 }  
  
 @Override  
 public User getUser(int id) {  
 System.*out*.println("得到一个用户--Sqlserver");  
 return null;  
 }  
}

**MysqlUser:**

*/\*\*  
 \** ***@Description:*** *\** ***@Date:*** *Create in 19:14 2018/4/12  
 \*/*public class MysqlUser implements User {  
 @Override  
 public void insert() {  
 System.*out*.println("增加一个用户--Mysql");  
 }  
  
 @Override  
 public User getUser(int id) {  
 System.*out*.println("得到一个用户--Mysql");  
 return null;  
 }  
}

**客户端：**

*/\*\*  
 \** ***@Author:*** *YongjieLv  
 \** ***@Description:*** *客户端  
 \** ***@Date:*** *Create in 19:18 2018/4/12  
 \** ***@Modified*** *By:  
 \*/*public class Main {  
 public static void main(String[] args) {  
 Factory factory=SqlserverFactory.*getInstance*(); // 创建具体的工厂  
 User user1=factory.createUser(); // 通过这个工厂去创建对应的产品  
 user1.insert(); // 对产品进行操作  
 user1.getUser(1);  
 Department department=factory.createDepartment();  
 department.insert();  
 department.getDepartment(1);  
 Project project=factory.createProject();  
 project.insert();  
 project.getProject(1);  
  
 Factory factory2=SqlserverFactory.*getInstance*(); // 验证单例模式，程序中只会存在一个具体工厂实例  
 }  
}