重构与模式:从重构的角度学习bridge设计模式

疯狂代码 http://CrazyCoder.cn/ j:http:/CrazyCoder.cn/Java/Article53622.html

从重构角度学习bridge设计模式 Bridge模式是个在实际系统中经常应用模式它最能体现设计模式原则针对接口进行编程和使用聚合不使用继承这两个原则

由于我们过分使用继承使类结构过于复杂不易理解难以维护特别是在Java中由于不能同时继承多个类这样就会造成多层继承维护更难

Bridge模式是解决多层继承根本原因如果你在实现应用中个类需要继承两个以上类并且这两者的间又持有某种关系它们两个都会有多种变化

Bridge模式是把这两个类分解为个抽象个实现使它们两个分离这样两种类可以独立变化

抽象就是把个实体共同概念(相同步骤)抽取出来(分解出几个相互独立步骤)作为个过程如我们把数据库 操作抽象为个过程有几个步骤创建SQL语句发送到数据库处理取得结果

实现就是怎样完成这个抽象步骤,如发送到数据库需要结合具体数据库考虑怎样完成这个步骤等并且同步骤可能存在区别实现如对区别数据库需要区别实现

现在我们假设个情况也是WEB中经常遇到在个page有输入框如客户信息姓名地址等输入信息后然后按查找按钮把查找结果显示出来

我们现在假设查找客户信息和帐户信息它们在区别表中

但是我们系统面对两种人群总部它们信息保存到oracle数据库但是各个分公司数据保存在Sybase中数据库位置等各不相同这两种操作区别

下面是我们般首先会使用方式使用 进行判断这样使用系统难以维护难以扩展不妨你增加种查询或者种数据库试试？

```
public  SearchAction{
public Vector searchData( ActionType,String DbType){
String SQL="";
(ActionType.equal("查找客户信息")){
//如果是查询客户信息拼SQL语句从客户表中读取数据
```

```
SQL="select * from Customer "
      (dbType.equal("oracle")){
//从总部数据库读取数据库为Oracle
String connect_ ="jdbc:oracle:thin:hr/hr@localhost:1521:HRDB";
DriverManager.registerDriver ( oracle.jdbc.OracleDriver);
Connection conn = DriverManager.getConnection (connect_);
// Create a statement
Statement stmt = conn.createStatement ;
ResultSet r = stmt.executeQuery (SQL);
//以下省略部分动态从数据库中取出数据组装成Vector,返回
.................................
.................................
}(dbType.equal("sybase")){
//从分公司数据库读取数据库为Sybase
String connect_ ="jdbc:sybase:Tds:cai/cai@192.168.1.12:1521:FIN";
DriverManager.registerDriver ( com.sybase.jdbc.SybDriver);
Connection conn = DriverManager.getConnection (connect_);
// Create a statement
Statement stmt = conn.createStatement ;
ResultSet r = stmt.executeQuery (SQL);
//以下省略部分动态从数据库中取出数据组装成Vector,返回
.................................
.................................
      }
      } (ActionType.equal("查找帐户信息")){
//如果是查询帐户信息拼接SQL语句从帐户表中读取数据
SQL="select * from Account "
(dbType.equal("oracle")){
.........................
.........................
(作者注:此处省略从oracle读取约300字)
      } (dbType.equal("Sybase")){
.........................
.........................
(作者注:此处省略从Sybase读取约300字)
}
```

```
    }
}
}
```

如果你认为这写比较弱智应该进行使用但是你也会大量使用

于是我们进行重构首先我们学习过DAO模式就是把数据读取进行分里我们定义个共同接口它负责数据库操作然后根据区别数据库进行实现在我们查询操作中使用接口进行操作这样就可以不用考虑具体实现我们只管实现过程　　查询共同接口:

```
    public erface searchDB{
public Vector searchFromDB(String SQL)
}
Oracle数据库查询实现
public  searchDBOracleImpl{
public Vector searchFromDB(String SQL){
//从总部数据库读取数据库为Oracle
String connect_ ="jdbc:oracle:thin:hr/hr@localhost:1521:HRDB";
DriverManager.registerDriver ( oracle.jdbc.OracleDriver);
    ResultSet r = stmt.executeQuery (SQL);
.............................
.............................
    }
}
Sybase数据库查询实现
public  searchDBSysbaseImpl{
public Vector searchFromDB(String SQL){
//从分公司数据库读取数据库为Sysbase
String connect_ ="jdbc:sybase:Tds:cai/cai@192.168.1.12:1521:FIN";
DriverManager.registerDriver ( com.sybase.jdbc.SybDriver);
ResultSet r = stmt.executeQuery (SQL);
.............................
.............................
    }
}
```

这样在我们查询中就可以使用接口searchDB但是创建有是个问题我们不能静态确定查询数据库类型必须动态确定于是我们又想到使用简单工厂思路方法来分别创建这里具体实现根据类别创建

```
        public  searchFactory{
```

```
public  searchDB createSearch( DBType){
(DBType.equal("oracle")){
 searchDBOracleImpl;
} (DBType.equal("sybase")){
 searchDBSysbaseImpl;
}
}
}
```

于是我们查询代码可以改变为这样了；

```
public  SearchAction{
      public Vector searchData( ActionType,String DbType){
String SQL="";
(ActionType.equal("查找客户信息")){
//如果是查询客户信息拼SQL语句从客户表中读取数据
SQL="select * from Customer "
searchDB obj=searchFactory.createSearch(DbType);
 obj.searchFromDB(SQL);
      } (ActionType.equal("查找帐户信息")){
//如果是查询帐户信息拼接SQL语句从帐户表中读取数据
SQL="select * from Account "
searchDB obj=searchFactory.createSearch(DbType);
 obj.searchFromDB(SQL);
}
}
}
```

是不是简单些如果增加个新数据库对我们只需增加个新数据库实现便可老代码不需改变这样便实现开－闭原则(Open-closed原则)在我们查询查询中使用是接口这就是设计模式原则针对接口进行编程并且使用聚合而不是直接继承大家可以考虑使用继承来完成该工作怎样实现上面是把实现进行分离实现可以动态变化！

我们把查询操作具体数据库实现进行了分离增强了灵活性但是我们查询仍然使用了 这样仍然不易进行扩展于是我们进行抽象个查询操作过程把它分成几个具体步骤创建SQL语句发送到数据库执行查询返回结果

它们虽然是区别查询SQL各不相同区别数据库执行区别返回结果内容区别但是这个过程却是不变于是我们声明个抽象类来完成这个过程

```
      public abstract  searchAction{
searchDB obj;
```

```
//两个步骤
public searchDB createSearchImple( DbType){
 searchFactory.createSearch(DbType);
}
public abstract String createSQL;
      //查询过程最后返回结果
public vector searchResult( DbType){
obj=createSearchImple(DbType);
 obj.searchFromDB(createSQL)
}
}
      //我们客户查询操作
public  searchCustomerAction{
public String createSQL{
 "select * from Customer"
}
}
//我们帐户查询操作
public  searchAccountAction{
public String createSQL{
 "select * from account"
}
}
```

这样我们查询编程简单创建SQL语句我们应该再创建个工厂思路方法来完成创建它们

```
public  actionFactory{
public  searchAction ceateAction( actionType){
(actionType.equal("customer")){
 searchCustomerAction;
} (actionType.equal("account")){
 searchAccountAction;
}
}
}
```

这样我们把查询操作过程进行了抽象定义了步骤和具体过程经过我们两次改变把抽象部分和实现部分进行分离使他们都可以独立变化增强灵活性

我们再看当初查询实现现在经过这两次地修改变成了什么模样？如下:

```
public  SearchAction{
public Vector searchData( ActionType,String DbType){
searchAction action=actionFactory.ceateAction(ActionType);
 action.searchResult(DbType);
}
```

2009-1-15 22:00:58
疯狂代码 http://CrazyCoder.cn/