**JavaEE第十二章 单例模式 代理模式**

1. **单例模式**

**单例模式：一个类只有一个对象**

**懒汉单例：线程不安全**

**饿汉单例：线程安全，使用synchronized修饰**

1. **JDBC单例实现步骤**
2. **把构造私有化，用private修饰构造**
3. **定义private static的Connection属性**
4. **把获取数据库连接Connection代码，放入私有构造中**
5. **提供static getConn方法，判断Connection是否为Null，为null则new 私有构造获取连接，不null则直接返回连接**

**实例：**

public class JDBCManager

{

    private static Connection con=null;

    private JDBCManager() {

        try

        {

            Class.forName("com.mysql.jdbc.Driver");

            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/yz?useUnicode=true&characterEncoding=utf-8", "root", "root");

        }

        catch (ClassNotFoundException e)

        {

            e.printStackTrace();

        }

        catch (SQLException e)

        {

            e.printStackTrace();

        }

    }

    public static Connection getConn() {

        if(con == null) {

           new JDBCManager();

        }

        return con;

    }

}

1. **代理模式**

**为其他对象提供一种代理以便控制对这个对象的访问**

1. **静态代理 （不够灵活，是以具体类为模板创建代理）**

**例子： 日志 功能案例：**

* 1. **创建学生日志代理类，实现IStudy接口**
  2. **在代理类里传入学生实现类StudentImpl,赋值给代理类的IStudy对象**
  3. **在代理类的Study方法中，调用IStudy对象的Study方法，并在它的前后，添加Log日志。**

**代码：**

public class StudentLogProxy implements IStudy

{

    private  IStudy stu;

    Logger log = Logger.getLogger(StudentLogProxy.class);

    public StudentLogProxy(IStudy stu) {

        this.stu = stu;

    }

    @Override

    public void study()

    {

        // TODO Auto-generated method stub

        log.debug("开始学习-------");

        stu.study();

        log.debug("学习结束-------");

    }

}

**2.动态代理(以巧妙的方式实现了代理模式的设计理念)**

        IStudy is = new StudentImpl();

        IStudy proxy = (IStudy)Proxy.newProxyInstance(is.getClass().getClassLoader(),is.getClass().getInterfaces(),

         new InvocationHandler()

         {

         @Override

         public Object invoke(Object arg0, Method arg1, Object[] arg2)

         throws Throwable

         {

         // TODO Auto-generated method stub

         System.out.println("打印。。。。");

         arg1.invoke(is, arg2);

         System.out.println("之后打印。。。。");

         return null;

         }

         });

        proxy.study();