2.3 为什么叫做面向切面编程

「切」：指的是横切逻辑，原有业务逻辑代码我们不能动，只能操作横切逻辑代码，所以面向横切逻辑

「面」：横切逻辑代码往往要影响的是很多个方法，每一个方法都如同一个点，多个点构成面，有一个面的概念在里面

第三部分 手写实现 IoC 和 AOP

上一部分我们理解了 IoC 和 AOP 思想，我们先不考虑 Spring 是如何实现这两个思想的，此处准备了一个『银行转账』的案例，请分析该案例在代码层次有什么问题？分析之后使用我们已有知识解决这些问题（痛点）。其实这个过程我们就是在一步步分析并手写实现 IoC 和 AOP。

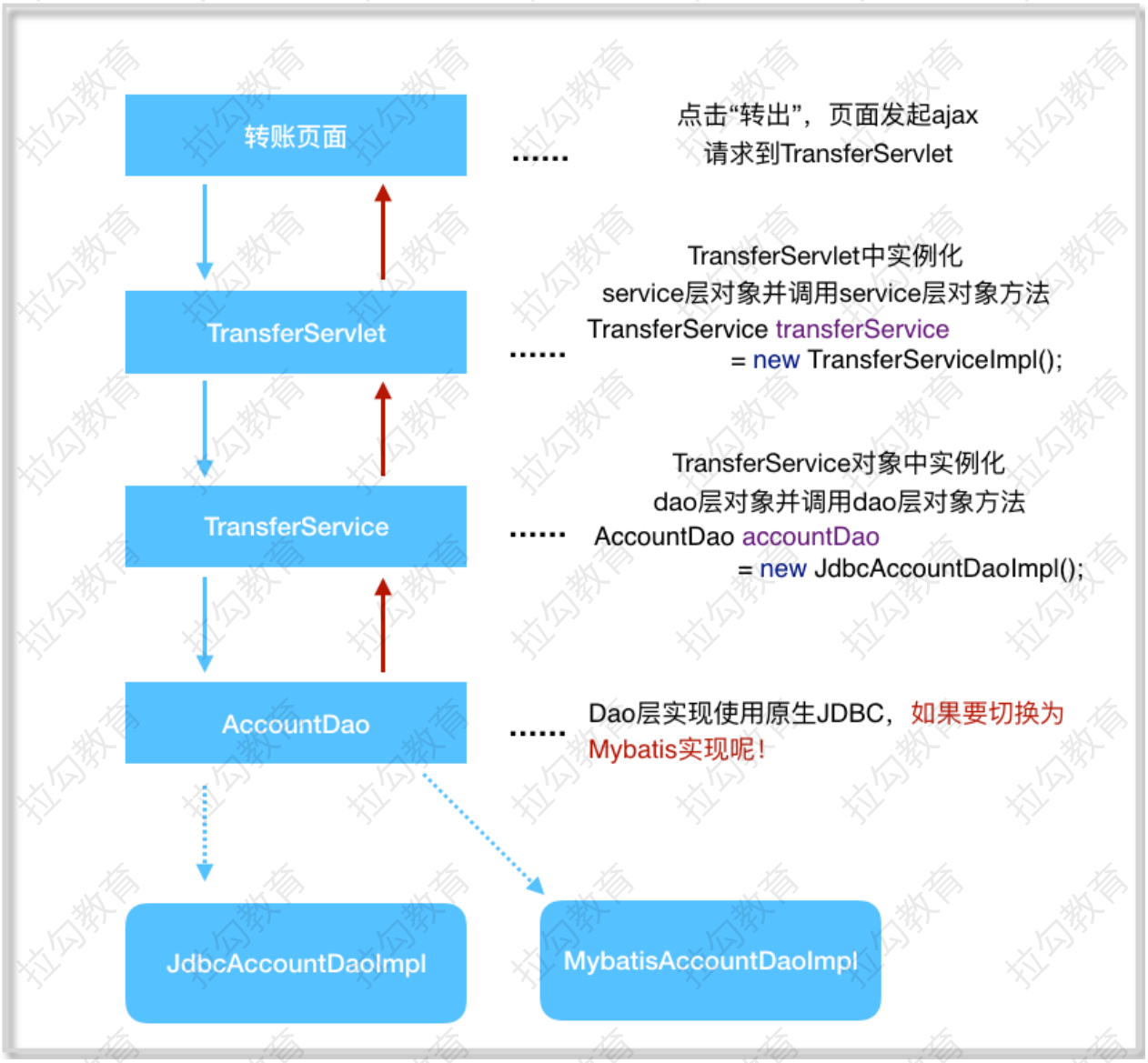
第1节 银行转账案例界面

第2节 银行转账案例表结构

字段 索引 外键 触发器 选项 注释 SQL 预览

名	类型	长度	小数点	不是 null	虚拟	键	注释
name	varchar	255	0	☐	☐		用户名
money	int	255	0	☐	☐		账户金额
cardNo	varchar	255	0	☑	☐	🔑 1	银行卡号

第3节 银行转账案例代码调用关系



第4节 银行转账案例关键代码

- TransferServlet

```
package com.lagou.edu.servlet;

import com.lagou.edu.service.impl.TransferServiceImpl;
import com.lagou.edu.utils.JsonUtils;
import com.lagou.edu.pojo.Result;
import com.lagou.edu.service.TransferService;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
```

```
 * @author 应癫
 */
@WebServlet(name="transferServlet",urlPatterns = "/transferServlet")
public class TransferServlet extends HttpServlet {


    // 1. 实例化service层对象
    private TransferService transferService = new TransferServiceImpl();

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        doPost(req,resp);
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse
resp) throws ServletException, IOException {

        // 设置请求体的字符编码
        req.setCharacterEncoding("UTF-8");

        String fromCardNo = req.getParameter("fromCardNo");
        String toCardNo = req.getParameter("toCardNo");
        String moneyStr = req.getParameter("money");
        int money = Integer.parseInt(moneyStr);

        Result result = new Result();

        try {

            // 2. 调用service层方法
            transferService.transfer(fromCardNo,toCardNo,money);
            result.setStatus("200");
        } catch (Exception e) {
            e.printStackTrace();
            result.setStatus("201");
            result.setMessage(e.toString());
        }

        // 响应
        resp.setContentType("application/json;charset=utf-8");
        resp.getWriter().print(JsonUtils.object2Json(result));
    }
}
```

- TransferService接口及实现类

```
package com.lagou.edu.service;

/**
 * @author 应癫
 */
public interface TransferService {
    void transfer(String fromCardNo,String toCardNo,int money) throws
Exception;
}
```

```
package com.lagou.edu.service.impl;

import com.lagou.edu.dao.AccountDao;
import com.lagou.edu.dao.impl.JdbcAccountDaoImpl;
import com.lagou.edu.pojo.Account;
import com.lagou.edu.service.TransferService;

/**
 * @author 应癫
 */
public class TransferServiceImpl implements TransferService {

    private AccountDao accountDao = new JdbcAccountDaoImpl();

    @Override
    public void transfer(String fromCardNo, String toCardNo, int money)
throws Exception {
        Account from = accountDao.queryAccountByCardNo(fromCardNo);
        Account to = accountDao.queryAccountByCardNo(toCardNo);

        from.setMoney(from.getMoney()-money);
        to.setMoney(to.getMoney()+money);

        accountDao.updateAccountByCardNo(from);
        accountDao.updateAccountByCardNo(to);
    }
}
```

- AccountDao层接口及基于Jdbc的实现类

```
package com.lagou.edu.dao;

import com.lagou.edu.pojo.Account;

/**
 * @author 应癫
 */
```

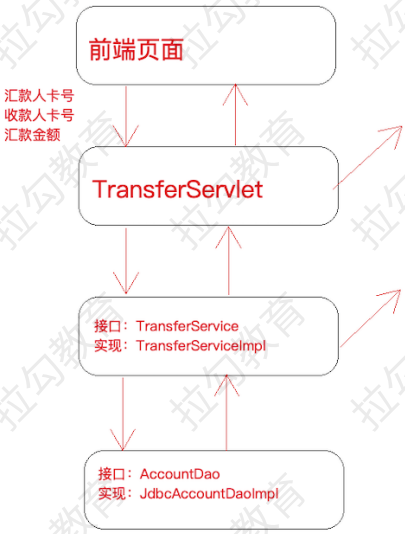
```
public interface AccountDao {

    Account queryAccountByCardNo(String cardNo) throws Exception;

    int updateAccountByCardNo(Account account) throws Exception;
}
```

JdbcAccountDaoImpl（Jdbc技术实现Dao层接口）

```
package com.lagou.edu.dao.impl;

import com.lagou.edu.pojo.Account;
import com.lagou.edu.dao.AccountDao;
import com.lagou.edu.utils.DruidUtils;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

/**
 * @author 应癫
 */
public class JdbcAccountDaoImpl implements AccountDao {


    @Override
    public Account queryAccountByCardNo(String cardNo) throws Exception {
        //从连接池获取连接
        Connection con = DruidUtils.getInstance().getConnection();
        String sql = "select * from account where cardNo=?";
        PreparedStatement preparedStatement = con.prepareStatement(sql);
        preparedStatement.setString(1,cardNo);
        ResultSet resultSet = preparedStatement.executeQuery();

        Account account = new Account();
        while(resultSet.next()) {
            account.setCardNo(resultSet.getString("cardNo"));
            account.setName(resultSet.getString("name"));
            account.setMoney(resultSet.getInt("money"));
        }

        resultSet.close();
        preparedStatement.close();
        con.close();

        return account;
    }
```

```
    @Override
    public int updateAccountByCardNo(Account account) throws Exception {

        //从连接池获取连接
        Connection con = DruidUtils.getInstance().getConnection();
        String sql = "update account set money=? where cardNo=?";
        PreparedStatement preparedStatement = con.prepareStatement(sql);
        preparedStatement.setInt(1,account.getMoney());
        preparedStatement.setString(2,account.getCardNo());
        int i = preparedStatement.executeUpdate();

        preparedStatement.close();
        con.close();
        return i;
    }
}
```

第5节 银行转账案例代码问题分析




```
public class TransferServlet extends HttpServlet {
    // 1. 实例化service层对象
    private TransferService transferService = new TransferServiceImpl();
```

```
public class TransferServiceImpl implements TransferService {

    private AccountDao accountDao = new JdbcAccountDaoImpl();
```

问题分析

问题一：new关键字将service层的实现类TransferServiceImpl和Dao层的具体实现类JdbcAccountDaoImpl耦合在了一起，当需要切换Dao层实现类的时候必须得修改service代码，不符合面向接口开发的最优原则

问题二：service层没有添加事务控制，出现异常可能导致数据错乱，问题很严重，尤其在金融银行行业

（1）问题一：在上述案例实现中，service 层实现类在使用 dao 层对象时，直接在 TransferServiceImpl 中通过 AccountDao accountDao = new JdbcAccountDaoImpl() 获得了 dao层对象，然而一个 new 关键字却将 TransferServiceImpl 和 dao 层具体的一个实现类 JdbcAccountDaoImpl 耦合在了一起，如果说技术架构发生一些变动，dao 层的实现要使用其它技术，比如 Mybatis，思考切换起来的成本？每一个 new 的地方都需要修改源代码，重新编译，面向接口开发的意义将大打折扣?

（2）问题二：service 层代码没有竟然还没有进行事务控制？！如果转账过程中出现异常，将可能导致数据库数据错乱，后果可能会很严重，尤其在金融业务。

第6节 问题解决思路

- 针对问题一思考：
  - 实例化对象的方式除了 new 之外，还有什么技术？反射 (需要把类的全限定类名配置在xml中)
- 考虑使用设计模式中的工厂模式解耦合，另外项目中往往有很多对象需要实例化，那就在工厂中使用反 射技术实例化对象，工厂模式很合适

问题一：new关键字将service层的实现类TransferServiceImpl和Dao层的具体实现类JdbcAccountDaoImpl耦合在了一起，当需要切换Dao层实现类的时候必须得修改service代码，不符合面向接口开发的最优原则

思考：

（1）new关键字在实例化对象，除了new以外还有什么技术可以实例化对象（反射）

Class.forName(“全限定类名”); com.lagou.edu.dao.JdbcAccountDaoImpl

可以把全限定类名配置在xml中

（2）使用工厂来通过反射技术生产对象，工厂模式是解耦合非常好的一种方式

- 更进一步，代码中能否只声明所需实例的接口类型，不出现 new 也不出现工厂类的字眼，如下图？ 能！声明一个变量并提供 set 方法，在反射的时候将所需要的对象注入进去吧

```
public class TransferServiceImpl implements TransferService {

    // 仅仅声明dao层接口
    private AccountDao accountDao;

    // 提供set方法供外部注入dao层实现类对象
    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }
```

- 针对问题二思考：
  - service 层没有添加事务控制，怎么办？没有事务就添加上事务控制，手动控制 JDBC 的 Connection 事务，但要注意将Connection和当前线程绑定（即保证一个线程只有一个 Connection，这样操作才针对的是同一个 Connection，进而控制的是同一个事务）

第7节 案例代码改造

（1）针对问题一的代码改造

- beans.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans>

    <bean id="transferService"
class="com.lagou.edu.service.impl.TransferServiceImpl">
        <property name="AccountDao" ref="accountDao"></property>
    </bean>
    <bean id="accountDao"
class="com.lagou.edu.dao.impl.JdbcAccountDaoImpl">
    </bean>

</beans>
```

- 增加 BeanFactory.java

```
package com.lagou.edu.factory;

import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.io.SAXReader;

import java.io.InputStream;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
```

```
/**
 * @author 应癫
 */
public class BeanFactory {

    /**
     * 工厂类的两个任务
     * 任务一：加载解析xml，读取xml中的bean信息，通过反射技术实例化bean对象，然后放入map待用
     * 任务二：提供接口方法根据id从map中获取bean（静态方法）
     */

    private static Map<String,Object> map = new HashMap<>();

    static {
        InputStream resourceAsStream = BeanFactory.class.getClassLoader().getResourceAsStream("beans.xml");
        SAXReader saxReader = new SAXReader();
        try {
            Document document = saxReader.read(resourceAsStream);
            Element rootElement = document.getRootElement();
            List<Element> list = rootElement.selectNodes("//bean");

            // 实例化bean对象
            for (int i = 0; i < list.size(); i++) {
                Element element =  list.get(i);
                String id = element.attributeValue("id");
                String clazz = element.attributeValue("class");

                Class<?> aClass = Class.forName(clazz);
                Object o = aClass.newInstance();
                map.put(id,o);
            }

            // 维护bean之间的依赖关系
            List<Element> propertyNodes = rootElement.selectNodes("//property");
            for (int i = 0; i < propertyNodes.size(); i++) {
                Element element =  propertyNodes.get(i);
                // 处理property元素
                String name = element.attributeValue("name");
                String ref = element.attributeValue("ref");



                String parentId = element.getParent().attributeValue("id");
                Object parentObject = map.get(parentId);
```

```
                Method[] methods = parentObject.getClass().getMethods();
                for (int j = 0; j < methods.length; j++) {
                    Method method = methods[j];
                    if(("set" + name).equalsIgnoreCase(method.getName()))
{
                        // bean之间的依赖关系（注入bean）
                        Object propertyObject = map.get(ref);
                        method.invoke(parentObject,propertyObject);
                    }
                }

                // 维护依赖关系后重新将bean放入map中
                map.put(parentId,parentObject);

            }
        } catch (DocumentException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            e.printStackTrace();
        }

    }


    public static Object getBean(String id) {
        return map.get(id);
    }
}
```

- 修改 TransferServlet

```
/**
 * @author 应癫
 */
@WebServlet(name="transferServlet",urlPatterns = "/transferServlet")
public class TransferServlet extends HttpServlet {

    private TransferService transferService = (TransferService) BeanFactory.getBean( id: "transferService");
```

- 修改 TransferServiceImpl

```java
/**
 * @author 应癫
 */
public class TransferServiceImpl implements TransferService {

    // 仅仅声明dao层接口
    private AccountDao accountDao;

    // 提供set方法供外部注入dao层实现类对象
    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }
```

(2) 针对问题二的改造

- 增加 ConnectionUtils

```java
package com.lagou.edu.utils;

import java.sql.Connection;
import java.sql.SQLException;

/**
 * @author 应癫
 */
public class ConnectionUtils {

    /*private ConnectionUtils() {

    }

    private static ConnectionUtils connectionUtils = new
ConnectionUtils();

    public static ConnectionUtils getInstance() {
        return connectionUtils;
    }*/


    private ThreadLocal<Connection> threadLocal = new ThreadLocal<>(); //
存储当前线程的连接

    /**
     * 从当前线程获取连接
     */
    public Connection getCurrentThreadConn() throws SQLException {
        /**
         * 判断当前线程中是否已经绑定连接，如果没有绑定，需要从连接池获取一个连接绑定到
当前线程
         */
```

```
        Connection connection = threadLocal.get();
        if(connection == null) {
            // 从连接池拿连接并绑定到线程
            connection = DruidUtils.getInstance().getConnection();
            // 绑定到当前线程
            threadLocal.set(connection);
        }
        return connection;

    }
}
```

- 增加 TransactionManager 事务管理器类

```
package com.lagou.edu.utils;

import java.sql.SQLException;

/**
 * @author 应癫
 */
public class TransactionManager {

    private ConnectionUtils connectionUtils;

    public void setConnectionUtils(ConnectionUtils connectionUtils) {
        this.connectionUtils = connectionUtils;
    }

    // 开启事务
    public void beginTransaction() throws SQLException {
        connectionUtils.getCurrentThreadConn().setAutoCommit(false);
    }

    // 提交事务
    public void commit() throws SQLException {
        connectionUtils.getCurrentThreadConn().commit();
    }

    // 回滚事务
    public void rollback() throws SQLException {
        connectionUtils.getCurrentThreadConn().rollback();
    }
}
```

- 增加 ProxyFactory 代理工厂类

```
package com.lagou.edu.factory;

import com.lagou.edu.utils.TransactionManager;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

/**
 * @author 应癫
 */
public class ProxyFactory {

    private TransactionManager transactionManager;

    public void setTransactionManager(TransactionManager
transactionManager) {
        this.transactionManager = transactionManager;
    }

    public Object getProxy(Object target) {
        return Proxy.newProxyInstance(this.getClass().getClassLoader(),
target.getClass().getInterfaces(), new InvocationHandler() {
            @Override
            public Object invoke(Object proxy, Method method, Object[]
args) throws Throwable {

                Object result = null;
                try{
                    // 开启事务
                    transactionManager.beginTransaction();
                    // 调用原有业务逻辑
                    result = method.invoke(target,args);
                    // 提交事务
                    transactionManager.commit();
                }catch(Exception e) {
                    e.printStackTrace();
                    // 回滚事务
                    transactionManager.rollback();

                    // 异常向上抛出,便于servlet中捕获
                    throw e.getCause();
                }

                return result;
            }
        });
    }
}
```

- 修改 beans.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--跟标签beans，里面配置一个又一个的bean子标签，每一个bean子标签都代表一个类的配置--
>
<beans>
    <!--id标识对象，class是类的全限定类名-->
    <bean id="accountDao"
class="com.lagou.edu.dao.impl.JdbcAccountDaoImpl">
        <property name="ConnectionUtils" ref="connectionUtils"/>
    </bean>
    <bean id="transferService"
class="com.lagou.edu.service.impl.TransferServiceImpl">
        <!--set+ name 之后锁定到传值的set方法了，通过反射技术可以调用该方法传入对应
的值-->
        <property name="AccountDao" ref="accountDao"></property>
    </bean>


    <!--配置新增的三个Bean-->
    <bean id="connectionUtils"
class="com.lagou.edu.utils.ConnectionUtils"></bean>

    <!--事务管理器-->
    <bean id="transactionManager"
class="com.lagou.edu.utils.TransactionManager">
        <property name="ConnectionUtils" ref="connectionUtils"/>
    </bean>

    <!--代理对象工厂-->
    <bean id="proxyFactory" class="com.lagou.edu.factory.ProxyFactory">
        <property name="TransactionManager" ref="transactionManager"/>
    </bean>
</beans>
```

- 修改 JdbcAccountDaoImpl

```
package com.lagou.edu.dao.impl;

import com.lagou.edu.pojo.Account;
import com.lagou.edu.dao.AccountDao;
import com.lagou.edu.utils.ConnectionUtils;
import com.lagou.edu.utils.DruidUtils;

import java.sql.Connection;
import java.sql.PreparedStatement;
```

```java
import java.sql.ResultSet;

/**
 * @author 应癫
 */
public class JdbcAccountDaoImpl implements AccountDao {

    private ConnectionUtils connectionUtils;

    public void setConnectionUtils(ConnectionUtils connectionUtils) {
        this.connectionUtils = connectionUtils;
    }

    @Override
    public Account queryAccountByCardNo(String cardNo) throws Exception {
        //从连接池获取连接
        // Connection con = DruidUtils.getInstance().getConnection();
        Connection con = connectionUtils.getCurrentThreadConn();
        String sql = "select * from account where cardNo=?";
        PreparedStatement preparedStatement = con.prepareStatement(sql);
        preparedStatement.setString(1,cardNo);
        ResultSet resultSet = preparedStatement.executeQuery();

        Account account = new Account();
        while(resultSet.next()) {
            account.setCardNo(resultSet.getString("cardNo"));
            account.setName(resultSet.getString("name"));
            account.setMoney(resultSet.getInt("money"));
        }

        resultSet.close();
        preparedStatement.close();
        //con.close();

        return account;
    }

    @Override
    public int updateAccountByCardNo(Account account) throws Exception {

        // 从连接池获取连接
        // 改造为：从当前线程当中获取绑定的connection连接
        //Connection con = DruidUtils.getInstance().getConnection();
        Connection con = connectionUtils.getCurrentThreadConn();
        String sql = "update account set money=? where cardNo=?";
        PreparedStatement preparedStatement = con.prepareStatement(sql);
        preparedStatement.setInt(1,account.getMoney());
        preparedStatement.setString(2,account.getCardNo());
        int i = preparedStatement.executeUpdate();
```

```
        preparedStatement.close();
        //con.close();
        return i;
    }
}
```

- 修改 TransferServlet

```
package com.lagou.edu.servlet;

import com.lagou.edu.factory.BeanFactory;
import com.lagou.edu.factory.ProxyFactory;
import com.lagou.edu.service.impl.TransferServiceImpl;
import com.lagou.edu.utils.JsonUtils;
import com.lagou.edu.pojo.Result;
import com.lagou.edu.service.TransferService;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * @author 应癫
 */
@WebServlet(name="transferServlet",urlPatterns = "/transferServlet")
public class TransferServlet extends HttpServlet {

    // 1. 实例化service层对象
    //private TransferService transferService = new TransferServiceImpl();
    //private TransferService transferService = (TransferService)
BeanFactory.getBean("transferService");

    // 从工厂获取委托对象（委托对象是增强了事务控制的功能）

    // 首先从BeanFactory获取到proxyFactory代理工厂的实例化对象
    private ProxyFactory proxyFactory = (ProxyFactory)
BeanFactory.getBean("proxyFactory");
    private TransferService transferService = (TransferService)
proxyFactory.getJdkProxy(BeanFactory.getBean("transferService")) ;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        doPost(req,resp);
```