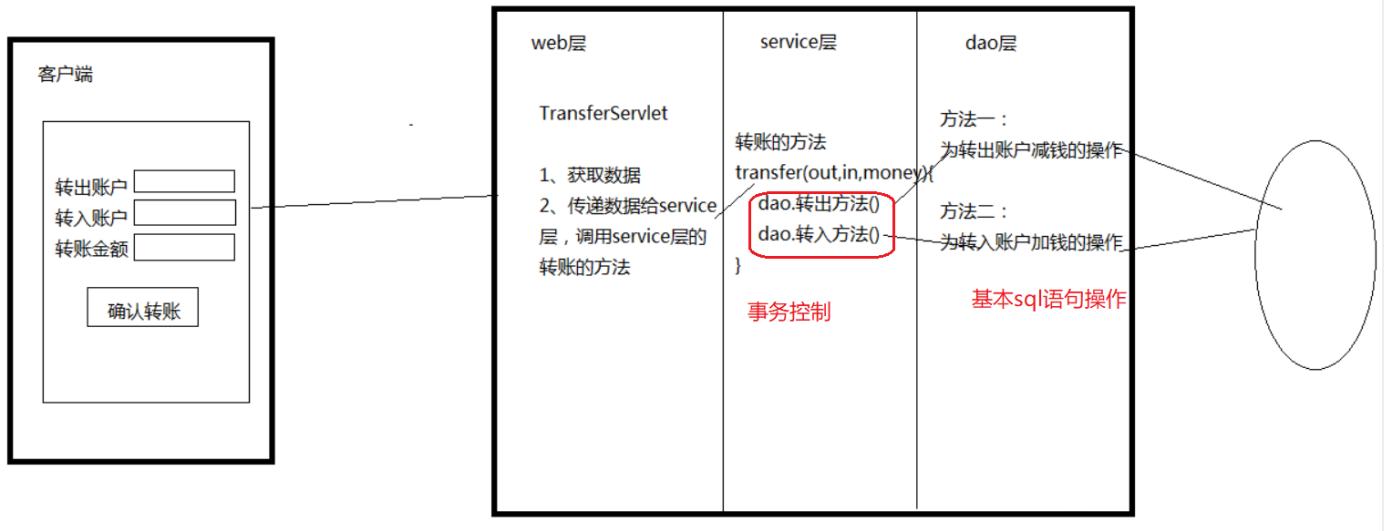
转账与ThreadLocal对象

一、转账初级

1.1 案例分析



1.2 环境搭建

1. 先导入包:操作数据库的mysql驱动包、c3p0连接池包、DBUtils包
2. 按照三层架构建包

3. 建一个转账的简单表单页面

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>转账</title>
</head>
<body>
    <form action="${pageContext.request.contextPath}/transfer" method="post">
    转出账户：<input type="text" name="out"><br>
    转入账户：<input type="text" name="in"><br>
    转出金额：<input type="text" name="money"><br>
    <input type="submit" value="确认转账"><br>
    </form>
</body>
</html>
```

1.3 代码详解

1.3.1 web层

```
public class TransferServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        //接收参数,需要进行非空判断
        String out = request.getParameter("out");
        String in = request.getParameter("in");
        String moneystr = request.getParameter("money");
        double money=Double.parseDouble(moneystr);
        //调用业务层的转账方法
        TransferService service=new TransferService();
        boolean isOK=service.transfer(out,in,money);
        if(isOK){
            response.getWriter().write("转账成功!!!");
        }else{
            response.getWriter().write("转账失败!!!");
        }


    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        doGet(request, response);
    }

}
```

获取表单数据

调用业务层的转账方法

1.3.2 service层

```java
    public boolean transfer(String out, String in, double money) {
        //调用dao层
        TransferDao dao=new TransferDao();
        boolean isOK=true;
        Connection conn=null;
        try {

            conn=DataSourceUtils.getConnection();
            conn.setAutoCommit(false);
            dao.transferOut(conn,out,money);
            //int i=1/0;
            dao.transferIn(conn,in,money);
        }catch (Exception e) {
            isOK=false;
            //有异常回滚事务
            try {
                conn.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
            e.printStackTrace();
        }finally{
            try {
            //提交事务，如果有异常则回滚到执行sql语句之前，此时再提交也无济于事，因为没有sql语句执行
                conn.commit();
            } catch (SQLException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        }

        return isOK;
```

异常通常不在dao层处理，而在本层service层

获取同一个连接

开启事务

异常回滚事务

就算有异常也提交，因为毕竟回滚了，不会有任何影响

注意，为什么要在finally那里提交事务，是为了确保事务能够关闭。

1.3.3 dao层

本层很简单

```java
public class TransferDao {


    public void transferOut(Connection conn,String out, double money) throws SQLException {
            QueryRunner runner=new QueryRunner();
            String sql="update account set umoney=umoney-? where uname=?";
            runner.update(conn, sql,money,out);

    }
    public void transferIn(Connection conn,String in, double money) throws SQLException {
            QueryRunner runner=new QueryRunner();

            String sql="update account set umoney=umoney+? where uname=?";
            runner.update(conn, sql,money,in);

    }

}
```

1.4 结果

略，没有遗漏同一个connection这种问题，事务控制也正确。再次提醒对DBUtils的记忆，本版本还是很简单。通过c3p0连接池，结合DBUtils，执行sql语句。

```java
package cn.scct.dataSourceUtils;
import java.sql.Connection;

public class DataSourceUtils {

    private static ComboPooledDataSource dataSource = new ComboPooledDataSource();

        public static Connection getConnection() throws SQLException{
            return dataSource.getConnection();
        }
        public static DataSource getDataSource() {
            return dataSource;
        }


}
```

二、 转账高级与ThreadLocal对象

2.1 不足之处

1. 为什么在service层中和dao层中都用到了一个Connection，这种跨层的数据不是污染了MVC架构吗?
2. 我们希望三层能够共享数据，如果需要共享数据的话，可以实现不通过传参的方法共享数据。
3. 这明显是单线程程序，为啥不考虑在线程中同步更新一个数据。

2.2 ThreadLocal对象

ThreadLocal一般称为**线程本地变量**，它是一种特殊的线程绑定机制，将变量与线程绑定在一起，为每一个线程维护一个独立的变量副本。通过ThreadLocal可以将对象的可见范围限制在同一个线程内。 ThreadLocal从本质上讲，无非是提供了一个“线程级”的变量作用域，它是一种线程封闭（每个线程独享变量）技术，更直白点讲，ThreadLocal可以理解为将对象的作用范围限制在一个线程上下文中，使得变量的作用域为“线程级”。

2.2.1 重新封装自己的DataSourceUtils包

```java
public class DataSourceUtils {
        //获得Connection ----- 从连接池中获取
        private static ComboPooledDataSource dataSource = new ComboPooledDataSource();

        //创建ThreadLocal
        private static ThreadLocal<Connection> tl = new ThreadLocal<Connection>();

        //开启事务
        public static void startTransaction() throws SQLException{
            Connection conn = getCurrentConnection();
            conn.setAutoCommit(false);
        }

        //回滚事务
        public static void rollback() throws SQLException {
            getCurrentConnection().rollback();
        }

        //提交事务
        public static void commit() throws SQLException {
            Connection conn = getCurrentConnection();
            conn.commit();
            //将Connection从ThreadLocal中移除
            tl.remove();
            conn.close();

        }
```

开启事务

获取当前Connection对象以执行事务操作

回滚事务

提交事务

最关键的代码是getCurrentConnection()

```java
    //获得当前线程上绑定的conn
    public static Connection getCurrentConnection() throws SQLException{
        //从ThreadLocal寻找 当前线程是否有对应Connection
        Connection conn = tl.get();
        if(conn==null){
            //获得新的connection
            conn = getConnection();
            //将conn资源绑定到ThreadLocal（map）上
            tl.set(conn);
        }
        return conn;
    }

    public static Connection getConnection() throws SQLException{
        return dataSource.getConnection();
    }
```

ThreadLocal对象中没有存Connection则获取一个并存入线程中

2.2.2 重写service层

```java
    public boolean transfer(String out, String in, double money) {
        //调用dao层
        TransferDao dao=new TransferDao();
        boolean isOK=true;
        try {
            //开启事务
            DataSourceUtils.startTransaction();

            dao.transferOut(out,money);
            //int i=1/0;
            dao.transferIn(in,money);
        }catch (Exception e) {
            isOK=false;
            //有异常回滚事务
            try {
                DataSourceUtils.rollback();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
            e.printStackTrace();
        }finally{
            try {
            //提交事务,如果有异常则回滚到执行sql语句之前，此时再提交也无济于事，因为没有sql语句执行
                DataSourceUtils.commit();
            } catch (SQLException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
        }

        return isOK;
    }
```

这就一目了然，没有connection对象的痕迹，但是调用的相关事务的操作所获得的Connection对象都能在各层获得

2.2.3 重写dao层

```java
public class TransferDao {


    public void transferOut(String out, double money) throws SQLException {
           QueryRunner runner=new QueryRunner();
           Connection conn=DataSourceUtils.getCurrentConnection();
           String sql="update account set umoney=umoney-? where uname=?";
           runner.update(conn, sql,money,out);

    }
    public void transferIn(String in, double money) throws SQLException {
           QueryRunner runner=new QueryRunner();
           Connection conn=DataSourceUtils.getCurrentConnection();
           String sql="update account set umoney=umoney+? where uname=?";
           runner.update(conn, sql,money,in);

    }

}
```

getCurrentConnection能实时获得线程中的Connection对象