第八章：JDBC

第1节：概述

在Java中，数据库存取技术可分为如下几类：

1. JDBC直接访问数据库
2. JDO技术（Java Data Object）
3. 第三方O/R工具，如Hibernate，Mybatis 等

JDBC是java访问数据库的基石，JDO，Hibernate等只是更好的封装了JDBC。

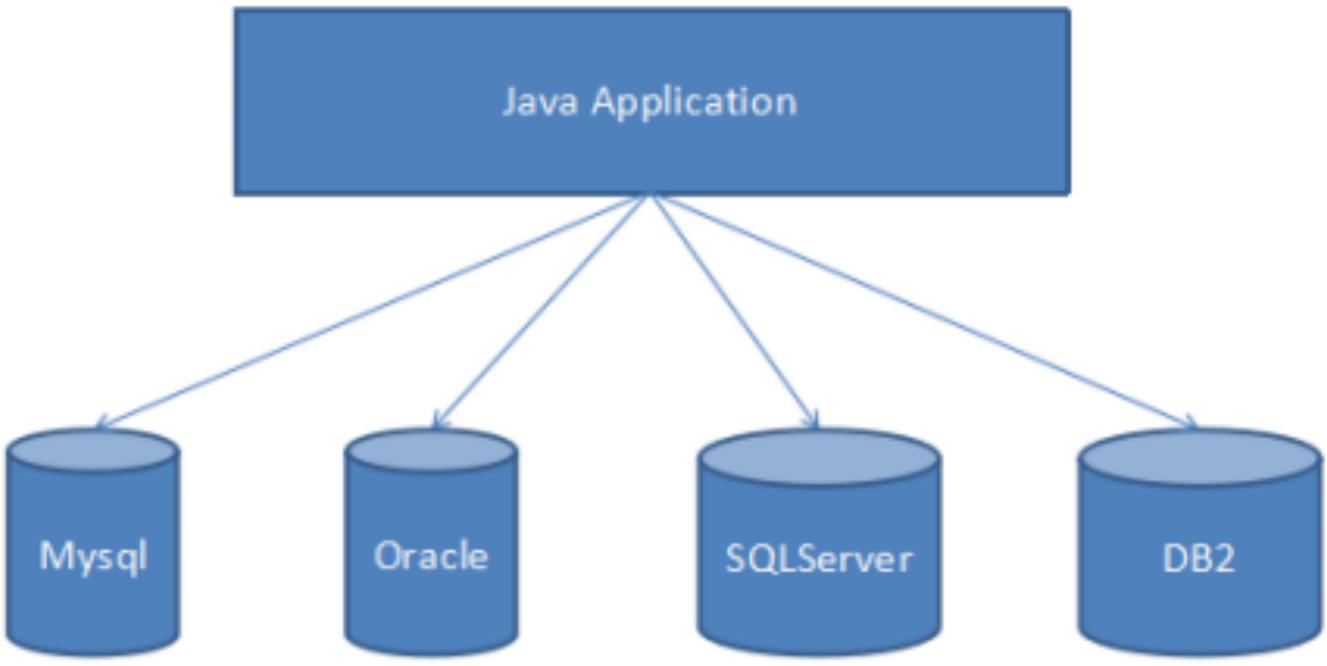
第2节：理解

JDBC(Java Database Connectivity)是一个独立于特定数据库管理系统（DBMS）、通用的SQL数据库存取和操作的公共接口（一组API），定义了用来访问数据库的标准Java类库，使用这个类库可以以一种标准的方法、方便地访问数据库资源

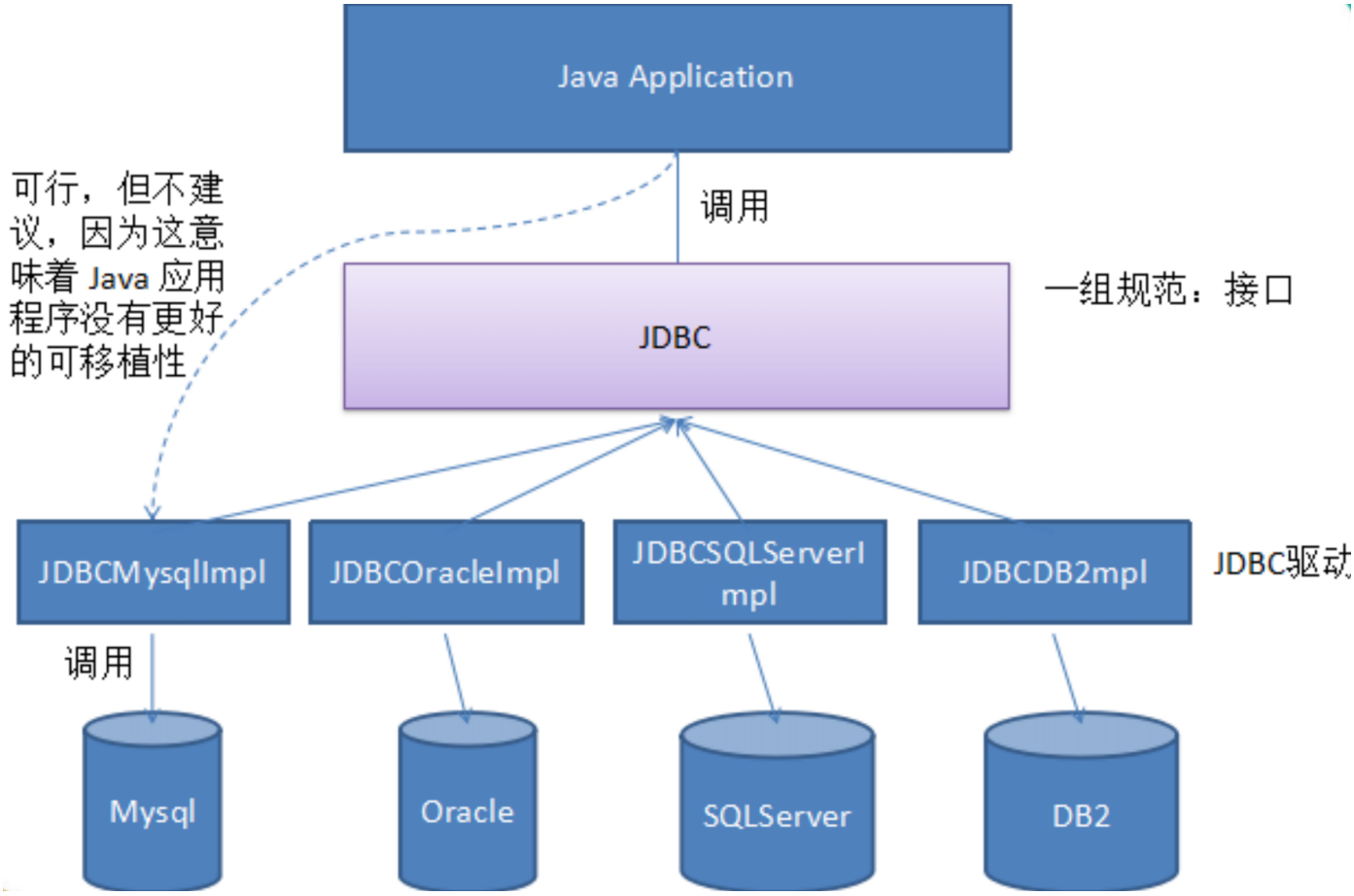
JDBC为访问不同的数据库提供了一种统一的途径，为开发者屏蔽了一些细节问题。

JDBC的目标是使Java程序员使用JDBC可以连接任何提供了JDBC驱动程序的数据库系统，这样就使得程序员无需对特定的数据库系统的特点有过多的了解，从而大大简化和加快了开发过程。

如果没有JDBC，那么Java程序访问数据库时是这样的：



有JDBC，那么Java程序访问数据库时是这样的：



第3节：JDBC API

JDBC API是一系列的接口，它统一和规范了应用程序与数据库的连接、执行SQL语句，并到得到返回结果等各类操作。声明在java.sql与javax.sql包中。

1. DriverManager类　管理不同的驱动
2. Connection 接口　应用和数据库的连接
3. Statement 接口　用于执行sql语句
4. ResultSet 接口　保存查询的结果

第4节：JDBC使用

1. 下载驱动:
   驱动程序由数据库提供商提供下载。 MySQL的驱动下载地址：http://dev.mysql.com/downloads/

2. 创建项目导入jar包

3. 使用:
   ① 注册驱动(加载驱动)
   ② 获取连接对象
   ③ 创建命令对象
   ④ 编写sql命令
   ⑤ 执行sql命令 返回结果
   ⑥ 处理结果
   ⑦ 释放资源

4. JDBC实现增删改查

第5节：junit

5.1 概述

JUnit是一个Java语言的单元测试框架。测试分为：黑盒测试和白盒测试。

黑盒测试又称功能测试，主要检测每个功能是否都能正常使用。在测试中，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，进行测试，主要针对软件界面和软件功能进行测试。

白盒测试又称结构测试、透明盒测试、逻辑驱动测试或基于代码的测试，主要检测程序内部逻辑是否正常。在测试中测试者必须检查程序的内部结构，从检查程序的逻辑着手，得出测试数据。按照程序内部的结构进行测试。这一方法是把测试对象看作一个打开的盒子，测试人员清楚盒子内部的东西以及里面是如何运作的

单元测试Junit就属于白盒测试.

5.2 使用

1. 把junit4.x的测试jar,添加到该项目中来;
2. 定义一个测试类 测试类的名字: XxxTest,如EmployeeDAOTest
3. 在EmployeeDAOTest中编写测试方法:如

```
@Test
public void testXxx() throws Exception {
}
```

注意:方法是public修饰的,无返回的,该方法上必须贴有@Test标签,XXX表示测试的功能名字

4. 选择某一个测试方法,鼠标右键选择 [run as junit],或则选中测试类,表示测试该类中所有的测试方法.

5.3 常见注解

1. @Test:要执行的测试方法
2. @Before:每次执行测试方法之前都会执行
3. @After: 每次执行测试方法之后都会执行
4. @BeforeClass:在所有的Before方法之前执行,只在最初执行一次. 只能修饰静态方法
5. @AfterClass:在所有的After方法之后执行,只在最后执行一次. 只能修饰静态方法

第6节：连接池

6.1 为什么使用连接池

1. 数据库连接是一种关键的有限的昂贵的资源，传统数据库连接每发出一个请求都要创建一个连接对象，使用完直接关闭不能重复利用；
2. 关闭资源需要手动完成，一旦忘记会造成内存溢出；
3. 请求过于频繁的时候，创建连接极其消耗内存；
4. 而且一旦高并发访问数据库，有可能会造成系统崩溃。

为了解决这些问题，我们可以使用连接池。

6.2 连接池原理

数据库连接池负责分配、管理和释放数据库连接，它的核心思想就是连接复用，通过建立一个数据库连接池，这个池中有若干个连接对象，当用户想要连接数据库，就要先从连接池中获取连接对象，然后操作数据库。一旦连接池中的连接对象被用完了，判断连接对象的个数是否已达上限，如果没有可以再创建新的连接对象，如果已达上限，用户必须处于等待状态，等待其他用户释放连接对象，直到连接池中有被释放的连接对象了，这时候等待的用户才能获取连接对象，从而操作数据库。这样就可以使连接池中的连接得到高效、安全的复用，避免了数据库连接频繁创建、关闭的开销。这项技术明显提高对数据库操作的性能。

6.3 连接池的好处

1. 程序启动时提前创建好连接，不用用户请求时创建，给服务器减轻压力；
2. 连接关闭的时候不会直接销毁connection，这样能够重复利用；
3. 如果超过设定的连接数量但是还没有达到最大值，那么可以再创建；
4. 如果空闲了，会默认销毁（释放）一些连接，让系统性能达到最优；

6.4 常见开源连接池

1. DBCP
   是Apache提供的数据库连接池，速度相对c3p0较快，但因自身存在BUG，Hibernate3已不再提供支持
2. C3P0
   是一个开源组织提供的一个数据库连接池，速度相对较慢，稳定性还可以
3. Druid
   是阿里提供的数据库连接池，据说是集DBCP 、C3P0 、Proxool 优点于一身的数据库连接池，但是速度不知道 是否有BoneCP快

6.5 连接池的使用

6.5.1 c3p0连接池

1. 下载导包 c3p0-0.9.5.2.jar mchange-commons-java-0.2.11.jar
2. 第一种方式代码配置
   ① 创建c3p0连接池对象

```
// 1. 创建一个连接池对象
ComboPooledDataSource cpds = new ComboPooledDataSource();
```

② 配置信息 （必配）

```
// 2. 配置信息（必配）
cpds.setDriverClass("com.mysql.jdbc.Driver");
cpds.setJdbcUrl("jdbc:mysql://localhost:3306/0831db");
cpds.setUser("root");
cpds.setPassword("root");
```

③ 可以选配信息

```
// 3. 选配
// 初始连接数
cpds.setInitialPoolSize(10);
// 最大连接数
cpds.setMaxPoolSize(100);
// 最小连接数
cpds.setMinPoolSize(10);

// 最大空闲时间
cpds.setMaxIdleTime(30);
```

④ 从池子中获取连接

```
// 4. 从池子中获取连接
Connection connection = cpds.getConnection();
```

3. 配置文件方式 （推荐使用）

① 创建 c3p0.properties 放在src目录下

```
# c3p0
c3p0.driverClass=com.mysql.jdbc.Driver
c3p0.jdbcUrl=jdbc:mysql://localhost:3306/0831db
c3p0.user=root
c3p0.password=root
# 选配
# 最大连接数
c3p0.maxPoolSize=100
# 最小连接数
c3p0.minPoolSize=10
# 初始化连接数
c3p0.initialPoolSize=10
# 最大空闲时间
c3p0.maxIdleTime=30
```

② 创建连接池对象 自动读取 配置文件信息

```java
// 1. 创建一个连接池对象
ComboPooledDataSource cpds = new ComboPooledDataSource();
```

```java
// 4. 从池子中获取连接
Connection connection = cpds.getConnection();
```

6.5.2 Druid连接池

1. 下载导包 druid-1.1.10.jar
2. 编写druid.properties 放在src目录下

```
#key=value
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/0831db?rewriteBatchedStatements=true
#url=jdbc:mysql://127.0.0.1:3306/db?useUnicode=true&characterEncoding=utf8&characterSetResults=utf8
username=root
password=root
initialSize=10
minIdle=5
maxActive=20
maxWait=5000
```

3. 读取配置文件

```java
// 1. 读取配置文件
// 属性集合 可以读取 .properties文件中的内容 当成流的形式
Properties properties = new Properties();

// 等价于 文件字节输入流
InputStream stream = Thread.currentThread()
        .getContextClassLoader().getResourceAsStream("druid.properties");


properties.load(stream);
```

4. 创建druid连接池对象

```java
// 2. 创建druid连接池对象
DataSource dataSource = DruidDataSourceFactory.createDataSource(properties);


// 3. 获取连接池的连接对象
Connection connection = dataSource.getConnection();


// 4. 创建命令对象
Statement statement = connection.createStatement();

// 5. 执行命令 返回结果
String sql = "select * from users where uid=1";
ResultSet resultSet = statement.executeQuery(sql);

// 6. 处理结果
if(resultSet.next()) {
    int uid = resultSet.getInt("uid");
    String uname = resultSet.getString("uname");
    String usex = resultSet.getString("usex");
    String ucardno = resultSet.getString("ucardno");
    int cid = resultSet.getInt("c_id");
    System.out.println(uid+"\t"+uname+"\t"+usex+"\t"+ucardno+"\t"+cid);
}

// 7. 将连接放回池子中
connection.close();
```

6.6 封装JDBCTools

ThreadLocal类

① 理解

JDK 1.2的版本中就提供java.lang.ThreadLocal，ThreadLocal为解决多线程程序的并发问题提供了一种新的思路。使用这个工具类可以很简洁地编写出优美的多线程程序。

② 原理

ThreadLocal用于保存某个线程共享变量，原因是在Java中，每一个线程中都有一个ThreadLocalMap<ThreadLocal, Object>，其key就是一个ThreadLocal，而Object即为该线程的共享变量。而这个map是通过ThreadLocal的set和get方法操作的。对于同一个static ThreadLocal，不同线程只能从中get，set，remove自己的变量，而不会影响其他线程的变量。

③ 使用

1、ThreadLocal.get: 获取ThreadLocal中当前线程共享变量的值。

2、ThreadLocal.set: 设置ThreadLocal中当前线程共享变量的值。

3、ThreadLocal.remove: 移除ThreadLocal中当前线程共享变量的值。

```java
// 1. 实例化ThreadLocal类
private static ThreadLocal<Connection> tl = new ThreadLocal<>();

static ComboPooledDataSource cpds;
static {

    cpds = new ComboPooledDataSource();

}

// 获取连接对象
public static Connection getConnection() throws SQLException {
    // 1. 从ThreadLocal中获取
    Connection connection = tl.get();

    if(connection==null) {

        // 2. 从池子中获取一个连接对象
        connection = cpds.getConnection();

        // 3. 将连接对象 绑定到 ThreadLocal
        tl.set(connection);
    }

    return connection;
}

// 封装释放连接对象的方法
public static void release() {

    try {
        Connection connection = tl.get();

        tl.remove(); // 将连接对象和ThreadLocal解绑

        connection.close();

    } catch (SQLException e) {

        e.printStackTrace();
    }

}
```

第7节：SQL注入

7.1 理解

SQL 注入是利用某些系统没有对用户输入的数据进行充分的检查，而在用户输入数据中注入非法的 SQL 语句段或命令，从而利用系统的 SQL 引擎完成恶意行为的做法。对于 Java 而言，要防范 SQL 注入，只要用 PreparedStatement 取代 Statement 就可以了。

7.2 解决

```
* 问题：sql注入
* 原因：sql语句中包含特殊符号 将整个sql语句改变
* 解决：将Statement 替换成子接口 PreparedStatment 预编译命令对象
* 原理：sql语句先编译 使用？作为占位符 将传递的参数如果出现特殊符号 自动转义
* PreparedStatment好处：
*   1. 避免了字符串和变量的拼接 屏蔽了细节问题
*   2. 解决了sql注入问题
*   3. sql语句可以复用
```

```
try {
    // 1. 获取连接对象
    Connection connection = JDBCUtils_final.getConnection();

    // 2. 创建命令对象
    String sql = "select * from user where username=? and password=?";
    PreparedStatement pStatement = connection.prepareStatement(sql);

    // 3. 将?替换成实际参数
    pStatement.setString(1, username);
    pStatement.setString(2, pwd);

    // 4. 执行命令 返回结果
    ResultSet rs = pStatement.executeQuery();

    System.out.println(rs.next()?"登录成功":"登录失败");


} catch (SQLException e) {

    e.printStackTrace();
} finally {
    JDBCUtils_final.release();
```

第8节：事务实现

1. 手动开启事务
   `setAutoCommit(false)`

```
// 开启事务：将事务的自动提交关闭 并且手动开启一个事务
connection.setAutoCommit(false);
```

2. 成功 提交
   `commit();`

```
// 如果没问题 提交
connection.commit();
```

3. 失败 回滚
   `rollback();`

```
// 如果有问题 回滚
try {
    connection.rollback();
} catch (SQLException e1) {
    // TODO Auto-generated catch block
    e1.printStackTrace();
}
```

第9节：批处理

9.1 概述

JDBC操作数据库的时候，需要一次性插入大量的数据的时候，如果每次只执行一条SQL语句，效率可能会比较低。这时可以使用batch操作，每次批量执行SQL语句，调高效率。

9.2 实现

1. 添加批处理
   `addBacth();`
2. 执行批处理
   `executeBacth();`
3. 清除批处理
   `clearBatch();`

第10节：DbUtils

10.1 简介

commons-dbutils 是 Apache 组织提供的一个开源 JDBC工具类库，它是对JDBC的简单封装，学习成本极低，并且使用dbutils能极大简化jdbc编码的工作量，同时也不会影响程序的性能。

10.2 作用

DbUtils ：提供如关闭连接、装载JDBC驱动程序等常规工作的工具类，里面的所有方法都是静态的。
该包封装了SQL的执行，是线程安全的。
（1）可以实现增、删、改、查、批处理、

（2）考虑了事务处理需要共用Connection。

（3）该类最主要的就是简单化了SQL查询，它与ResultSetHandler组合在一起使用可以完成大部分的数据库操作，能够大大减少编码量。

10.3 常用方法

1. 操作：update()
   public int update(Connection conn, String sql, Object... params) throws SQLException:用来执行一个更新（插入、更新或删除）操作。

2. 查询：query()
   public Object query(Connection conn, String sql, ResultSetHandler rsh,Object... params) throws SQLException：执行一个查询操作，在这个查询中，对象数组中的每个元素值被用来作为查询语句的置换参数。该方法会自行处理 PreparedStatement 和 ResultSet 的创建和关闭。

注：
   该接口用于处理 java.sql.ResultSet，将数据按要求转换为另一种形式。ResultSetHandler 接口提供了一个单独的方法：Object handle (java.sql.ResultSet rs)该方法的返回值将作为QueryRunner类的query()方法的返回值

方法：
   ArrayHandler：把结果集中的第一行数据转成对象数组。
   ArrayListHandler：把结果集中的每一行数据都转成一个数组，再存放到List中。
   BeanHandler：将结果集中的第一行数据封装到一个对应的JavaBean实例中。
   BeanListHandler：将结果集中的每一行数据都封装到一个对应的JavaBean实例中，存放到List里。
   ColumnListHandler：将结果集中某一列的数据存放到List中。
   KeyedHandler(name)：将结果集中的每一行数据都封装到一个Map里，再把这些map再存到一个map里，其key为指定的key。
   MapHandler：将结果集中的第一行数据封装到一个Map里，key是列名，value就是对应的值。
   MapListHandler：将结果集中的每一行数据都封装到一个Map里，然后再存放到List

10.4 使用

```
1. 下载导包   commons-dbutils-1.7.jar
2. 创建核心对象
      QueryRunner qr = new QueryRunner();

3. 执行命令
  int update = qr.update(connection,sql,params); // 执行增删改
  T t = qr.query(connection,sql,new BeanHandler,params);
  List t = qr.query(connection,sql,new BeanListHandler,params);
  Object t = qr.query(connection,sql,new ScalerHandler,params);
```

第11节：Dao封装

11.1 介绍

```
  Data Access Object访问数据信息的类和接口，包括了对数据的CRUD（Create、Retrival、Update、Delete），而不包含任何业务相关的信息
```

11.2 作用

为了实现功能的模块化，更有利于代码的维护和升级。

11.3 封装

```
package com.ujiuye.dao;
/**
 * 封装通用的增删改查的方法
 *  1. 增删改方法
 *  2. 单查
 * 3. 多查
 * 4. 查询个数
 * @author junguang
 *
 */

import java.sql.SQLException;
import java.util.List;

import org.apache.commons.dbutils.QueryRunner;
import org.apache.commons.dbutils.handlers.BeanHandler;
import org.apache.commons.dbutils.handlers.BeanListHandler;
import org.apache.commons.dbutils.handlers.ScalarHandler;

import com.ujiuye.utils.JDBCUtils;

public class BasicDao<T> {
```

```
  QueryRunner qr;
  {
    qr = new QueryRunner();
  }
  // 增删改
  public int update(String sql,Object...params) {

    try {

      return qr.update(JDBCUtils.getConnection(), sql, params);

    } catch (SQLException e) {

      // 将编译时异常转换成运行时异常
      throw new RuntimeException(e);
    } finally {
      JDBCUtils.release();
    }

  }

  // 单查
  public T querySingle(String sql,Class<T> clazz,Object...params) {

    try {
      return qr.query(JDBCUtils.getConnection(), sql, new BeanHandler<T>(clazz),
params);
    } catch (SQLException e) {

      // 将编译时异常转换成运行时异常
      throw new RuntimeException(e);
    } finally {
      JDBCUtils.release();
    }

  }

  // 多查
  public List<T> queryMore(String sql,Class<T> clazz,Object...params) {
    try {
      return qr.query(JDBCUtils.getConnection(), sql, new BeanListHandler<T>(clazz),
params);
    } catch (SQLException e) {

      // 将编译时异常转换成运行时异常
      throw new RuntimeException(e);
    } finally {
      JDBCUtils.release();
```

```
    }

  }

  // 查询聚合数据
  public Object scale(String sql,Object...params) {
    try {
      return qr.query(JDBCUtils.getConnection(), sql, new ScalarHandler<T>(), params);
    } catch (SQLException e) {
      throw new RuntimeException(e);
    } finally {
      JDBCUtils.release();
    }
  }
}
```

第12节：项目三层架构

12.1 理解

三层架构 通常意义上的三层架构就是将整个业务应用划分为：界面层、业务逻辑层、数据访问层。区分层次的目的即为了 “高内聚低耦合” 的思想。在软件体系架构设计中，分层式结构是最常见，也是最重要的一种结构。微软推荐的分层式结构一般分为三层，从下至上分别为：数据访问层、业务逻辑层（又或称为领域层）、表示层。

表示层（JSP）

表现层也称为界面层，位于最外层（最上层），离用户最近。用于显示数据和接收用户输入的数据，为用户提供一种交互式操作的界面。

业务层（service）

业务层在体系架构中的位置很关键，它处于数据访问层与表示层中间，起到了数据交换中承上启下的作用。

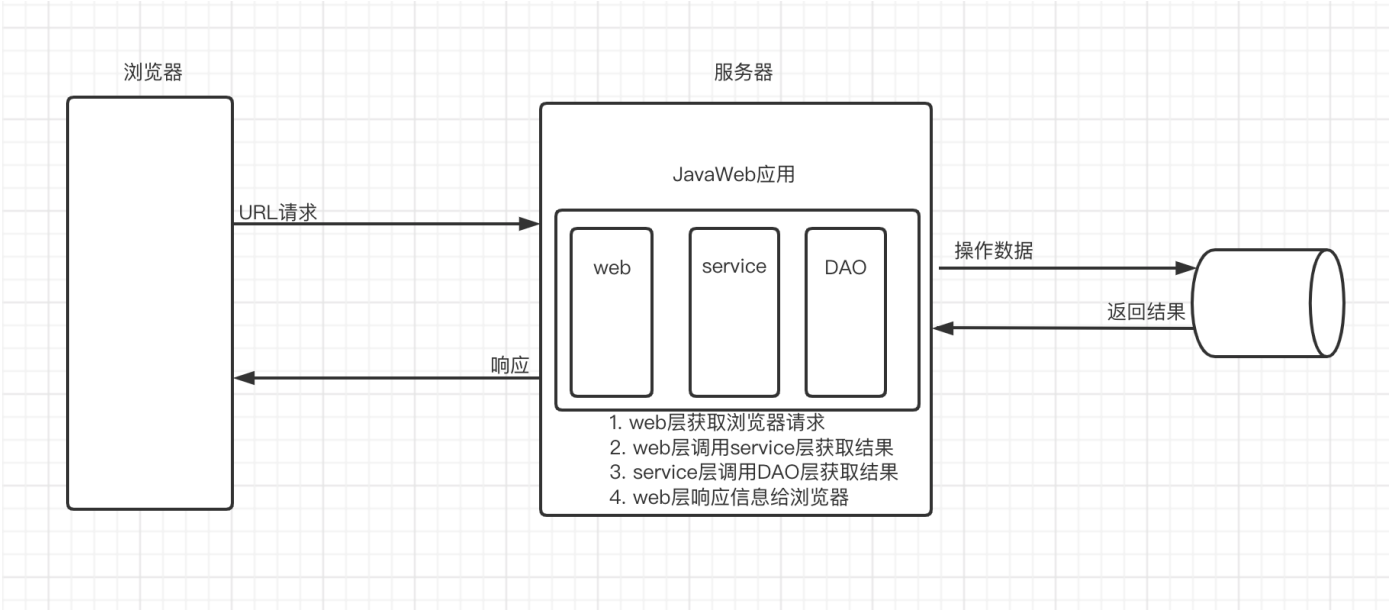
由于层是一种弱耦合结构，层与层之间的依赖是向下的，底层对于上层而言是“无知”的，改变上层的设计对于其调用的底层而言没有任何影响。如果在分层设计时，遵循了面向接口设计的思想，那么这种向下的依赖也应该是一种弱依赖关系。

持久层（DAO）

持久层，有时候也称为是数据访问层，其功能主要是负责数据库的访问，可以访问数据库系统

12.2 特点

1. 上一层可以调用下一层　但是 下一层不能调用上一层
2. 不可以隔层调用


浏览器
服务器
JavaWeb应用
URL请求
web
service
DAO
操作数据
返回结果
响应
1. web层获取浏览器请求
2. web层调用service层获取结果
3. service层调用DAO层获取结果
4. web层响应信息给浏览器