目标:

1. 连接池
2. 分页

# 连接池

思考：

程序中连接如何管理？

1. 连接资源宝贵；需要对连接管理 MySql同时连接数只有几百个，数据库承受能力有限
2. 连接：
   1. 操作数据库，创建连接
   2. 操作结束， 关闭！

分析：

涉及频繁的连接的打开、关闭，影响程序的运行效率！

连接管理：

预先创建一组连接，用的时候每次取出一个； 用完后，放回；

本质上连接池就是一个Connection的集合

学习连接池：

1. 自定义一个连接池
2. 学习优秀的连接池组件
   1. DBCP
   2. C3P0

## 自定义连接池

程序实现思路：

1. 指定“初始化连接数目” 【3】

(App启动时候，就执行创建!)

2. 指定“最大连接数” 【6】

3. 指定“当前使用连接个数” 【不能超出最大连接数】

代码实现：

1. MyPool.java 连接池类，

2. 指定全局参数： 初始化数目、最大连接数、当前存在的连接数、 连接池集合

3. 构造函数：循环创建3个连接

4. 写一个创建连接的方法

5. 获取连接

------> 判断： 池中有连接， 直接拿

------> 池中没有连接，

------> 判断，是否达到最大连接数； 达到，抛出异常；没有达到最大连接数，

创建新的连接

6. 释放连接

-------> 连接放回集合中(判断是否达到初始化连接数，达到了则关闭连接，没达到则放入集合中)

代理：

如果对某个接口中的某个指定的方法的功能进行扩展，而不想实现接口里所有方法，可以使用(动态)代理模式!

Java中代理模式：静态/**动态**/Cglib代理(spring)

使用动态代理，可以监测接口中方法的执行！

如何针对Connection对象，生成一个代理对象：

|--Proxy

static Object newProxyInstance(

ClassLoader loader, 当前使用的类加载器

Class<?>[] interfaces, 目标对象(Connection)实现的接口类型

InvocationHandler h 事件处理器：当执行上面接口中的方法的时候，就会自动触发事件处理器代码，把当前执行的方法(method)作为参数传入。

)

|  |
| --- |
| /\*\*  \* 自定义连接池, 管理连接  \* 代码实现：  1. MyPool.java 连接池类，  2. 指定全局参数： 初始化数目、最大连接数、当前连接、 连接池集合  3. 构造函数：循环创建3个连接  4. 写一个创建连接的方法  5. 获取连接  ------> 判断： 池中有连接， 直接拿  ------> 池中没有连接，  ------> 判断，是否达到最大连接数； 达到，抛出异常；没有达到最大连接数，  创建新的连接  6. 释放连接  -------> 连接放回集合中(..)  \*  \*/  **public** **class** MyPool {  **private** **int** init\_count = 3; // 初始化连接数目  **private** **int** max\_count = 6; // 最大连接数  **private** **int** current\_count = 0; // 记录当前使用连接数  // 连接池 （存放所有的初始化连接）  **private** LinkedList<Connection> pool = **new** LinkedList<Connection>();      //1. 构造函数中，初始化连接放入连接池  **public** MyPool() {  // 初始化连接  **for** (**int** i=0; i<init\_count; i++){  // 记录当前连接数目  current\_count++;  // 创建原始的连接对象  Connection con = createConnection();  // 把连接加入连接池  pool.addLast(con);  }  }    //2. 创建一个新的连接的方法  **private** Connection createConnection(){  **try** {  Class.*forName*("com.mysql.jdbc.Driver");  // 原始的目标对象  **final** Connection con = DriverManager.*getConnection*("jdbc:mysql:///jdbc\_demo", "root", "root");    /\*\*\*\*\*\*\*\*\*\*对con对象代理\*\*\*\*\*\*\*\*\*\*\*\*\*\*/    // 对con创建其代理对象  Connection proxy = (Connection) Proxy.*newProxyInstance*(    con.getClass().getClassLoader(), // 类加载器  //con.getClass().getInterfaces(), // 当目标对象是一个具体的类的时候  **new** Class[]{Connection.**class**}, // 目标对象实现的接口    **new** InvocationHandler() { // 当调用con对象方法的时候， 自动触发事务处理器  @Override  **public** Object invoke(Object proxy, Method method, Object[] args)  **throws** Throwable {  // 方法返回值  Object result = **null**;  // 当前执行的方法的方法名  String methodName = method.getName();    // 判断当执行了close方法的时候，把连接放入连接池  **if** ("close".equals(methodName)) {  System.*out*.println("begin:当前执行close方法开始！");  // 连接放入连接池  pool.addLast(con);  System.*out*.println("end: 当前连接已经放入连接池了！");  } **else** {  // 调用目标对象方法  result = method.invoke(con, args);  }  **return** result;  }  }  );  **return** proxy;  } **catch** (Exception e) {  **throw** **new** RuntimeException(e);  }  }    //3. 获取连接  **public** Connection getConnection(){    // 3.1 判断连接池中是否有连接, 如果有连接，就直接从连接池取出  **if** (pool.size() > 0){  **return** pool.removeFirst();  }    // 3.2 连接池中没有连接： 判断，如果没有达到最大连接数，创建；  **if** (current\_count < max\_count) {  // 记录当前使用的连接数  current\_count++;  // 创建连接  **return** createConnection();  }    // 3.3 如果当前已经达到最大连接数，抛出异常  **throw** **new** RuntimeException("当前连接已经达到最大连接数目 ！");  }      //4. 释放连接  **public** **void** realeaseConnection(Connection con) {  // 4.1 判断： 池的数目如果小于初始化连接，就放入池中  **if** (pool.size() < init\_count){  pool.addLast(con);  } **else** {  **try** {  // 4.2 关闭  current\_count--;  con.close();  } **catch** (SQLException e) {  **throw** **new** RuntimeException(e);  }  }  }    **public** **static** **void** main(String[] args) **throws** SQLException {  MyPool pool = **new** MyPool();  System.*out*.println("当前连接: " + pool.current\_count); // 3    // 使用连接  pool.getConnection();  pool.getConnection();  Connection con4 = pool.getConnection();  Connection con3 = pool.getConnection();  Connection con2 = pool.getConnection();  Connection con1 = pool.getConnection();    // 释放连接, 连接放回连接池  // pool.realeaseConnection(con1);  /\*  \* 希望：当关闭连接的时候，要把连接放入连接池！【当调用Connection接口的close方法时候，希望触发pool.addLast(con);操作】  \* 把连接放入连接池  \* 解决1：实现Connection接口，重写close方法  \* 解决2：动态代理  \*/  con1.close();    // 再获取  pool.getConnection();    System.*out*.println("连接池：" + pool.pool.size()); // 0  System.*out*.println("当前连接: " + pool.current\_count); // 3  }    } |

代理的总结：(了解会用)

使用代理，可以在不实现接口的情况，对接口的方法进行扩展，添加额外的用户需要的业务逻辑！

配置文件主要分为Properties和XML，那么什么时候用Properties，什么时候用XML呢？

当配置内容是简单的键值对的时候用Properties

当配置内容是一个关系型结构时用XML

# 2.开源的连接池技术

概述：

Sun公司约定： 如果是连接池技术，需要实现一个接口！

javax.sql.DataSource;

连接池：

DBCP

C3P0

所有的连接池技术都包括如下几方面

1.初始化连接数

2.最大连接数

3.当前存在的连接数

4.获取连接

5.释放连接

## DBCP连接池：

* DBCP 是 Apache 软件基金组织下的开源连接池实现，使用DBCP数据源，应用程序应在系统中增加如下两个 jar 文件：
  + Commons-dbcp.jar：连接池的实现
  + Commons-pool.jar：连接池实现的依赖库
* Tomcat 的连接池正是采用该连接池来实现的。该数据库连接池既可以与应用服务器整合使用，也可由应用程序独立使用。
* 核心类：BasicDataSource
* 使用步骤
  + 引入jar文件
    - commons-dbcp-1.4.jar
    - commons-pool-1.5.6.jar

|  |
| --- |
| /\*\*  \* 硬编码方式连接池使用  \*/  @Test  **public** **void** test1()  {  **try** {    /\*\*  \* 1.初始化DBCP数据源参数  \*/  BasicDataSource ds=**new** BasicDataSource();  ds.setUrl("jdbc:mysql://localhost:3306/demo?useSSL=false&useUnicode=true&characterEncoding=UTF8");//连接url  ds.setUsername("root");//用户名  ds.setPassword("root");//密码  ds.setDriverClassName("com.mysql.jdbc.Driver");//驱动    ds.setInitialSize(3);//初始化连接数  ds.setMaxActive(6);//最大连接数  ds.setMaxIdle(3000);//连接空闲时间，优化性能    /\*\*  \* 2.获取连接  \*/  Connection con=ds.getConnection();  QueryRunner qr=**new** QueryRunner();  List<User> list=qr.query(con, "select \* from user",**new** BeanListHandler<User>(  User.**class**));    System.***out***.println(list);    } **catch** (Exception e) {  e.printStackTrace();  }  }    /\*\*  \* 配置文件  \*/  @Test  **public** **void** test2()  {  **try** {  /\*\*  \* 1.读取配置文件，初始化参数  \*/  InputStream inputStream = MyPool.**class**.getResourceAsStream("/db.properties");    Properties properties=**new** Properties();  properties.load(inputStream);    /\*\*  \* 2.获取数据源(连接池)  \*/  DataSource ds = BasicDataSourceFactory.*createDataSource*(properties);    /\*\*  \* 3.获取连接  \*/  Connection con=ds.getConnection();  QueryRunner qr=**new** QueryRunner();  List<User> list=qr.query(con, "select \* from user",**new** BeanListHandler<User>(  User.**class**));    System.***out***.println(list);    } **catch** (Exception e) {  e.printStackTrace();  }  } |
| 配置方式实现DBCP连接池， 配置文件中的key与BaseDataSouce中的属性一样： |
| db.properties |
| url=jdbc:mysql:///demo  driverClassName=com.mysql.jdbc.Driver  username=root  password=root  initialSize=3  maxActive=6  maxIdle=3000 |

## C3P0连接池：

C3P0连接池：

最常用的连接池技术！Spring框架，默认支持C3P0连接池技术！

C3P0连接池，核心类：

CombopooledDataSource ds;

使用：

1. 下载，引入jar文件: c3p0-0.9.1.2.jar
2. 使用连接池，创建连接
   1. 硬编码方式
   2. 配置方式(xml)

|  |
| --- |
| /\*\*  \* 硬编码演示  \*/  @Test  **public** **void** test1() {  **try** {  ComboPooledDataSource ds = **new** ComboPooledDataSource();  /\*\*  \* 1.配置参数  \*/  ds.setJdbcUrl("jdbc:mysql://localhost:3306/demo?useSSL=false&useUnicode=true&characterEncoding=UTF8");// 连接url  ds.setUser("root");// 用户名  ds.setPassword("root");// 密码  ds.setDriverClass("com.mysql.jdbc.Driver");// 驱动  ds.setInitialPoolSize(3);// 初始化连接数  ds.setMaxPoolSize(6);// 最大连接数  ds.setMaxIdleTime(3000);// 连接空闲时间，优化性能  /\*\*  \* 2.获取连接  \*/  Connection con = ds.getConnection();  QueryRunner qr = **new** QueryRunner();  List<User> list = qr.query(con, "select \* from user",  **new** BeanListHandler<User>(User.**class**));  System.***out***.println(list);  /\*\*  \* 3.释放连接，猜测是做了代理  \*/  con.close();  } **catch** (Exception e) {  e.printStackTrace();  }  }  /\*\*  \* 配置文件 配置文件的名称是固定的，配置文件要放在src的根目录下  \*/  @Test  **public** **void** test2() {  **try** {  // 加载默认的配置  ComboPooledDataSource ds = **new** ComboPooledDataSource();  // 加载其他数据库配置  // ComboPooledDataSource ds=new  // ComboPooledDataSource("oracle\_config");  /\*\*  \* 2.获取连接  \*/  Connection con = ds.getConnection();  QueryRunner qr = **new** QueryRunner();  List<User> list = qr.query(con, "select \* from user",  **new** BeanListHandler<User>(User.**class**));  System.***out***.println(list);  /\*\*  \* 3.释放连接，猜测是做了代理  \*/  con.close();  } **catch** (Exception e) {  e.printStackTrace();  }  } |

## 优化

项目，连接的管理，交给连接池！

# 分页技术

分页技术：

JSP页面，用来显示数据！ 如果数据有1000条，分页显示，每页显示10条，共100页; 好处: 利于页面布局，且显示的效率高！

分页关键点：

1. 分页SQL语句；
2. 后台处理： dao/service/servlet/JSP

实现步骤：

1. 环境准备
   1. 引入jar文件及引入配置文件
      1. 数据库驱动包
      2. C3P0连接池jar文件 及 配置文件
      3. DbUtis组件: QueryRunner qr = new QueryRuner(dataSouce);

qr.update(sql);

* 1. 公用类: JdbcUtils.java

1. 先设计：PageBean.java
2. Dao接口设计/实现： 2个方法
3. Service/servlet
   1. JSP