1. 数据库连接池
2. 概述

数据库连接池是一个用于存放数据库连接的容器。当系统初始化后，数据库连接池被创建，它会申请一些连接对象，当用户访问数据库时，，会从连接池中获取连接对象用于连接数据库；当访问结束时，将连接对象返回给数据库连接池。

1. 优势

访问数据库效率提高，且节约了资源。

1. 实现
2. 接口

Javax.sql包下的DataSource接口。

1. 方法

从数据库连接池中获取连接： getConnection()

将连接归还到数据库连接池：Connection.close()，直接用close()就可以将连接归还到连接池。

1. 数据库连接池示例

C3P0： 数据库连接池技术

Druid: 阿里巴巴的数据库连接池技术

1. C3P0数据库连接池示例
2. 步骤
3. 在项目中创建lib目录，导入jar包，c3p0.jar，mchange-commons-java.jar和数据库驱动jar包。
4. 添加配置文件 c3p0.properties或者c3p0.xml，将配置文件直接放到src目录下即可。
5. 创建数据库连接池对象 ComboPooledDataSource对象。
6. 获取连接 getConnection()
7. 配置文件参数
8. 默认配置

<default-config>  
 <!-- 连接参数 -->  
 <property name="driverClass">com.mysql.cj.jdbc.Driver</property>  
 <property name="jdbcUrl">jdbc:mysql://localhost:3306/heima</property>  
 <property name="user">root</property>  
 <property name="password">123</property>  
   
 <!-- 连接池参数 -->

<!-- 初始连接对象数 -->  
 <property name="initialPoolSize">5</property>  
 <!-- 最大连接对象数 -->

<property name="maxPoolSize">10</property>  
 <!-- 超时时间 -->

<property name="checkoutTimeout">3000</property>  
</default-config>

1. 指定配置名

<named-config name="otherc3p0">   
 <!-- 连接参数 -->  
 <property name="driverClass">com.mysql.cj.jdbc.Driver</property>  
 <property name="jdbcUrl">jdbc:mysql://localhost:3306/heima</property>  
 <property name="user">root</property>  
 <property name="password">123</property>  
   
 <!-- 连接池参数 -->  
 <property name="initialPoolSize">5</property>  
 <property name="maxPoolSize">8</property>  
 <property name="checkoutTimeout">1000</property>  
</named-config>

1. Druid连接池技术的使用

（一）步骤

1. 导入druid.jar包
2. 定义配置文件，xx.properties，可放在任意目录下。
3. 加载配置文件，可将配置文件放在src目录下，通过classLoader()来获取该配置文件并加载。
4. 获取数据库连接池对象，通过DruidDataSourceFactory()获取。
5. 获取连接，通过getConnection()获取。
6. 连接池工具类

package utils;  
  
import com.alibaba.druid.pool.DruidDataSourceFactory;  
  
import javax.sql.DataSource;  
import java.io.IOException;  
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.Properties;  
  
public class JDBCUtils {  
  
 private static DataSource *ds*;  
 static{  
  
 try {  
 //加载配置文件  
 Properties pro = new Properties();  
 pro.load(JDBCUtils.class.getClassLoader().getResourceAsStream("druid.properties"));  
 //获取连接池对象  
 *ds* = DruidDataSourceFactory.*createDataSource*(pro);  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 //获取连接 方法  
 public static Connection getConnection() throws SQLException {  
 return *ds*.getConnection();  
 }  
  
 //释放资源  
 public static void close(Statement stm, Connection con){  
 if(stm != null){  
 try {  
 stm.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 if(con != null){  
 try {  
 con.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public static void close(ResultSet rs, Statement stm, Connection con){  
 if(rs != null){  
 try {  
 rs.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 if(stm != null){  
 try {  
 stm.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 if(con != null){  
 try {  
 con.close();  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 //获取连接池方法  
 public static DataSource getDataSource(){  
 return *ds*;  
 }  
}

1. JdbcTemplate
2. 概述

Spring框架提供了JDBC的简单封装，提供了一个JDBCTemplate对象简化了JDBC的开发。不需要自己创建连接，释放资源，只需要专注于需要执行的SQL语句，其他的交给JdbcTemplate完成。

1. 方法
2. update():执行DML语句，增删改操作
3. queryForMap(): 将查询结果集封装为map集合，列名为key，值为value，只能查询一条结果，封装为一个Map集合
4. queryForList(): 将查询结果集封装为list集合，将多条记录封装为map集合，再将map结合装到list集合中
5. query():将查询结果封装为JavaBean对象，参数为RowMapper，我们一般使用BeanPropertyRowMapper实现类，可以完成数据到JavaBean的自动封装：

new BeanPropertyRowMapper<类型>(类型.class)

1. queryForObject():将查询结果封装为一般对象
2. 步骤
3. 导入spring-beans.jar，spring-core.jar，spring-jdbc.jar，sprin-tx.jar，commons-logging,jar包
4. 创建连接池对象DataSource，然后创建JdbcTemplate对象
5. 调用JdbcTemplate()的方法完成增删改查
6. 代码实现

package SprintTets;  
  
import org.springframework.jdbc.core.JdbcTemplate;  
import utils.JDBCUtils;  
  
public class test {  
 public static void main(String[] args) {  
 //创建JDBCTemplate对象  
 JdbcTemplate template = new JdbcTemplate(JDBCUtils.*getDataSource*());  
 //调用方法  
 String sql = "update student set age = 12 where id = 12";  
 int count = template.update(sql);  
 System.*out*.println(count);  
 }  
}