目的：

1 使用原始的jdbc操作数据库

2 重构抽取jdbc的重复代码

3 使用泛型构造通用代码

第一步

引入操作jdbc的maven包

Java.sql.Connection 是由jdk提供

此处仅需要引入jdbc连接mysql的连接包

<**dependency**>  
 <**groupId**>mysql</**groupId**>  
 <**artifactId**>mysql-connector-java</**artifactId**>  
 <**version**>5.1.47</**version**>  
</**dependency**>

第二步

1 加载驱动

2 获取连接

3 创建sql语句对象

4 执行sql

5 返回结果

6 处理结果

第三步

Api分析

**public static void** main(String[] args){  
 Connection conn = **null**;  
 **try** {  
 *// 加载驱动* Class.*forName*(**"com.mysql.jdbc.Driver"**);  
  
 *// 连接数据库获得连接对象* conn = DriverManager.*getConnection*(**"jdbc:mysql://127.0.0.1:3306/test?serverTimezone=UTC&useUnicode=true&characterEncoding=utf8&useSSL=false"**, **"root"**,**"123456"** );  
   
 Statement statement = conn.createStatement(**""**);  
 PreparedStatement ps = conn.prepareStatement(**""**);  
  
 } **catch** (ClassNotFoundException cnfe) {  
 cnfe.printStackTrace();  
 } **catch** (SQLException sqle) {  
 sqle.printStackTrace();  
 }  
}

上面的代码中conn实例可以有两种方式执行sql，

1 createStatement();

2 prepareStatement();

createStatement()处理的是静态sql，无法动态处理参数

prepareStatement()支持处理占位符sql，根据参数的顺序动态处理sql

建议：一般使用prepareStatement();

7 构造数据库连接池

·1 定义连接池抽象类

连接池都有一个共同的特点

配置文件的加载

驱动的加载

获取连接池单例方法

初始化方法

拥有连接对象的存储连接池

最大连接数

初始化连接数

获取连接对象

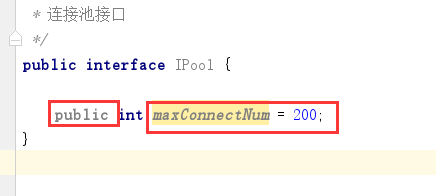
释放连接对象

附加：

总结1

1 interface 接口定义中，支持定义属性字段

需要注意的是：定义的变量必须是public，并且变量必须被初始化



Protected、private修改编译器会报错

2 接口中不能使用 protected、private synchronized 关键字

1. 设计模式

1 单例模式

1.1常见单例模式:

serlvet中 ServletContext、ServletContextConfig

Spring容器中 ApplicationContext

数据库连接池

1.2实现方式

1. 饿汉式：类被加载时就实例化

线程安全

1. 懒汉式: 需要时在实例化

需要处理保证线程

使用synchronize

|  |
| --- |
| **public class** InnerClass {  **private static** InnerClass *instance*;  **public static synchronized** InnerClass getInstance(){  **if**(*instance* == **null**){  *instance* = **new** InnerClass();  }  **return** *instance* }    **public static** InnerClass getInstance01(){  **if**(*instance* == **null**){  **synchronized**(InnerClass.**class**){  **if**(*instance* == **null**){  *instance* = **new** InnerClass();  }  }  }  **return** *instance*;  } } |

使用内部类(推荐)

不需要加锁，同时又保证了线程安全。

因为在加载外部类的时候，首先会初始化内部类，此时内部类只能被加载一次，所以利用内部类中饿汉式处理

|  |
| --- |
| **public class** InnerClass {   **private** InnerClass(){  System.***out***.println(**"1111"**);  }   **public static** InnerClass getInnerClass(){  **return** Inner.***INNER\_CLASS***;  }   **private static class** Inner{  **public static final** InnerClass ***INNER\_CLASS*** = **new** InnerClass();  **private** Inner(){  System.***out***.println(**"222"**);  }  }   **public static void** main(String[] agrs){  *getInnerClass*();  }  } |