## **任务一**

#### 目标：

选择一款关系型数据库和非关系型数据库（建议mysql和redis），使用代码进行增删改查各种操作，并形成模块和接口。

代码：

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

import entity.UserInfo;

public class JDBC // JDBC: Java Database Connectivity

{

Connection conn;

PreparedStatement pst;

ResultSet rst;

//连接数据库

public Connection getConnection(){

String url="jdbc:mysql://localhost:端口/数据库";

String userName="Admin";

String password="12345";

conn=DriverManager.getConnection(url, userName, password);

if(conn!=null){

System.out.println("connection successful");

}

} catch (SQLException e) {

// TODO Auto-generated catch block

System.out.println( "connection fail");

e.printStackTrace();

}

return conn;

}

//数据库查询

public void Query()

{

UserInfo userObj;

String sql= "select \* from userInfo"; //Sql中查询语句

conn=getConnection(); //连接数据库

pst=conn.prepareStatement(sql); //创建Satement

rst=pst.executeQuery(sql); //执行SQL语句

while(rst.next())

{

userObj=new UserInfo();

userObj.setUserId(rst.getID("userId"));

userObj.setUserName(rst.getString("userName"));

userObj.setPassword(rst.getString("password"));

userObj.setRemark(rst.getString("remark"));

System.out.println("uesrName"+u.getUserName());

}

System.out.println(rst.next());

rst.close();

pst.close();

conn.close();

//断开连接

}

@SuppressWarnings("null")

//数据库添加

public int add(UserInfo user)

{

int row=0;

String sql="insert into userInfo(userName,password) values(?,?)";

conn=getConnection(); //连接数据库

pst=conn.prepareStatement(sql); //创建Satement

pst.setString(1, user.getUserName()); //第一个问号用user.getUerName填充

pst.setString(2, user.getPassword()); //第二个同理

row=pst.executeUpdate();

System.out.println(row+ " "+ user.getUserName()+ " " + user.getPassword());

pst.close();

conn.close();

return row;

}

//数据库更新

public int update(UserInfo user)

{

int row=0;

String sql="update userInfo set userName=? ,password=? where userId=?";

conn=getConnection();

pst=conn.prepareStatement(sql);

pst.setString(1, user.getUserName());

pst.setString(2, user.getPassword());

pst.setInt (3, user.getUserId());

row=pst.executeUpdate();

System.out.println(row);

pst.close();

conn.close();

return row;

}

//数据库删除

public int delete(UserInfo user)

{

int row=0;

String sql="delete from userInfo where userId=?";

conn=getConnection();

pst=conn.prepareStatement(sql);

pst.setInt(1, user.getUserId());

row=pst.executeUpdate();

System.out.println(row);

pst.close();

conn.close();

return row;

}

public static void main(String[] args)

{

JDBC link=new JDBC();

link.QuerySql();

UserInfo userObj1=new UserInfo();

userObj1.setUserId(12);

userObj1.setUserName("www");

userObj1.setPassword("1234567");

link.update(userObj1);

}

}

**要点：**

* **思考怎样封装可以使得代码更加友好，更加可用，每使用一种设计模式有额外加分。**
* **怎样操作数据库才能更加高效？每提出一点优化方法，可以额外加分。**

答：

1. 把数据、日志、索引放到不同的I/O设备上，增加读取速度
2. 扩大服务器内存
3. 妥善处理日志问题

* **如果是图片和视频应该如何存储？每提出一种解决方法，可以额外加分。**

答：

直接把文件的存储路径保存在数据库中。

#### 作业格式：

输出代码，简述所用技术和思路。

## **任务二**

#### 目标：

熟悉spring-boot，如各注解的使用场合，项目的配置用什么方式进行管理。将数据库操作进行封装，输出restful风格的接口。进行测试并打包。

实体类：

@Entity

@Table(name="account")

public class Account {

@Id

@GeneratedValue(generator = "IDGenerator")

@GenericGenerator(name = "IDGenerator", strategy = "increment")

private int id;

private String name;

private String now\_money;

public Account(){

}

public Account(String name,String now\_money){

this.name =name;

this.now\_money = now\_money;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getNow\_money() {

return now\_money;

}

public void setNow\_money(String now\_money) {

this.now\_money = now\_money;

}

}

对应service：

@Service

public interface AccountCurd extends JpaRepository<Account,Integer>{

//泛型分别是对应的实体类，该表对应的主键的类型

}

完成CURD的Controller：

@RestController

public class ListController {

@Autowired

AccountCurd accountCurd;

@RequestMapping("/select")

public List<Account> selectAll() {

return accountCurd.findAll();

}

@RequestMapping("/add")

public String addAccount(String name,String now\_money){

Account account = new Account(name,now\_money);

accountCurd.save(account);

return "success";

}

@RequestMapping("/delete")

public String deleteAccount(@RequestParam("id") int id){

accountCurd.delete(id);

return "success";

}

@RequestMapping("/")

public String home(){

return "Hello World";

}

}

**要点：**

* **如果我的某个配置需要临时修改应该怎么办？每提出一点解决方法可额外加分。**
* **Restful风格接口有哪些要求？所有接口都符合restful设计风格可额外加分。**

1、应用的部署设计规范

把网络应用程序放在一个应用服务器上面，然后通过应用服务器的运行、发布、部署，然后形成一个容器，可以让应用程序在里面跑起来，通过发布的接口和电脑的IP地址可以访问到你的应用服务器，然后通过一个应用发布的标识可以访问到你的应用程序。部署的时候也是如此，比如在Tomcat中发布了一个应用名为shop的应用程序，我们就可以用localhost:8080/shop 就可以访问到我们的shop应用程序的默认页面。

2、方法的访问

Restful风格接口从本质上来说是一种接口，包括了很多接口的访问规范。把整个接口的访问通过一个标准转化成了Http请求，我们通过在地址栏输入URL就可以访问到方法并且可以传输数据和接收数据。

在一个类上有一个@Path注解，这个代表通过这个path可以定位到这个类里面来，在类的方法上面还要有一个@Path注解，还有一个请求方式的标识，主要有POST、GET、PUT、DELETE。

3、数据的元操作

即CRUD(增删查改)操作，分别对应于HTTP方法：GET用来获取资源，POST用来新建资源（也可以用于更新资 源），PUT用来更新资源，DELETE用来删除资源，这样就统一了数据操作的接口，仅通过HTTP方法，就可以完成对数据的所有增删查改工作。

4、请求中使用URL定位器

一个类的path即是这个类的访问URL路径，比如有个类为student，他的Path注解为“/api/student” 那么我们在访问的时候就可以写成“http://loaclhost:8080/应用名/api/student”。在里面写方法的时候可以不用path标记，但是必须用类型标记，如@GET，那么在发送请求的时候只要发送一个get类型的请求，访问地址还是“http://loaclhost:8080/应用名/api/student”，就可以直接访问到写的这个方法。

* **如果有很多人访问我的接口，如何保证稳定？没提出一种解决方法可额外加分。**

1、定义接口的时候，要把本质的东西放在接口里，使得接口相对来说保持不变，也就是所谓的接口的永恒。要对所做的系统有细致的分析，尽可能预见出该系统中的变化，抽象出比较好的对象。

2、过载保护，限流，限制调用频率。

#### 作业格式：

输出代码，简述所用技术和思路。

## **任务三**

#### 目标：

选择常用的消息队列进行研究（建议kafka或者rabbitmq），使用java进行操作。

**要点：**

* **思考为什么需要消息队列？并和其他解决方案进行比较，每提出一点有额外加分。**

主要原因是由于在高并发环境下，由于来不及同步处理，请求往往会发生堵塞，比如说，大量的insert，update之类的请求同时到达MySQL，直接导致无数的行锁表锁，甚至最后请求会堆积过多，从而触发too many connections错误。通过使用消息队列，我们可以异步处理请求，从而缓解系统的压力。

解决以下两个问题：

系统解耦：项目开始时，无法确定最终需求，不同进程间，添加一层，实现解耦，方便今后的扩展。

消息缓存：系统中，不同进程处理消息速度不同，MQ，可以实现不同Process之间的缓冲，即，写入MQ的速度可以尽可能地快，而处理消息的速度可以适当调整（或快、或慢）。

#### 作业格式：

输出代码，简述所用技术和思路。