计算机科学与工程学院作业

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 课程名称 | 软件体系结构与分析 | 班级 | 19软件工程2班 | |
| 作业名称 | **小组作业一** | 指导教师 | | 袁晓峰 |
| 小组成员 | **李文杰，李伟新，黄敏** | 组别 | | 第六组 |

（1）体系结构风格：C/S架构

网络视频播放器：

用户使用网络视频播放器手机app为客户端，存放网络视频的网络服务器为服务端，用户通过客户端的视频地址来访问服务端。

1. **表现层：**

显示用户UI操作，有文本框可以输入网络地址，视频播放窗口可以在线播放视频。

1. **业务逻辑层：**

根据用户输入的网络地址，解析地址视频文件，调用视频播放组件在视频窗口播放视频。

1. **数据访问层：**

在网络上的数据服务器，存放着测试用例视频。

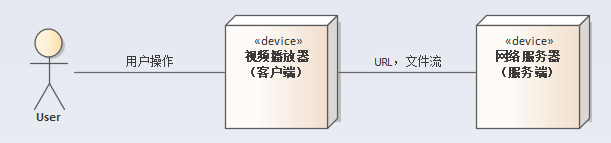


图1：C/S风格用例图



图2：客户端测试图

关键代码：

@Override  
public View onCreateView(LayoutInflater inflater, ViewGroup container,  
 Bundle savedInstanceState) {  
 if(rootView == null){  
 rootView = inflater.inflate(R.layout.*fragment\_home*, container, false);  
 }  
 edittext = rootView.findViewById(R.id.*et\_home*);  
 play = rootView.findViewById(R.id.*bt\_play*);  
 videoView = rootView.findViewById(R.id.*vv\_home*);  
 videoView.setMediaController(new MediaController(rootView.getContext()));  
 videoView.setVisibility(View.*INVISIBLE*);  
 play.setOnClickListener(this);  
 newplayer = rootView.findViewById(R.id.*newplayer*);  
 newplayer.setOnClickListener(new View.OnClickListener() {  
 @Override  
 public void onClick(View v) {  
 Intent intent = new Intent(getContext(),MovieActivity.class);  
 intent.setData(Uri.*parse*(edittext.getText().toString()));  
 startActivity(intent);  
 }  
 });  
 return rootView;  
}  
  
@Override  
public void onClick(View v) {  
 if(videoView!=null && videoView.isPlaying()){  
 videoView.stopPlayback();  
 }  
 videoView.setVideoURI(Uri.*parse*(edittext.getText().toString()));  
 videoView.setVisibility(View.*VISIBLE*);  
 videoView.start();  
}

（2）体系结构风格（B/S架构）

B/S架构即浏览器和[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8" \t "https://baike.baidu.com/item/BS%E6%9E%B6%E6%9E%84/_blank)架构模式

**（1）表示层**

位于三层构架的最上层，与用户直接接触，B/S架构系统表示层主要是Web浏览界面，主要展示用户和后台的交互并将查询结果输出功能，通过Spring Data JPA框架实现基本增删改查等功能，其自定义查询方法将对数据库中数据进行处理，处理后会将处理结果最终反馈到表示层，供用户查阅，表示层的页面由Spring Boot中的Thymeleaf模板进行设计。

**（2）业务逻辑层**

主要利用服务器对具体问题进行逻辑判断与执行等应用功能。用户只需下载浏览器，通过浏览器实现客户端功能，请求服务器端运行。接收到表示层用户指令，通过逻辑层连接数据层，服务器开始对数据库进行相应操作，继而将客户自定义查询结果传输到表示层。

本系统利用Spring Data JPA 框架，可以省略实现持久层业务逻辑的工作，只需声明持久层的接口，就可实现数据访问。

**（3）数据访问层**

本系统利用Springboot依托spring data技术，可以对数据库进行访问，并将操作结果反馈回逻辑层，继而反馈到表示层（浏览器）呈现给用户。利用SpringBoot集成JPA框架，PersonRepository 继承了JPA提供的 JpaRepository。查看JpaRepository可以发现，已经封装好了对数据库基本的增删改查操作，直接使用即可。

本系统使用intelliJ IDEA开发环境＋navicat数据库管理工具+Thymeleaf页面模板+客户端浏览器，工作流程如下图：

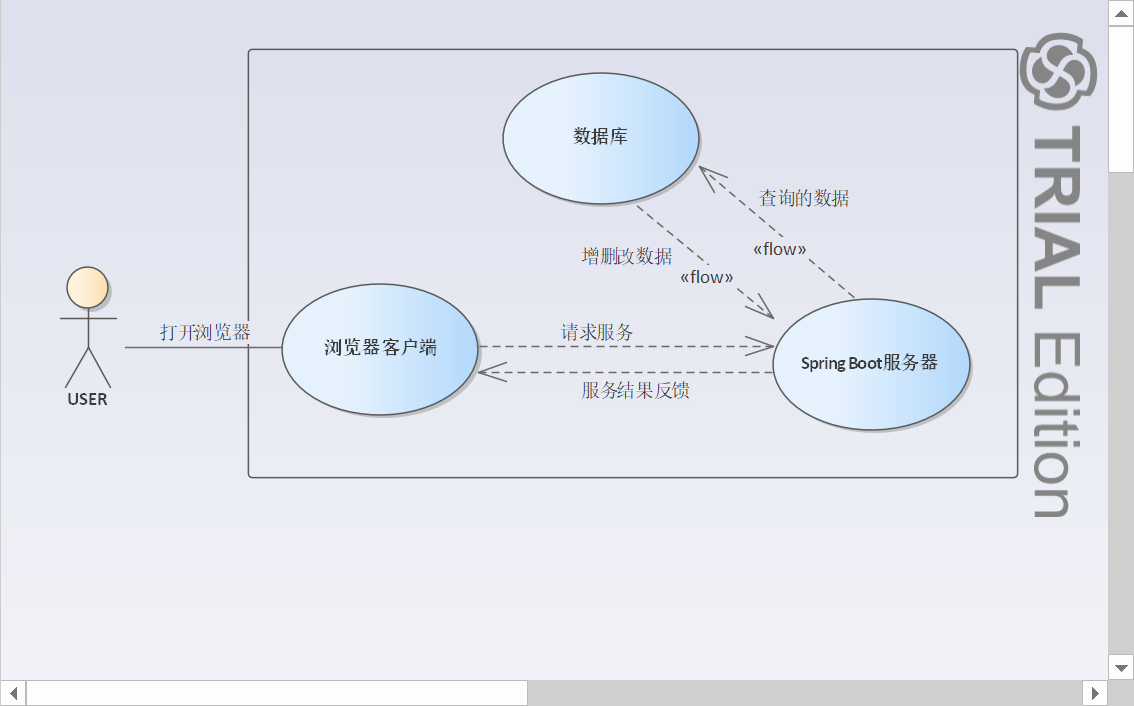


图1spring data jpa用例图



图2客户端浏览器界面

核心编码：

###服务层接口层

public interface UserService {

User findById(Integer id);

List<User> findAll();

User save(User user);

void delete(Integer id);

List<User> findByUpetname(String u\_pet\_name);

List<User> findByUpetnameLike(String u\_pet\_name);

List<User> findByUserialnumberIn(Collection<Integer> user);

User findByIdAndPassword(Integer id,String pwd);

}

###接口实现类

public class UserServiceImpl implements UserService {

@Resource

private UserRepository userRepository;

@Override

public User findById(Integer id) {

return userRepository.getById(id);

}

@Override

public List<User> findAll() {

return userRepository.findAll();

}

@Override

public User save(User user) {

return userRepository.save(user);

}

@Override

public void delete(Integer id) {

userRepository.deleteById(id);

}

@Override

public List<User> findByUpetname(String u\_pet\_name) {

return userRepository.findByUpetname(u\_pet\_name);

}

@Override

public List<User> findByUpetnameLike(String u\_pet\_name) {

return userRepository.findByUpetnameLike("%"+u\_pet\_name+"%");

}

@Override

public List<User> findByUserialnumberIn(Collection<Integer> user) {

return userRepository.findByUserialnumberIn(user);

}

@Override

public User findByIdAndPassword(Integer id, String pwd) {

return userRepository.findByUserialnumberAndUpassword(id,pwd);

}

}

###开发控制层类，将NewTeacherService服务注入到控制层类中，model对象接口将查 询出来的结果设置到该类，前端浏览器其会从对象中获取数据。

@Controller

@RequestMapping("/User")

public class UserController {

@Resource

private UserService userService;

@RequestMapping("/test")

public String test(Model model){

List<User> users = userService.findAll();

model.addAttribute("users",users);

return "User";

}

}

###在html文件使用Thymeleaf标签语言。

<table class="table table-hover table-striped">

<thead>

<tr>

<th>用户名</th>

<th>密码</th>

<th>宠物名</th>

<th>宠物类型</th>

</tr>

</thead>

<tbody>

<tr th:each="user:${users}">

<td th:text="${user.userialnumber}"></td>

<td th:text="${user.upassword}"></td>

<td th:text="${user.upetname}"></td>

<td th:text="${user.upettype}"></td>

</tr>

</tbody>

</table>