springboot后端重点

1 axio封装

```
//定制请求的实例
// import { globals } from "@/main";
//导入axios  npm install axios
import axios from 'axios';


import { ElMessage } from 'element-plus'
//定义一个变量,记录公共的前缀  ,  baseURL
const baseURL = '/api';
const instance = axios.create({ baseURL })

import { useTokenStore } from '@/stores/token.js'
//添加 Axios 请求拦截器 ，用于在请求发送之前执行某些操作
instance.interceptors.request.use(
    (config) => {
        // 第一个参数是一个回调函数，它在请求发送之前被调用。这个函数接受一个 config 对象作为参数，这个对象包含了请求的所有配置。
        //请求前的回调
        //添加token
        const tokenStore = useTokenStore();
        // 检查 tokenStore 是否存在 token。
        if (tokenStore.token) {
            // 如果存在 token，就将它添加到请求头的 Authorization 字段中。这通常用于 API 身份验证。
            config.headers.Authorization = tokenStore.token
        }
        return config;
        // 修改后的配置被返回给 Axios，随后请求将继续进行。
    },
    (err) => {
        //请求错误的回调
        // 如果请求配置过程中发生了错误，它会将错误通过 Promise.reject(err) 传递出去，以便后续的错误处理可以捕获到这个错误。
        Promise.reject(err)
    }
)
/*
这段代码的主要目的是在每次发送请求之前，检查用户是否已经登录（通过 token），如果是，就将 token 添加到请求头中，
以便进行身份验证。当请求发生错误时，也会正常处理错误。这种做法常见于需要认证的 API 请求场景。
*/


/* import {useRouter} from 'vue-router'
const router = useRouter(); */

import router from '@/router'  // '@/router' 将自动导入 `router/index.js` 中的默认导出
//添加 Axios 的响应拦截器，可以在接收到服务器响应后执行某些操作。
instance.interceptors.response.use(
    result => {
```

```
        // 判断业务状态码
        // 如果业务逻辑中的状态码为 0，表示请求成功，返回 result.data
        if (result.data.code === 0) {
            return result.data;
        }

        //操作失败
        //alert(result.data.msg?result.data.msg:'服务异常')
        ElMessage.error(result.data.msg ? result.data.msg : '服务异常')
        //异步操作的状态转换为失败
        return Promise.reject(result.data)

    },
    err => {
        //判断响应状态码,如果为401,则证明未登录,提示请登录,并跳转到登录页面
        if (err.response.status === 401) {
            ElMessage.error('请先登录')
            router.push('/login')
        } else {
            ElMessage.error('服务异常')
        }

        return Promise.reject(err);//异步的状态转化成失败的状态
    }
)

export default instance;
```

2 文件上传功能实现

```
@RestController
public class FileUploadController {

    @Autowired
    private UploadedFilesService uploadedFilesService;


    @Value("${file.upload-dir}")
    private String uploadDir;

    @Value("${python.interpreter}")
    private String pythonInterpreter;

}
```

2.1 uploadurlbyid

FileUploadController

```
    @PostMapping("/uploadurlbyid")
 public Result<String> uploadtest(@RequestParam("file") MultipartFile file,
@RequestParam Integer id) {
        // 检查文件是否为空
        if (file.isEmpty()) {
            return Result.error("File is empty. Please upload a valid file.");
        }
```

```
7
8           // 获取并检查文件名
9           String originalFilename = file.getOriginalFilename();
10          if (originalFilename == null ||
!originalFilename.toLowerCase().endsWith(".nii")) {
11              return Result.error("Invalid file type. Please upload a file with .nii
extension.");
12          }
13
14          // 安全文件名生成
15          String safeFileName = UUID.randomUUID().toString() + ".nii"; // 生成唯一文件名
16          String filePath = uploadDir + safeFileName;
17
18          try {
19              // 确保上传目录存在
20              Files.createDirectories(Paths.get(uploadDir));
21
22              // 保存上传的NII文件
23              file.transferTo(new File(filePath));
24
25              // 执行 Python 脚本并处理输出
26              String sliceFilePath = executePythonScript(filePath);
27              if (sliceFilePath == null) {
28                  return Result.error("Failed to execute Python script.");
29              }
30
31              // 将 JPG 文件以流的形式上传到云端
32              String uploadedUrl = uploadSliceFile(sliceFilePath);
33
34              // 保存上传链接到数据库或做其他处理
35              uploadedFilesService.addFileAndUrlById(id, originalFilename, filePath,
uploadedUrl);
36
37              return Result.success(uploadedUrl);
38          } catch (IOException e) {
39              return Result.error("Failed to upload file: " + e.getMessage());
40          }
41      }
42
43      private String executePythonScript(String filePath) {
44          try {
45              String pythonScriptPath = "script/extract_slice.py"; // 替换为你的脚本路径
46              ProcessBuilder processBuilder = new ProcessBuilder(pythonInterpreter,
pythonScriptPath, filePath);
47              processBuilder.redirectErrorStream(true); // 合并错误与正常输出流
48              Process process = processBuilder.start();
49
50              // 读取脚本输出
51              try (BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()))) {
52                  String line;
53                  while ((line = reader.readLine()) != null) {
54                      System.out.println(line); // 打印输出日志
55                  }
56              }
57
```

```
            // 等待脚本执行完成并检查退出代码
            int exitCode = process.waitFor();
            if (exitCode != 0) {
                throw new IOException("Python script error: exit code " + exitCode);
            }

            // 提取的 JPG 文件路径
            return filePath.replace(".nii", "_middle_slice.jpg");
        } catch (IOException | InterruptedException e) {
            System.out.println("Error executing Python script: ");
            return null;
        }
    }

    private String uploadSliceFile(String sliceFilePath) throws IOException {

        // 将 JPG 文件以流的形式上传到云端
        try (FileInputStream fis = new FileInputStream(sliceFilePath)) {
            String objectName =
sliceFilePath.substring(sliceFilePath.lastIndexOf("\\") + 1);
            return AliOssUtil.uploadFile(objectName, fis);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
```

UploadedFilesServiceImpl

```
@Service
public class UploadedFilesServiceImpl implements UploadedFilesService {
    @Override
    public void addFileAndUrlById(Integer associatedId, String fileName, String
filePath, String uploadedUrl) {
        uploadedFilesMapper.insertFileAndUrl(associatedId, fileName, filePath,
uploadedUrl);
    }
}
```

UploadedFilesMapper

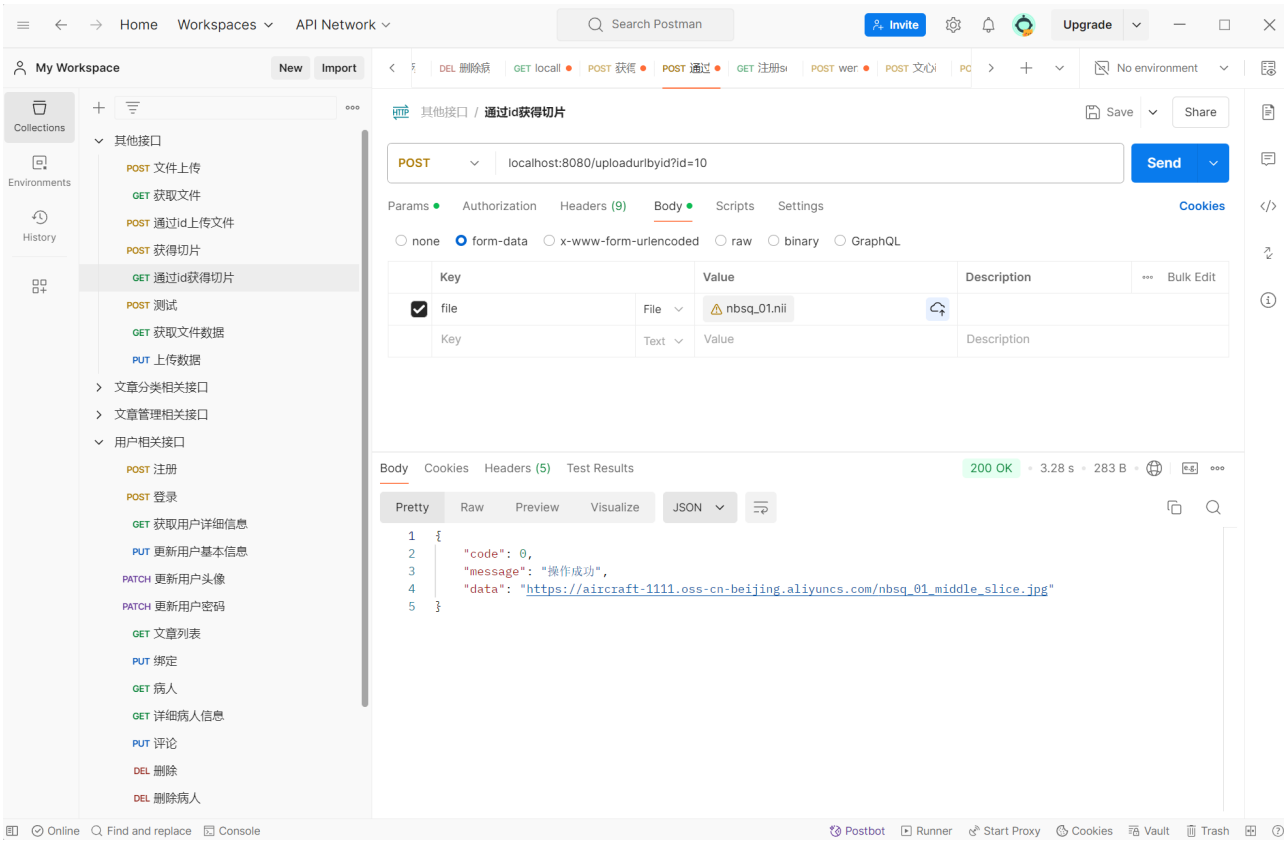
```
@Mapper
public interface UploadedFilesMapper {

    @Insert("INSERT INTO uploaded_files (associated_id, file_name, url, file_path) " +
            "VALUES (#{associatedId}, #{fileName}, #{uploadedUrl}, #{filePath})")
    void insertFileAndUrl(Integer associatedId, String fileName, String filePath,
String uploadedUrl);

}
```

vue

```
<el-upload
 ref="dialogUploadRef"
 :action="`/api/uploadurlbyid?id=${userInfo.id}`"
 :headers="{'Authorization': tokenStore.token}"
 drag
 list-type="picture-card"
>
```



2.2 getImage

FileUploadController

```
@GetMapping("/getImage")
    public Result<List<String>> getImageUrls(@RequestParam Integer id) throws Exception {
        String filePath = uploadedFilesService.getniipathById(id);
        List<String> urls = new ArrayList<>();

        try {
            // 调用Python脚本
            String pythonScriptPath = "script/segimage.py"; // 替换为你的脚本路径
            ProcessBuilder processBuilder = new ProcessBuilder(pythonInterpreter, pythonScriptPath, filePath);

            processBuilder.redirectErrorStream(true); // 将错误流合并到输入流中
            Process process = processBuilder.start();

            BufferedReader reader = new BufferedReader(new InputStreamReader(process.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                System.out.println(line); // 打印输出
```

```
            }

            // 等待脚本执行完成
            int exitCode = process.waitFor();
            if (exitCode != 0) {
                throw new IOException("Python script error: " + exitCode);
            }

            // 处理生成的JPG文件
            String sliceFilePath = filePath.replace(".nii", "_slice.jpg");
            for (int i = 0; i < 10; i++) {
                String newsliceFilePath = sliceFilePath.replace(".jpg", "_" + String.valueOf(i) + ".jpg");

                // 将 JPG 文件以流的形式上传到云端
                try (FileInputStream fis = new FileInputStream(newsliceFilePath)) {
                    // 使用文件名作为对象名
                    String objectName = newsliceFilePath.substring(newsliceFilePath.lastIndexOf("\\") + 1);
                    String uploadedUrl = AliOssUtil.uploadFile(objectName, fis);
                    urls.add(uploadedUrl);
                }
            }
            return Result.success(urls);
        } catch (IOException | InterruptedException e) {
            return Result.error("Failed to upload file: " + e.getMessage());
        }
    }
```

UploadedFilesServiceImpl

```
@Service
public class UploadedFilesServiceImpl implements UploadedFilesService {

    @Override
    public String getniipathById(Integer id) {
        return uploadedFilesMapper.getniipathById(id);
    }

}
```

UploadedFilesMapper

```
@Mapper
public interface UploadedFilesMapper {

    @Select("SELECT file_path FROM uploaded_files WHERE id = #{id}")
    String getniipathById(@Param("id") Integer id);

}
```

@/api/upload.js

```
export const ImagesService = (imgId) => {
    const params = new URLSearchParams(imgId).toString();
    return request.get(`/getImage?${params}`);
};
```

@/stores/images.js

```
const fetchImages = async (imageId) => {
    images.value = [];
    const Id = {
        id: imageId
    };
    try {
        const response = await ImagesService(Id);
        console.log('API Response:', response);  // 调试输出
        if (response.code === 0) {  // 确保响应成功
            const imageUrls = response.data;
            imageUrls.forEach(url => {
                addImage(url);
            });
            console.log('Fetched Infos:', imageList.value);  // 调试输出
        } else {
            console.error('Error fetching images:', response.message);
        }
    } catch (error) {
        console.error('Error fetching images:', error);
    }
};
```

2.3 patientfiles

FileUploadController

```
    @GetMapping("/patientfiles")
    public Result<List<UploadedFile>> getPatientFilesById(@RequestParam Integer id) {
        if (id == null) {
            return Result.error("ID cannot be null or empty");
        }

        try {
            return Result.success(uploadedFilesService.getPatientFilesById(id));
        } catch (Exception e) {
            return Result.error("Failed to get patients files: " + e.getMessage());
        }
    }
```

UploadedFilesServiceImpl

```
@Service
public class UploadedFilesServiceImpl implements UploadedFilesService {

    @Override
    public List<UploadedFile> getPatientFilesById(Integer id) {
        return uploadedFilesMapper.getFilesById(id);
    }

}
```

UploadedFilesMapper

```java
@Mapper
public interface UploadedFilesMapper {

    @Select("SELECT * FROM uploaded_files WHERE associated_id = #{id}")
    List<UploadedFile> getFilesById(Integer id);

}
```

@/api/upload.js

```javascript
//获取用户详细信息
export const filesNameService = (patientId) => {
    // 将 patientId 对象转化为查询参数字符串
    const params = new URLSearchParams(patientId).toString();
    return request.get(`/patientfiles?${params}`);
}

```

@/stores/files.js

```javascript
const fetchInfos = async (userid) => {
    const Id = {
        id: userid
    };
    console.log("ID:", Id);
    try {
        const response = await filesNameService(Id);
        console.log('API Response:', response)  // 调试输出
        if (response.code === 0) {
            infos.value = response.data
            console.log(response.data);
        } else {
            console.error('Error fetching files:', response.message);
        }
    } catch (error) {
        console.error('Error fetching filesName:');
    }
}
```

2.4 delete

FileUploadController

```java
    @DeleteMapping("/delete")
    public Result<String> deleteFile(@RequestBody Map<String, Integer> params) {
        Integer fileId = params.get("fileId");

        try {
            // 获取文件信息
            UploadedFile file = uploadedFilesService.getFileById(fileId);
            if (file == null) {
                return Result.error("File not found");
            }

            uploadedFilesService.deleteFile(fileId);
            boolean isDeleted = deleteFileWithPython(file.getFilePath());
            if (!isDeleted) {
```

```
                return Result.error("Failed to delete file from disk");
            }

            return Result.success("File deleted successfully");
        } catch (Exception e) {
            return Result.error("Failed to delete file: " + e.getMessage());
        }
    }

    private boolean deleteFileWithPython(String filePath) {
        try {
            ProcessBuilder pb = new ProcessBuilder("python", "script/delete_file.py", filePath);
            Process process = pb.start();
            int exitCode = process.waitFor();
            return exitCode == 0;
        } catch (Exception e) {
            return false;
        }
    }
```

UploadedFilesServiceImpl

```
@Service
public class UploadedFilesServiceImpl implements UploadedFilesService {

    @Override
    public void deleteFile(Integer fileId) {
        uploadedFilesMapper.deleteFile(fileId);
    }

}
```

UploadedFilesMapper

```
@Mapper
public interface UploadedFilesMapper {

    @Delete("DELETE FROM uploaded_files WHERE id = #{fileId}")
    void deleteFile(Integer fileId);

}
```

@/api/upload.js

```
export const deleteFileService = (data) => {
    return request.delete("/delete", {
        data: data
    });
};
```

@\components\user.vue

```
const deleteItem = async (id, index) => {
  const data = {
    fileId: id
  };

```

```
  try {
    const result = await deleteFileService(data);
    if (result.code === 0) {
      imagesresult.value.splice(index, 1);
      console.log(result.message);
    } else {
      console.error('Error deleting file:', result.message);
    }
  } catch (error) {
    console.error('Error deleting file:', error);
  }
  getFilesName();
};
```

2.5 patientdetails

DoctorPatientController

```
@RestController
@RequestMapping("/doctor-patients")
@Validated
public class DoctorPatientController {
    @Autowired
    private DoctorPatientService doctorPatientService;

    @GetMapping("/patientdetails")
    public Result<List<User>> getPatientDetailsByDoctorId() {
        return Result.success(doctorPatientService.getPatientDetailsByDoctorId());

}

```

DoctorPatientServiceImpl

```
@Service
public class DoctorPatientServiceImpl implements DoctorPatientService {
    public List<User> getPatientDetailsByDoctorId() {
        Map<String,Object> map = ThreadLocalUtil.get();
        Integer doctorId = (Integer) map.get("id");
        // 获取病人 ID 列表
        List<Integer> patientIds =
doctorPatientMapper.findPatientIdsByDoctorId(doctorId);
        // 获取病人详细信息
        return userMapper.findUsersByIds(patientIds);
    }
}
```

DoctorPatientMapper

```
@Mapper
public interface DoctorPatientMapper {

    @Select("SELECT patient_id FROM doctor_patient WHERE doctor_id = #{doctorId}")
    List<Integer> findPatientIdsByDoctorId(@Param("doctorId") Integer doctorId);

}
```

UserMapper

```java
@Mapper
public interface UserMapper {

    @Select("<-script->" +
            "SELECT * FROM user WHERE id IN " +
            "<-foreach item='id' collection='ids' open='(' separator=',' close=')'->" +
            "#{id}" +
            "<-/foreach->" +
            "<-/script->")
    List<User> findUsersByIds(@Param("ids") List<Integer> ids);

}
```