

# SpringBootWeb案例

前面我们已经实现了员工信息的条件分页查询以及删除操作。关于员工管理的功能，还有两个需要实现：

- 新增员工
- 修改员工



首先我们先完成"新增员工"的功能开发，再完成"修改员工"的功能开发。而在"新增员工"中，需要添加头像，而头像需要用到"文件上传"技术。当整个员工管理功能全部开发完成之后，我们再通过配置文件来优化一些内容。

综上所述，我们今天的课程内容包含以下四个部分：

- 新增员工
- 文件上传
- 修改员工
- 配置文件

## 1. 新增员工

## 1.1 需求

新增员工

\* 用户名

请输入用户名，2-20字符，不可重复

\* 员工姓名

请输入员工姓名，2-10个字

\* 性 别

请选择

图 像



职 位

请选择

入职日期

2022-07-22

归属部门

请选择

保存

取消

在新增用户时，我们需要保存用户的基本信息，并且还需要上传的员工图片，目前我们先完成第一步操作，保存用户的基本信息。

## 1.2 接口文档

我们参照接口文档来开发新增员工功能

- 基本信息

- 1 请求路径：/emps
- 2
- 3 请求方式：POST
- 4
- 5 接口描述：该接口用于添加员工的信息

- 请求参数

参数格式：application/json

参数说明：

名称	类型	是否必须	备注
username	string	必须	用户名
name	string	必须	姓名
gender	number	必须	性别，说明：1 男，2 女
image	string	非必须	图像
deptId	number	非必须	部门id
entrydate	string	非必须	入职日期
job	number	非必须	职位，说明：1 班主任, 2 讲师, 3 学工主管, 4 教研主管, 5 咨询师

请求数据样例：

```
1  {
2    "image": "https://web-framework.oss-cn-hangzhou.aliyuncs.com/2022-09-03-07-37-38222.jpg",
3    "username": "linpingzhi",
4    "name": "林平之",
5    "gender": 1,
6    "job": 1,
7    "entrydate": "2022-09-18",
8    "deptId": 1
9  }
```

• 响应数据

参数格式：application/json

参数说明：

参数名	类型	是否必须	备注
code	number	必须	响应码, 1 代表成功, 0 代表失败
msg	string	非必须	提示信息
data	object	非必须	返回的数据

响应数据样例:

```

1  {
2      "code":1,
3      "msg":"success",
4      "data":null
5  }
```

### 1.3 思路分析

新增员工的具体的流程:



接口文档规定:

- 请求路径: /emps
- 请求方式: POST
- 请求参数: Json格式数据
- 响应数据: Json格式数据

问题1: 如何限定请求方式是POST?

```
1  @PostMapping
```

问题2: 怎么在controller中接收json格式的请求参数?

```
1  @RequestBody //把前端传递的json数据填充到实体类中
```

## 1.4 功能开发

### EmpController

```
1  @Slf4j
2  @RestController
3  @RequestMapping("/emps")
4  public class EmpController {
5
6      @Autowired
7      private EmpService empService;
8
9      //新增
10     @PostMapping
11     public Result save(@RequestBody Emp emp) {
12         //记录日志
13         log.info("新增员工, emp:{}", emp);
14         //调用业务层新增功能
15         empService.save(emp);
16         //响应
17         return Result.success();
18     }
19
20     //省略...
21 }
```

### EmpService

```
1  public interface EmpService {
2
3      /**
4       * 保存员工信息
5       * @param emp
6       */
7      void save(Emp emp);
8
9      //省略...
10 }
11
```

### EmpServiceImpl

```
1  @Slf4j
2  @Service
3  public class EmpServiceImpl implements EmpService {
4      @Autowired
```

```

5     private EmpMapper empMapper;
6
7     @Override
8     public void save(Emp emp) {
9         //补全数据
10        emp.setCreateTime(LocalDateTime.now());
11        emp.setUpdateTime(LocalDateTime.now());
12        //调用添加方法
13        empMapper.insert(emp);
14    }
15
16    //省略...
17 }

```

### EmpMapper

```

1  @Mapper
2  public interface EmpMapper {
3      //新增员工
4      @Insert("insert into emp (username, name, gender, image, job,
5      entrydate, dept_id, create_time, update_time) " +
6      "values ({username}, #{name}, #{gender}, #{image}, #
7      {job}, #{entrydate}, #{deptId}, #{createTime}, #{updateTime});")
8      void insert(Emp emp);
9
10     //省略...
11 }

```

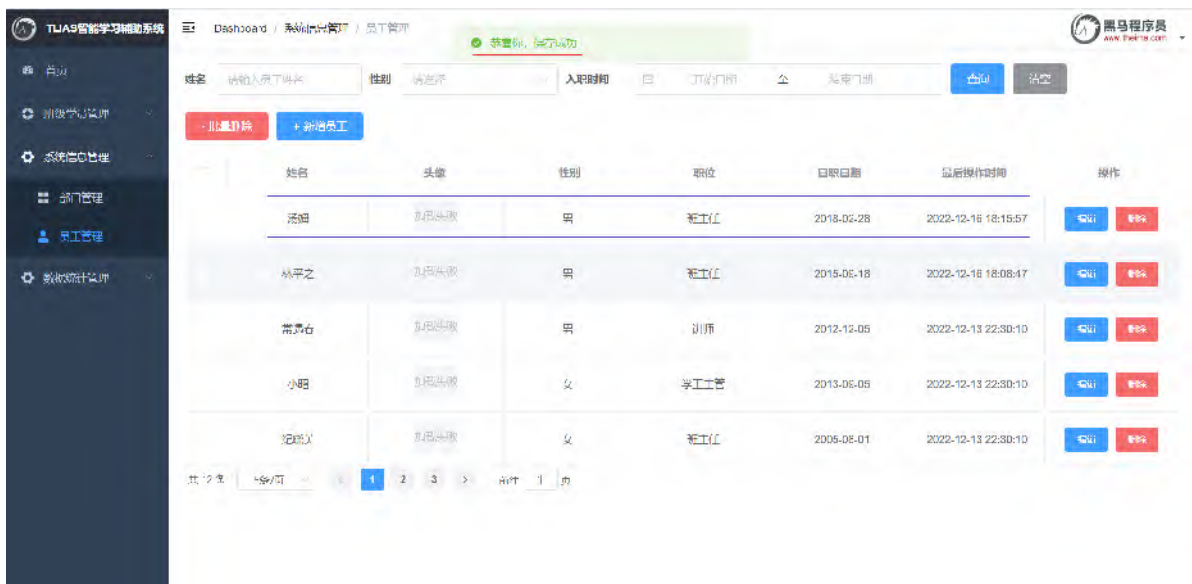
## 1.5 功能测试

代码开发完成后，重启服务器，打开Postman发送 POST 请求，请求路径：<http://localhost:8080/emp>s



## 1.6 前后端联调

功能测试通过后，我们再进行通过打开浏览器，测试后端功能接口：



## 2. 文件上传

在我们完成的新增员工功能中，还存在一个问题：没有头像（图片缺失）

<div>- 批量删除</div> <div>+ 新增员工</div>					
<input type="checkbox"/>	姓名	头像	性别	职位	入职日期
<input type="checkbox"/>	汤姆	加载失败	男	班主任	2018-02-28
<input type="checkbox"/>	林平之	加载失败	男	班主任	2015-09-18
<input type="checkbox"/>	常遇春	加载失败	男	讲师	2012-12-05

上述问题，需要我们通过文件上传技术来解决。下面我们就进入到文件上传技术的学习。

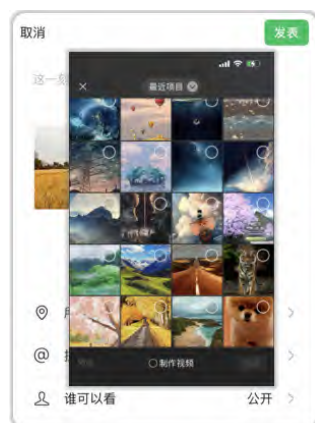
文件上传技术这块我们主要讲解三个方面：首先我们先对文件上传做一个整体的介绍，接着再学习文件上传的本地存储方式，最后学习云存储方式。

接下来我们就先来学习下什么是文件上传。

### 2.1 简介

文件上传，是指将本地图片、视频、音频等文件上传到服务器，供其他用户浏览或下载的过程。

文件上传在项目中应用非常广泛，我们经常发微博、发微信朋友圈都用到了文件上传功能。



在我们的案例中，在新增员工的时候，要上传员工的头像，此时就会涉及到文件上传的功能。在进行文件上传时，我们点击加号或者是点击图片，就可以选择手机或者是电脑本地的图片文件了。当我们选择了某一个图片文件之后，这个文件就会上传到服务器，从而完成文件上传的操作。



想要完成文件上传这个功能需要涉及到两个部分：

1. 前端程序
2. 服务端程序

我们先来看看在前端程序中要完成哪些代码：

```
1 <form action="/upload" method="post" enctype="multipart/form-data">
2     姓名：<input type="text" name="username"><br>
3     年龄：<input type="text" name="age"><br>
4     头像：<input type="file" name="image"><br>
5     <input type="submit" value="提交">
6 </form>
```

上传文件的原始form表单，要求表单必须具备以下三点（上传文件页面三要素）：

- 表单必须有file域，用于选择要上传的文件

```
1 <input type="file" name="image"/>
```

- 表单提交方式必须为POST

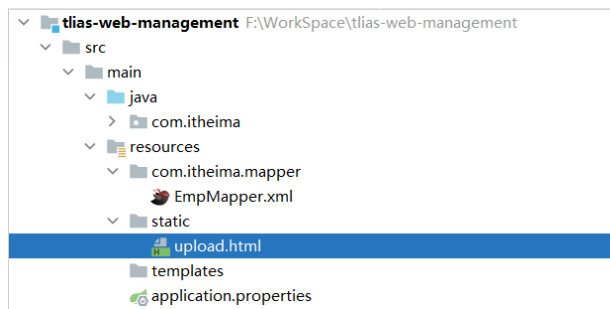
通常上传的文件会比较大，所以需要使用 POST 提交方式

- 表单的编码类型enctype必须要设置为：multipart/form-data

普通默认的编码格式是不适合传输大型的二进制数据的，所以在文件上传时，表单的编码格式必须设置为multipart/form-data

前端页面的3要素我们了解后，接下来我们就来验证下所讲解的文件上传3要素。

在提供的"课程资料"中有一个名叫"文件上传"的文件夹，直接将里的"upload.html"文件，复制到springboot项目工程下的static目录里面。

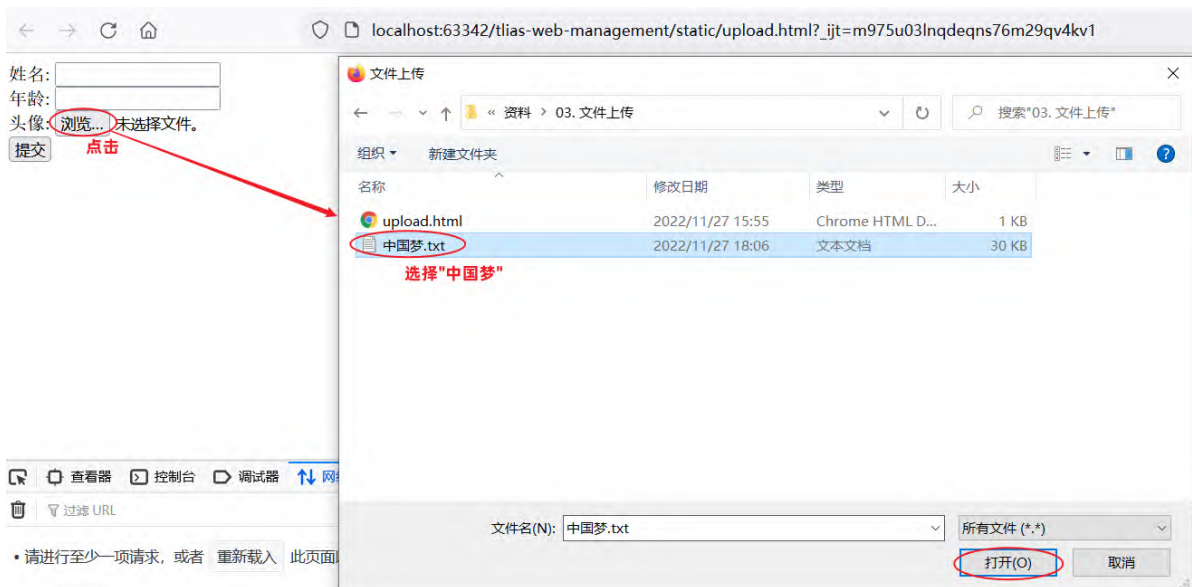


下面我们来验证：删除form表单中enctype属性值，会是什么情况？

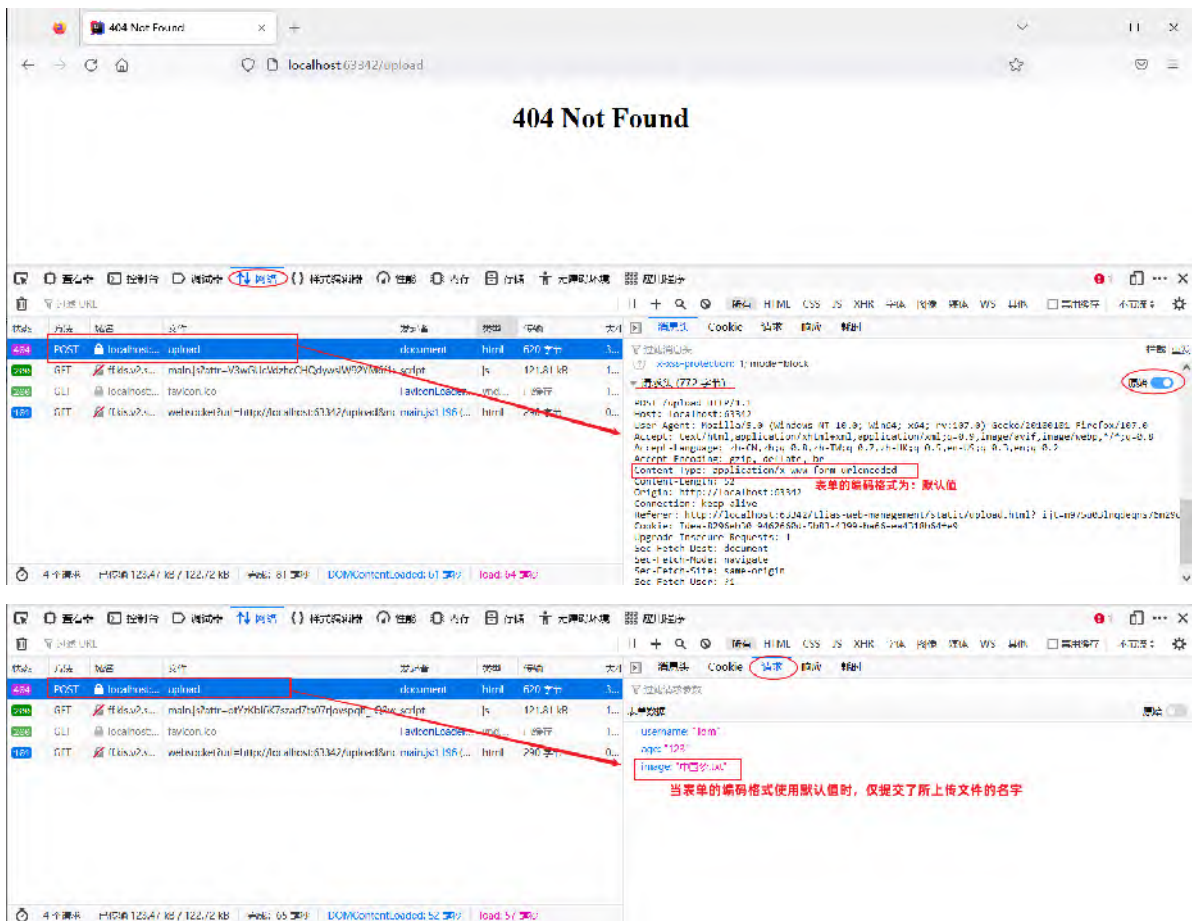
1. 在IDEA中直接使用浏览器打开upload.html页面



2. 选择要上传的本地文件



3. 点击"提交"按钮，进入到开发者模式观察

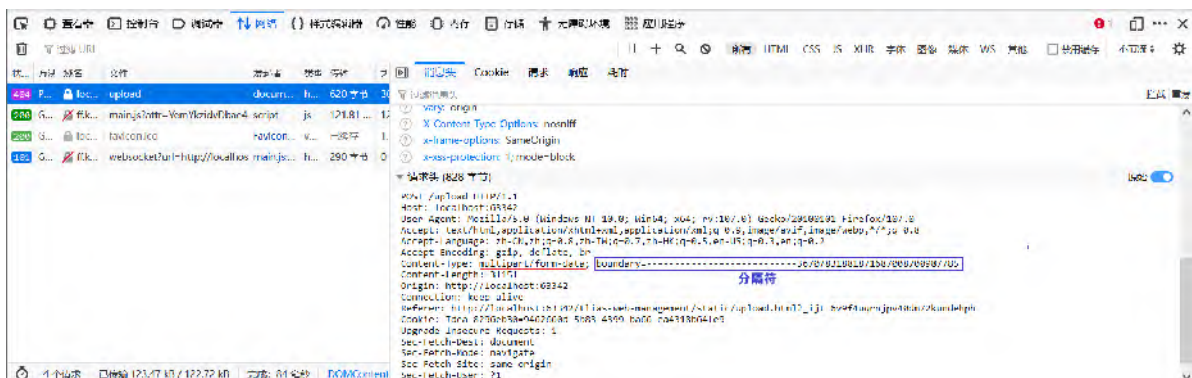


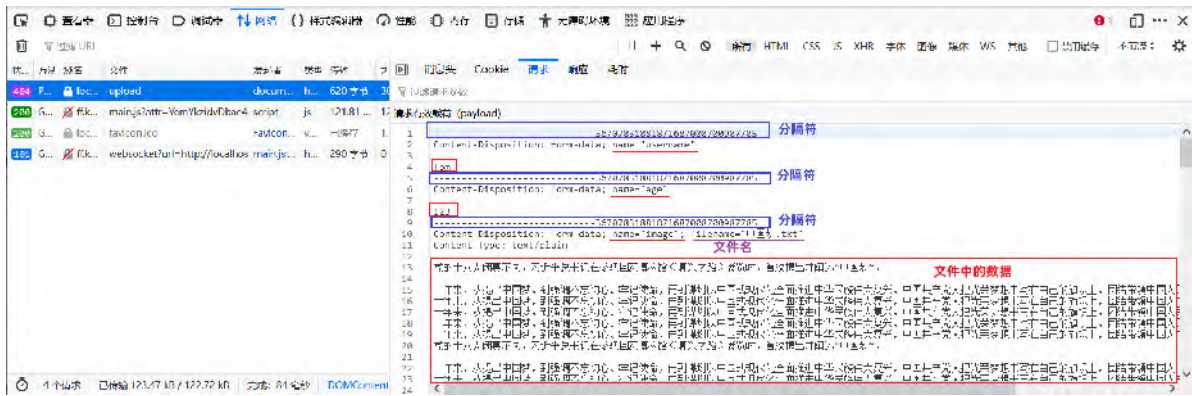
我们再来验证：设置form表单中enctype属性值为multipart/form-data，会是什么情况？

```

1 <form action="/upload" method="post" enctype="multipart/form-data">
2     姓名: <input type="text" name="username"><br>
3     年龄: <input type="text" name="age"><br>
4     头像: <input type="file" name="image"><br>
5     <input type="submit" value="提交">
6 </form>

```





知道了前端程序中需要设置上传文件页面三要素，那我们的后端程序又是如何实现的呢？

- 首先在服务端定义这么一个controller，用来进行文件上传，然后在controller当中定义一个方法来处理 `/upload` 请求
- 在定义的方法中接收提交过来的数据（方法中的形参名和请求参数的名字保持一致）
  - 用户名: `String name`
  - 年龄: `Integer age`
  - 文件: `MultipartFile image`

Spring中提供了一个API: `MultipartFile`，使用这个API就可以来接收到上传的文件

```
<form action="/upload" method="post" enctype="multipart/form-data">
  姓名: <input type="text" name="username"><br>
  年龄: <input type="text" name="age"><br>
  头像: <input type="file" name="image"><br>
  <input type="submit" value="提交">
</form>
```

前端页面三要素

```
@RestController
public class UploadController {
    @PostMapping("/upload")
    public Result upload(String username, Integer age, MultipartFile image){
        return Result.success();
    }
}
```

和表单中的名字保持一致

服务端接收文件

问题：如果表单的名字和方法中形参名不一致，该怎么办？

- 1 `public Result upload(String username,`
- 2 `Integer age,`
- 3 `MultipartFile file)` //file形参名和请求参数名image不一致

解决：使用`@RequestParam`注解进行参数绑定

```

1 public Result upload(String username,
2                       Integer age,
3                       @RequestParam("image") MultipartFile
4                       file)

```

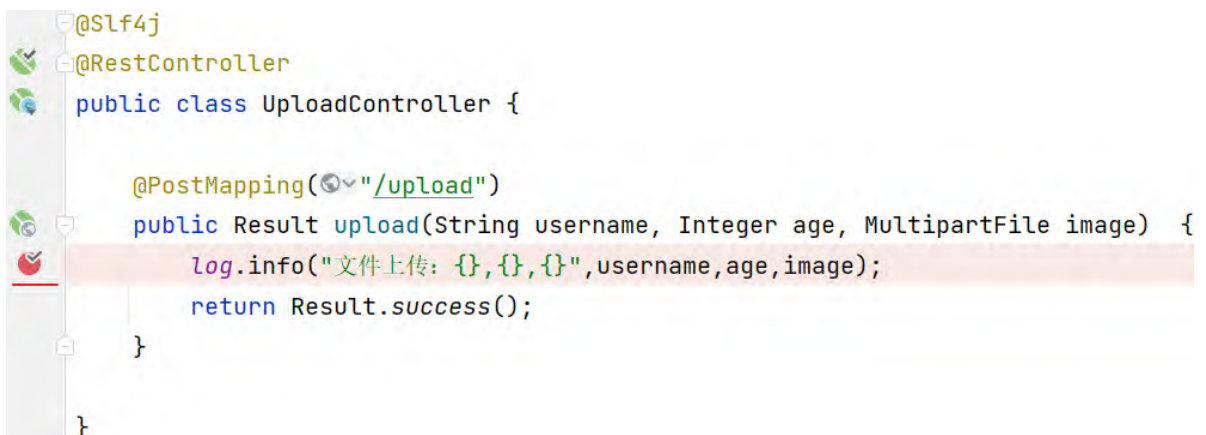
**UploadController代码:**

```

1  @Slf4j
2  @RestController
3  public class UploadController {
4
5      @PostMapping("/upload")
6      public Result upload(String username, Integer age, MultipartFile
7      image) {
8          log.info("文件上传: {}, {}, {}", username, age, image);
9          return Result.success();
10     }
11 }

```

后端程序编写完成之后，打个断点，以debug方式启动SpringBoot项目



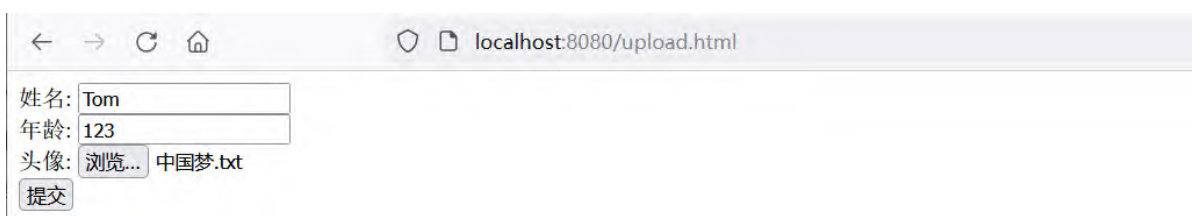
```

@Slf4j
@RestController
public class UploadController {

    @PostMapping("/upload")
    public Result upload(String username, Integer age, MultipartFile image) {
        log.info("文件上传: {}, {}, {}", username, age, image);
        return Result.success();
    }
}

```

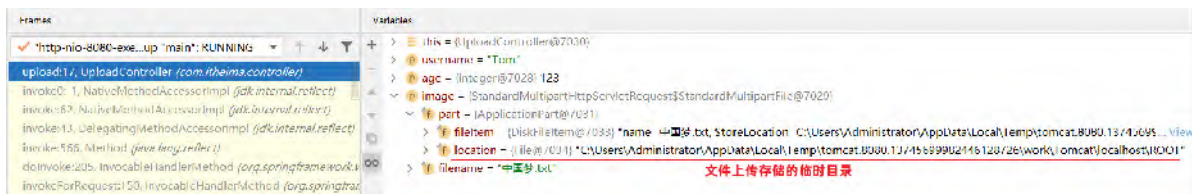
打开浏览器输入: <http://localhost:8080/upload.html> , 录入数据并提交



中国梦.txt

通过后端程序控制台可以看到，上传的文件是存放在一个临时目录





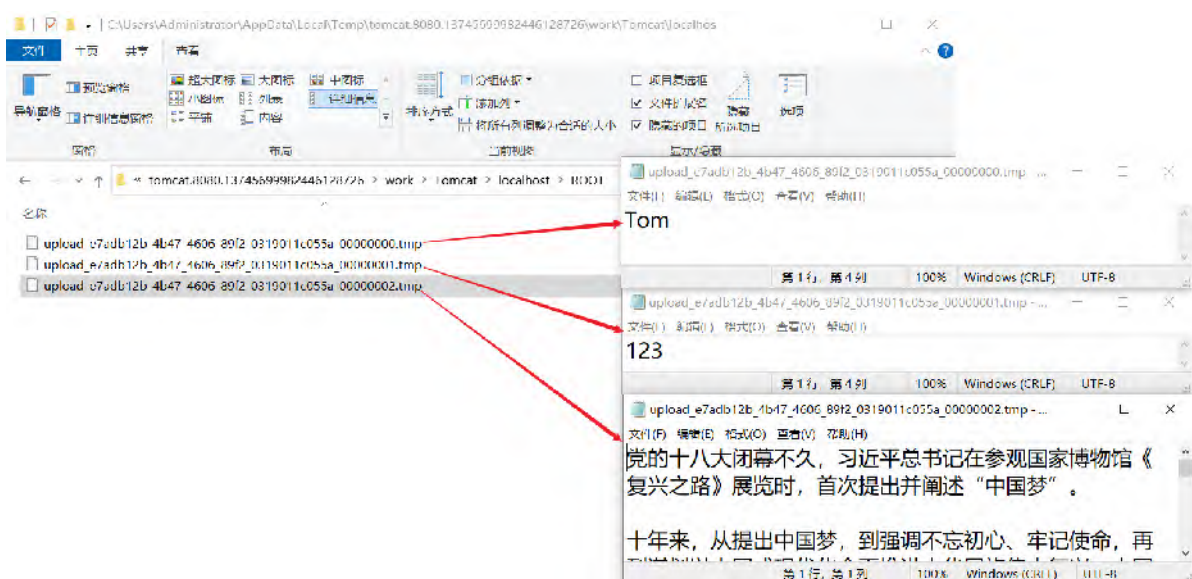
打开临时目录可以看到以下内容：

tomcat.8080.13745699982446128726 > work > Tomcat > localhost > ROOT

名称

- ☐ upload\_e7adb12b\_4b47\_4606\_89f2\_0319011c055a\_00000000.tmp
- ☐ upload\_e7adb12b\_4b47\_4606\_89f2\_0319011c055a\_00000001.tmp
- ☐ upload\_e7adb12b\_4b47\_4606\_89f2\_0319011c055a\_00000002.tmp

表单提交的三项数据（姓名、年龄、文件），分别存储在不同的临时文件中：



当我们程序运行完毕之后，这个临时文件会自动删除。

所以，我们如果想要实现文件上传，需要将这个临时文件，要转存到我们的磁盘目录中。

## 2.2 本地存储

前面我们已分析了文件上传功能前端和后端的基础代码实现，文件上传时在服务端会产生一个临时文件，请求响应完成之后，这个临时文件被自动删除，并没有进行保存。下面呢，我们就需要完成将上传的文件保存在服务器的本地磁盘上。

代码实现：

1. 在服务器本地磁盘上创建images目录，用来存储上传的文件（例：E盘创建images目录）
2. 使用MultipartFile类提供的API方法，把临时文件转存到本地磁盘目录下

MultipartFile 常见方法：

- `String getOriginalFilename();` //获取原始文件名
- `void transferTo(File dest);` //将接收的文件转存到磁盘文件中
- `long getSize();` //获取文件的大小, 单位: 字节
- `byte[] getBytes();` //获取文件内容的字节数组
- `InputStream getInputStream();` //获取接收到的文件内容的输入流

```

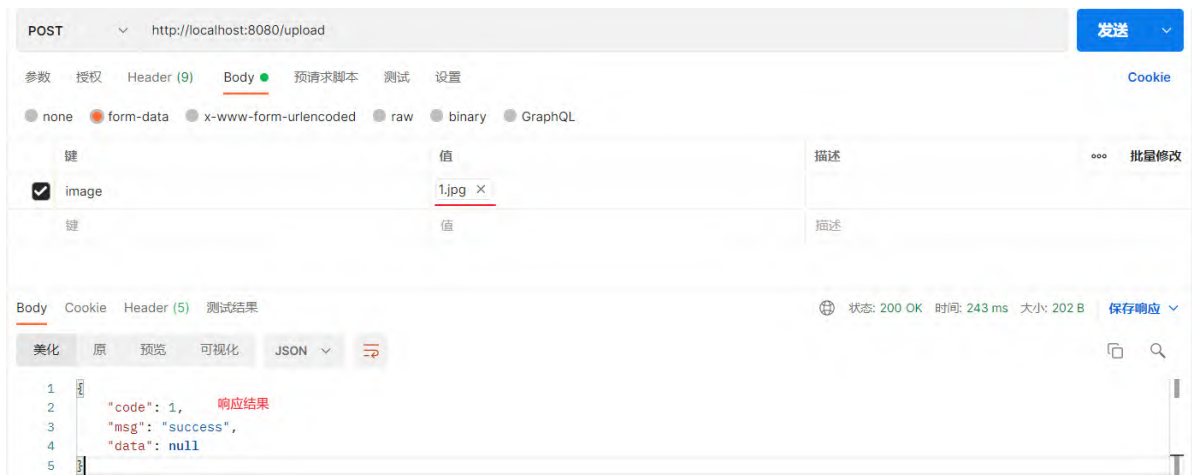
1  @Slf4j
2  @RestController
3  public class UploadController {
4
5      @PostMapping("/upload")
6      public Result upload(String username, Integer age, MultipartFile
image) throws IOException {
7          log.info("文件上传: {}, {}, {}", username, age, image);
8
9          //获取原始文件名
10         String originalFilename = image.getOriginalFilename();
11
12         //将文件存储在服务器的磁盘目录
13         image.transferTo(new File("E:/images/"+originalFilename));
14
15         return Result.success();
16     }
17
18 }

```

利用postman测试:

注意: 请求参数名和controller方法形参名保持一致





通过postman测试，我们发现文件上传是没有问题的。但是由于我们是使用原始文件名作为所上传文件的存储名字，当我们再次上传一个名为1.jpg文件时，发现会把之前已经上传成功的文件覆盖掉。

解决方案：保证每次上传文件时文件名都唯一的（使用UUID获取随机文件名）

```
1  @Slf4j
2  @RestController
3  public class UploadController {
4
5      @PostMapping("/upload")
6      public Result upload(String username, Integer age, MultipartFile
image) throws IOException {
7          log.info("文件上传: {}, {}, {}", username, age, image);
8
9          //获取原始文件名
10         String originalFilename = image.getOriginalFilename();
11
12         //构建新的文件名
13         String extname =
originalFilename.substring(originalFilename.lastIndexOf(".")); //文件
扩展名
14         String newFileName = UUID.randomUUID().toString()+extname; //
随机名+文件扩展名
15
16         //将文件存储在服务器的磁盘目录
17         image.transferTo(new File("E:/images/"+newFileName));
18
19         return Result.success();
20     }
21
22 }
```

在解决了文件名唯一性的问题后，我们再次上传一个较大的文件(超出1M)时发现，后端程序报错：



```
apache.tomcat.util.http.fileupload.impl.FileSizeLimitExceededException Create breakpoint : The field image exceeds its maximum permitted size of 1048576 bytes.
at java.io.FilterInputStream.read(FilterInputStream.java:107) ~[na:1.8.0_171] <5 internal lines>
at org.springframework.web.multipart.support.StandardMultipartHttpServletRequest.parseRequest(StandardMultipartHttpServletRequest.java:95) ~[spring-web-5.3.22.jar:5.3.22]
at org.springframework.web.multipart.support.StandardMultipartHttpServletRequest.<init>(StandardMultipartHttpServletRequest.java:88) ~[spring-web-5.3.22.jar:5.3.22]
at org.springframework.web.multipart.support.StandardServletMultipartResolver.resolveMultipart(StandardServletMultipartResolver.java:122) ~[spring-web-5.3.22.jar:5.3.22]
at javax.servlet.http.HttpServlet.service(HttpServlet.java:681) ~[tomcat-embed-core-9.0.65.jar:4.0.FR] <1 internal line>
at javax.servlet.http.HttpServlet.service(HttpServlet.java:764) ~[tomcat-embed-core-9.0.65.jar:4.0.FR] <33 internal lines>
```

报错原因呢是因为：在SpringBoot中，文件上传时默认单个文件最大大小为1M

那么如果需要上传大文件，可以在application.properties进行如下配置：

```
1 #配置单个文件最大上传大小
2 spring.servlet.multipart.max-file-size=10MB
3
4 #配置单个请求最大上传大小(一次请求可以上传多个文件)
5 spring.servlet.multipart.max-request-size=100MB
```

到时此，我们文件上传的本地存储方式已完成了。但是这种本地存储方式还存在一问题：



如果直接存储在服务器的磁盘目录中，存在以下缺点：

- 不安全：磁盘如果损坏，所有的文件就会丢失
- 容量有限：如果存储大量的图片，磁盘空间有限（磁盘不可能无限扩容）
- 无法直接访问

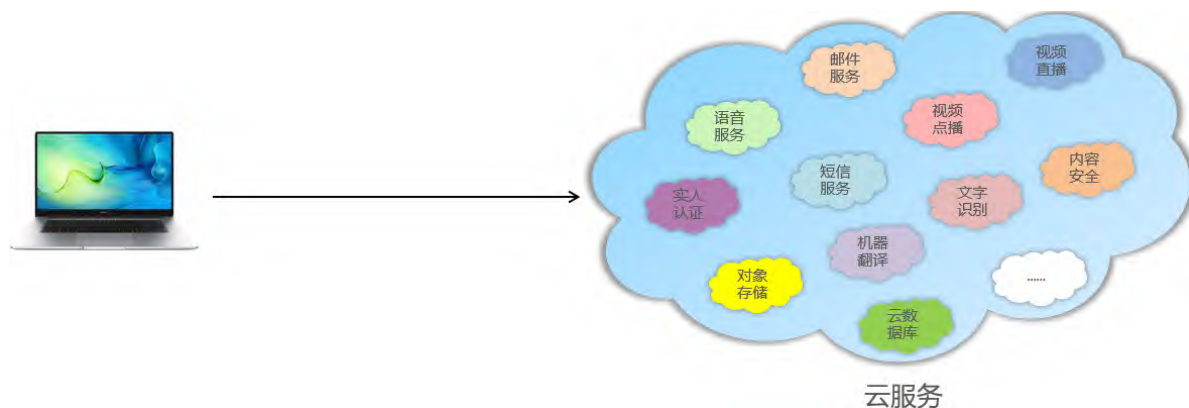
为了解决上述问题呢，通常有两种解决方案：

- 自己搭建存储服务器，如：fastDFS、MinIO
- 使用现成的云服务，如：阿里云，腾讯云，华为云

## 2.3 阿里云OSS

### 2.3.1 准备

阿里云是阿里巴巴集团旗下全球领先的云计算公司，也是国内最大的云服务提供商。



云服务指的就是通过互联网对外提供的各种各样的服务，比如像：语音服务、短信服务、邮件服务、视频直播服务、文字识别服务、对象存储服务等等。

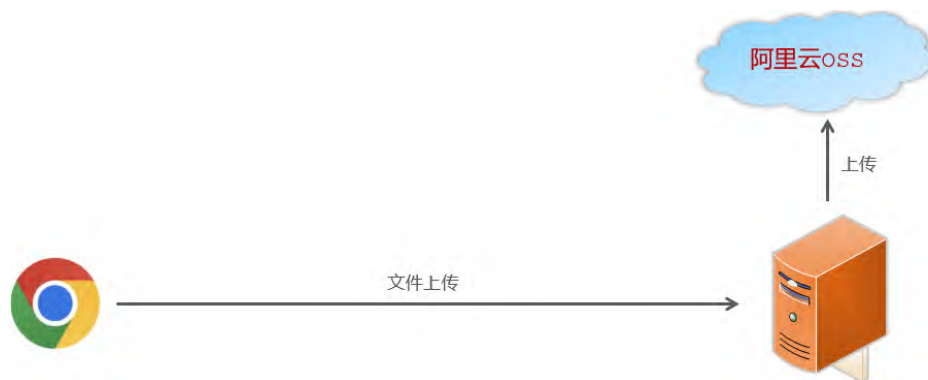
当我们在项目开发时需要用到某个或某些服务，就不需要自己来开发了，可以直接使用阿里云提供好的这些现成服务就可以了。比如：在项目开发当中，我们要实现一个短信发送的功能，如果我们项目组自己实现，将会非常繁琐，因为你需要和各个运营商进行对接。而此时阿里云完成了和三大运营商对接，并对外提供了一个短信服务。我们项目组只需要调用阿里云提供的短信服务，就可以很方便的来发送短信了。这样就降低了我们项目的开发难度，同时也提高了项目的开发效率。（大白话：别人帮我们实现好了功能，我们只要调用即可）

云服务提供商给我们提供的软件服务通常是需要收取一部分费用的。

阿里云对象存储OSS (Object Storage Service)，是一款海量、安全、低成本、高可靠的云存储服务。使用OSS，您可以通过网络随时存储和调用包括文本、图片、音频和视频等在内的各种文件。



在我们使用了阿里云OSS对象存储服务之后，我们的项目当中如果涉及到文件上传这样的业务，在前端进行文件上传并请求到服务端时，在服务器本地磁盘当中就不需要再来存储文件了。我们直接将接收到的文件上传到OSS，由OSS帮我们存储和管理，同时阿里云的OSS存储服务还保障了我们所存储内容的安全可靠。



那我们学习使用这类云服务，我们主要学习什么呢？其实我们主要学习的是如何在项目当中来使用云服务完成具体的业务功能。而无论使用什么样的云服务，阿里云也好，腾讯云、华为云也罢，在使用第三方的服务时，操作的思路都是一样的。



SDK: Software Development Kit 的缩写，软件开发工具包，包括辅助软件开发的依赖（jar包）、代码示例等，都可以叫做SDK。

简单说，sdk中包含了我们使用第三方云服务时所需要的依赖，以及一些示例代码。我们可以参照sdk所提供的示例代码就可以完成入门程序。

第三方服务使用的通用思路，我们做一个简单介绍之后，接下来我们就来介绍一下我们当前要使用的阿里云OSS对象存储服务具体的使用步骤。



Bucket: 存储空间是用户用于存储对象（Object，就是文件）的容器，所有的对象都必须隶属于某个存储空间。

下面我们根据之前介绍的使用步骤，完成准备工作：

1. 注册阿里云账户（注册完成后需要实名认证）
2. 注册完账号之后，就可以登录阿里云



### 3. 通过控制台找到对象存储OSS服务



如果是第一次访问，还需要开通对象存储服务OSS



### 4. 开通oss服务之后，就可以进入到阿里云对象存储的控制台





⚠ 注意：Bucket 创建成功后，您所选择的 **存储类型**、**区域**、**存储冗余类型** 不支持变更。

\* Bucket 名称

web-397

7/63 ✓

\* 地域

华东1（杭州）

相同区域内的产品内网可以互通；订购后不支持更换区域，请谨慎选择。

Endpoint

oss-cn-hangzhou.aliyuncs.com

所属资源组

请选择

存储类型

标准存储

低频访问存储

归档存储

冷归档存储

标准：高可靠、高可用、高性能，数据会经常被访问到。

[如何选择适合您的存储类型？](#)

HDFS服务 New

目前在邀测使用中，[申请使用](#)

HDFS服务说明请参考[HDFS服务](#)，开启后暂不支持关闭。

同城冗余存储 Hot



未开通

OSS 将您的数据以冗余的方式存储在同一区域（Region）的 3 个可用区（Zone）中。提供机房级容灾能力，能提高您的数据可用性，**同城冗余存储属性开启后，将不支持关闭**。更多详情请参见 [同](#)

版本控制 Hot



未开通

开启版本控制后，针对数据的覆盖和删除操作将会以历史版本的形式保存下来，若不开启版本控制则数据删除或被覆盖将无法找回，了解 [版本控制](#)。当前未开启版本控制功能，数据删除或被覆盖后将无法找回。

开启版本控制后，对当前版本和所有历史版本的文件都会收取存储费用，详情请查看[计费文档](#)

读写权限

私有

公共读

公共读写

公共读：对文件写操作需要进行身份验证；可以对文件进行匿名读。

服务端加密方式

☒ 无 ☐ OSS 完全托管 ☐ KMS [?](#)

实时日志查询



未开通

开通该功能后，您可对Bucket的访问记录进行实时查询分析。

OSS与日志服务深度结合，**免费提供最近7天内的OSS实时日志查询**（限额900 GB/天的日志写入额度，超出部分由日志服务单独收费，假设一条日志为1 KB，约为9亿条）。详情请查看[计费文档](#)

定时备份



未开通

混合云备份服务为OSS 提供数据处理保护，防止误修改、误删除，可低成本长期保存历史数据。服务首次开通后 30 天内免费试用，详情请查看[计费文档](#)

确定

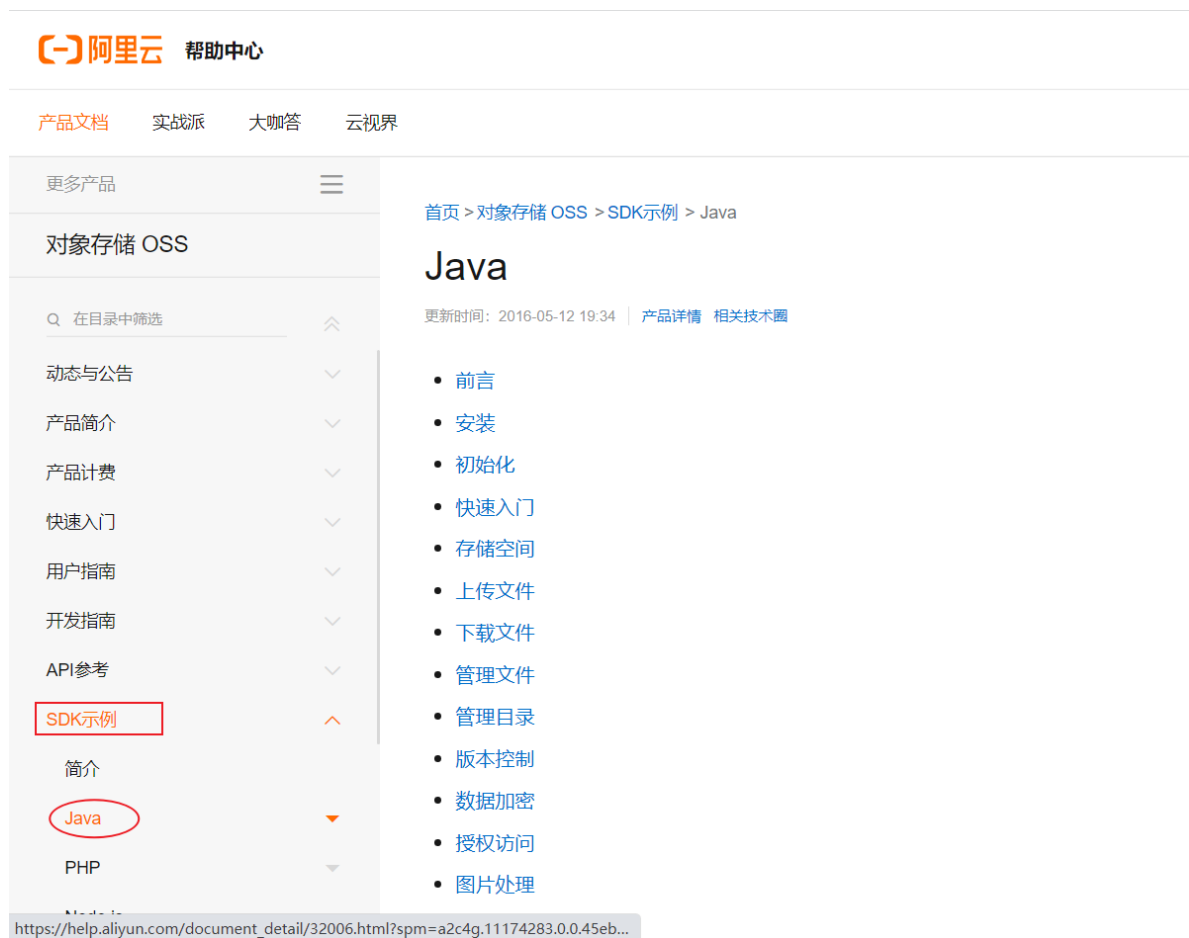
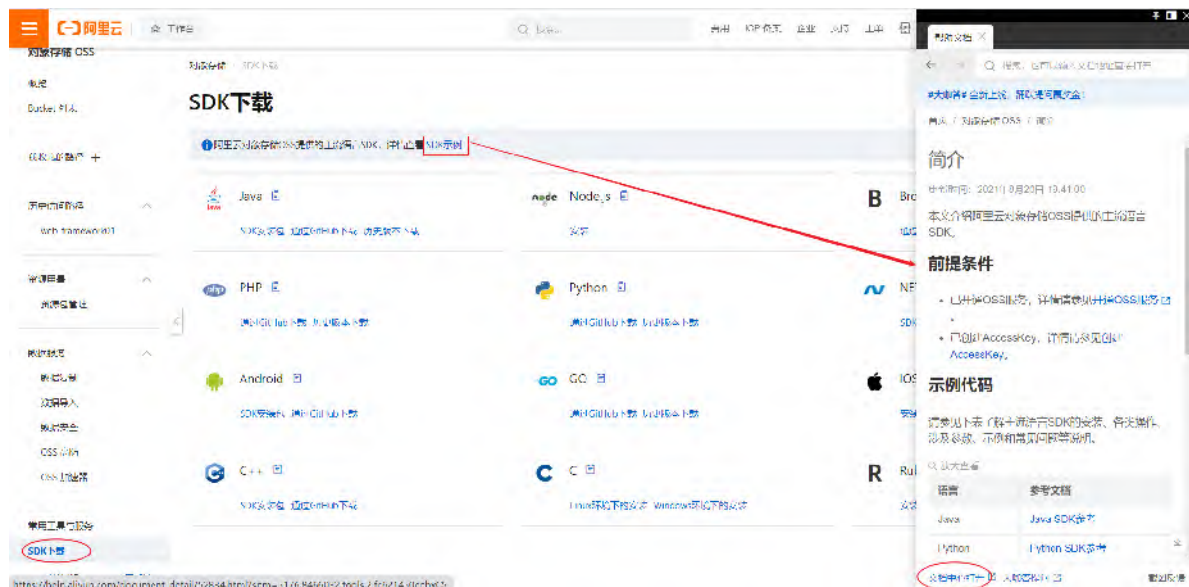
取消

大家可以参照“资料\04. 阿里云OSS”中提供的文档，开通阿里云OSS服务。

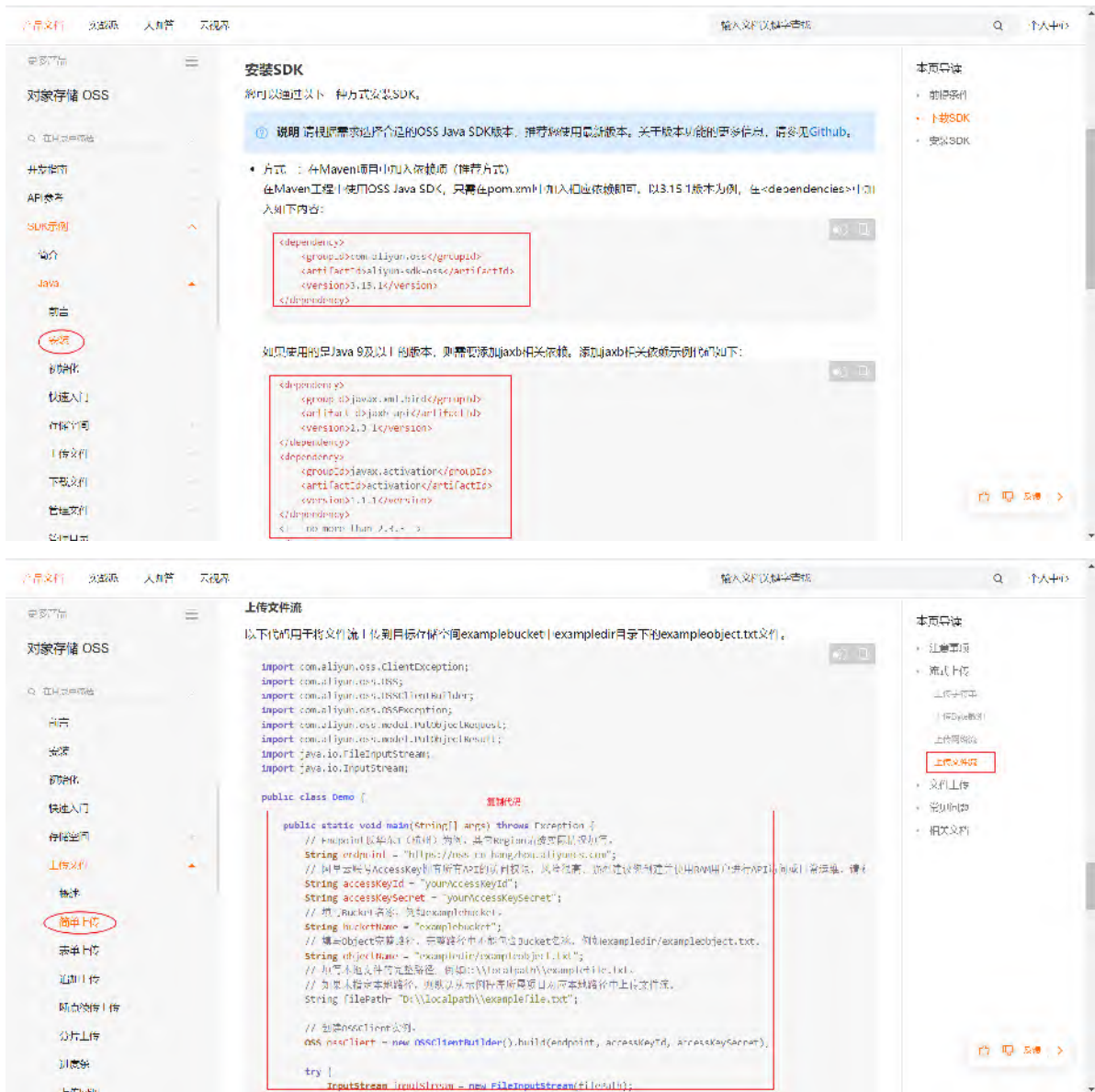
## 2.3.2 入门

阿里云OSS 对象存储服务的准备工作我们已经完成了，接下来我们就来完成第二步操作：参照官方所提供的sdk示例来编写入门程序。

首先我们需要来打开阿里云OSS的官方文档，在官方文档中找到 SDK 的示例代码：



如果是在实际开发当中，我们是需要从前往后仔细的去阅读这一份文档的，但是由于现在是教学，我们就只挑重点的去看。有兴趣的同学大家下来也可以自己去看一下这份官方文档。



参照官方提供的SDK，改造一下，即可实现文件上传功能：

```
1  import com.aliyun.oss.ClientException;
2  import com.aliyun.oss.OSS;
3  import com.aliyun.oss.OSSClientBuilder;
4  import com.aliyun.oss.OSSException;
5  import com.aliyun.oss.model.PutObjectRequest;
6  import com.aliyun.oss.model.PutObjectResult;
7
8  import java.io.FileInputStream;
9  import java.io.InputStream;
10
11 public class AliOssTest {
12     public static void main(String[] args) throws Exception {
13         // Endpoint以华东1（杭州）为例，其它Region请按实际情况填写。
14         String endpoint = "oss-cn-shanghai.aliyuncs.com";
15
16         // 阿里云账号AccessKey拥有所有API的访问权限，风险很高。强烈建议您
            创建并使用RAM用户进行API访问或日常运维，请登录RAM控制台创建RAM用户。
```



```

17         String accessKeyId = "LTAI5t9MZK8iq5T2Av5GLDxX";
18         String accessKeySecret = "C0IrHzKZGKqU8S7YQcevcotD3Zd5Tc";
19
20         // 填写Bucket名称，例如examplebucket。
21         String bucketName = "web-framework01";
22         // 填写Object完整路径，完整路径中不能包含Bucket名称，例如
        examplemdir/exampleobject.txt。
23         String objectName = "1.jpg";
24         // 填写本地文件的完整路径，例如D:\\localpath\\examplefile.txt。
25         // 如果未指定本地路径，则默认从示例程序所属项目对应本地路径中上传文
        件流。
26         String filePath=
        "C:\\Users\\Administrator\\Pictures\\1.jpg";
27
28         // 创建OSSClient实例。
29         OSS ossClient = new OSSClientBuilder().build(endpoint,
        accessKeyId, accessKeySecret);
30
31         try {
32             InputStream inputStream = new FileInputStream(filePath);
33             // 创建PutObjectRequest对象。
34             PutObjectRequest putObjectRequest = new
        PutObjectRequest(bucketName, objectName, inputStream);
35             // 设置该属性可以返回response。如果不设置，则返回的response为
        空。
36             putObjectRequest.setProcess("true");
37             // 创建PutObject请求。
38             PutObjectResult result =
        ossClient.putObject(putObjectRequest);
39             // 如果上传成功，则返回200。
40
41             System.out.println(result.getResponse().getStatusCode());
42             } catch (OSSException oe) {
43                 System.out.println("Caught an OSSException, which means
        your request made it to OSS, "
44                 + "but was rejected with an error response for
        some reason.");
45                 System.out.println("Error Message:" +
        oe.getErrorMessage());
46                 System.out.println("Error Code:" + oe.getErrorCode());
47                 System.out.println("Request ID:" + oe.getRequestId());
48                 System.out.println("Host ID:" + oe.getHostId());
49             } catch (ClientException ce) {

```

```

49         System.out.println("Caught an ClientException, which
means the client encountered "
50             + "a serious internal problem while trying to
communicate with OSS, "
51             + "such as not being able to access the
network.");
52         System.out.println("Error Message:" + ce.getMessage());
53     } finally {
54         if (ossClient != null) {
55             ossClient.shutdown();
56         }
57     }
58 }
59 }
60

```

在以上代码中，需要替换的内容为：

- accessKeyId：阿里云账号AccessKey
- accessKeySecret：阿里云账号AccessKey对应的密钥
- bucketName：Bucket名称
- objectName：对象名称，在Bucket中存储的对象的名称
- filePath：文件路径

AccessKey：



运行以上程序后，会把本地的文件上传到阿里云OSS服务器上：



### 2.3.3 集成

阿里云oss对象存储服务的准备工作以及入门程序我们都已经完成了，接下来我们就需要在案例当中集成oss对象存储服务，来存储和管理案例中上传的图片。



在新增员工的时候，上传员工的图像，而之所以需要上传员工的图像，是因为将来我们需要在系统页面当中访问并展示员工的图像。而要想完成这个操作，需要做两件事：

1. 需要上传员工的图像，并把图像保存起来（存储到阿里云OSS）
2. 访问员工图像（通过图像在阿里云OSS的存储地址访问图像）
  - OSS中的每一个文件都会分配一个访问的url，通过这个url就可以访问到存储在阿里云上的图片。所以需要把url返回给前端，这样前端就可以通过url获取到图像。

我们参照接口文档来开发文件上传功能：

#### • 基本信息

- 1 请求路径：/upload
- 2
- 3 请求方式：POST
- 4
- 5 接口描述：上传图片接口

#### • 请求参数

参数格式：multipart/form-data

参数说明：

参数名称	参数类型	是否必须	示例	备注
image	file	是		

#### • 响应数据

参数格式: application/json

参数说明:

参数名	类型	是否必须	备注
code	number	必须	响应码, 1 代表成功, 0 代表失败
msg	string	非必须	提示信息
data	object	非必须	返回的数据, 上传图片的访问路径

响应数据样例:

```
1  {
2      "code": 1,
3      "msg": "success",
4      "data": "https://web-framework.oss-cn-
           hangzhou.aliyuncs.com/2022-09-02-00-27-0400.jpg"
5  }
```

引入阿里云oss上传文件工具类（由官方的示例代码改造而来）

```
1  import com.aliyun.oss.OSS;
2  import com.aliyun.oss.OSSClientBuilder;
3  import org.springframework.stereotype.Component;
4  import org.springframework.web.multipart.MultipartFile;
5
6  import java.io.IOException;
7  import java.io.InputStream;
8  import java.util.UUID;
9
10 @Component
11 public class AliOSSUtils {
12     private String endpoint = "https://oss-cn-
           shanghai.aliyuncs.com";
13     private String accessKeyId = "LTAI5t9MZK8iq5T2Av5GLDxX";
14     private String accessKeySecret =
           "C0IrHzKZGKqU8S7YQcevcotD3Zd5Tc";
15     private String bucketName = "web-framework01";
16
17     /**
18      * 实现上传图片到oss
19      */
20 }
```

```

20     public String upload(MultipartFile multipartFile) throws
IOException {
21         // 获取上传的文件的输入流
22         InputStream inputStream = multipartFile.getInputStream();
23
24         // 避免文件覆盖
25         String originalFilename =
multipartFile.getOriginalFilename();
26         String fileName = UUID.randomUUID().toString() +
originalFilename.substring(originalFilename.lastIndexOf("."));
27
28         //上传文件到 oss
29         OSS ossClient = new OSSClientBuilder().build(endpoint,
accessKeyId, accessKeySecret);
30         ossClient.putObject(bucketName, fileName, inputStream);
31
32         //文件访问路径
33         String url = endpoint.split("//")[0] + "://" + bucketName +
"." + endpoint.split("//")[1] + "/" + fileName;
34
35         // 关闭ossClient
36         ossClient.shutdown();
37         return url; // 把上传到oss的路径返回
38     }
39 }

```

修改UploadController代码:

```

1  import com.itheima.pojo.Result;
2  import com.itheima.utils.AliOSSUtils;
3  import lombok.extern.slf4j.Slf4j;
4  import org.springframework.beans.factory.annotation.Autowired;
5  import org.springframework.web.bind.annotation.PostMapping;
6  import org.springframework.web.bind.annotation.RestController;
7  import org.springframework.web.multipart.MultipartFile;
8  import java.io.IOException;
9
10  @Slf4j
11  @RestController
12  public class UploadController {
13
14      @Autowired
15      private AliOSSUtils aliOSSUtils;
16
17      @PostMapping("/upload")

```

```

18     public Result upload(MultipartFile image) throws IOException {
19         //调用阿里云oss工具类，将上传上来的文件存入阿里云
20         String url = aliOSSUtils.upload(image);
21         //将图片上传完成后的url返回，用于浏览器回显展示
22         return Result.success(url);
23     }
24
25 }

```

使用postman测试：



### 3. 修改员工

需求：修改员工信息

<input type="checkbox"/>	姓名	图 像	性别	职位	入职日期	最后操作时间	操作
<input type="checkbox"/>	赵敏		女	班主任	2008-12-18	2022-07-22 12:05:20	<a href="#">编辑</a> <a href="#">删除</a>
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	<a href="#">编辑</a> <a href="#">删除</a>
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	<a href="#">编辑</a> <a href="#">删除</a>
<input type="checkbox"/>	风清扬		男	讲师	2015-07-22	2022-07-21 15:00:00	<a href="#">编辑</a> <a href="#">删除</a>

修改员工

\* 用户名

zhaomin

\* 员工姓名

赵敏

\* 性 别

女

图 像



职 位

班主任

入职日期

2008-12-18

归属部门

学工部

保存

取消

在进行修改员工信息的时候，我们首先先要根据员工的ID查询员工的信息用于页面回显展示，然后用户修改员工数据之后，点击保存按钮，就可以将修改的数据提交到服务端，保存到数据库。 具体操作为：

1. 根据ID查询员工信息
2. 保存修改的员工信息

### 3.1 查询回显

#### 3.1.1 接口文档

根据ID查询员工数据

- 基本信息

- 1 请求路径：/emps/{id}
- 2
- 3 请求方式：GET
- 4
- 5 接口描述：该接口用于根据主键ID查询员工的信息

- 请求参数

参数格式：路径参数

参数说明：

参数名	类型	是否必须	备注
id	number	必须	员工ID

请求参数样例:

```
1 /emps/1
```

• 响应数据

参数格式: application/json

参数说明:



名称	类型	是否必须	默认值	备注
code	number	必须		响应码, 1 成功 , 0 失败
msg	string	非必须		提示信息
data	object	必须		返回的数据
- id	number	非必须		id
- username	string	非必须		用户名
- name	string	非必须		姓名
- password	string	非必须		密码
- entrydate	string	非必须		入职日期
- gender	number	非必须		性别 , 1 男 ; 2 女

名称	类型	是否必须	默认值	备注
- image	string	非必须		图像
- job	number	非必须		职位, 说明: 1 班主任, 2 讲师, 3 学工主管, 4 教研主管, 5 咨询师
- deptId	number	非必须		部门id
- createTime	string	非必须		创建时间
- updateTime	string	非必须		更新时间

响应数据样例:

```
1  {
2    "code": 1,
3    "msg": "success",
4    "data": {
5      "id": 2,
6      "username": "zhangwuji",
7      "password": "123456",
8      "name": "张无忌",
9      "gender": 1,
10     "image": "https://web-framework.oss-cn-hangzhou.aliyuncs.com/2022-09-02-00-27-53B.jpg",
11     "job": 2,
12     "entrydate": "2015-01-01",
13     "deptId": 2,
14     "createTime": "2022-09-01T23:06:30",
```

```

15         "updateTime": "2022-09-02T00:29:04"
16     }
17 }

```

### 3.1.2 实现思路



### 3.1.3 代码实现

- EmpMapper

```

1  @Mapper
2  public interface EmpMapper {
3
4      //根据ID查询员工信息
5      @Select("select id, username, password, name, gender, image,
6              job, entrydate, dept_id, create_time, update_time " +
7              "from emp " +
8              "where id = #{id}")
9      public Emp findById(Integer id);
10
11     //省略...
12 }

```

- EmpService

```

1  public interface EmpService {
2
3      /**
4       * 根据ID查询员工
5       * @param id
6       * @return
7       */
8      public Emp getById(Integer id);
9
10     //省略...
11 }

```

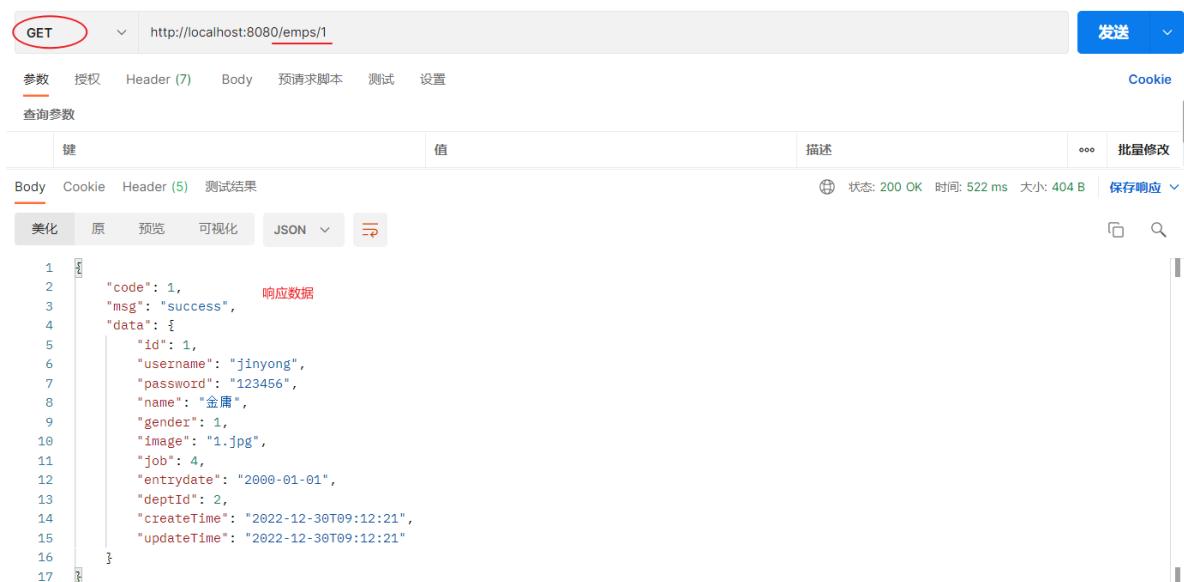
- EmpServiceImpl

```
1  @Slf4j
2  @Service
3  public class EmpServiceImpl implements EmpService {
4      @Autowired
5      private EmpMapper empMapper;
6
7      @Override
8      public Emp getById(Integer id) {
9          return empMapper.findById(id);
10     }
11
12     //省略...
13 }
```

- EmpController

```
1  @Slf4j
2  @RestController
3  @RequestMapping("/emps")
4  public class EmpController {
5
6      @Autowired
7      private EmpService empService;
8
9      //根据id查询
10     @GetMapping("/{id}")
11     public Result getById(@PathVariable Integer id){
12         Emp emp = empService.getById(id);
13         return Result.success(emp);
14     }
15
16     //省略...
17 }
```

### 3.1.4 postman测试



### 3.2 修改员工

修改员工

\* 用户名

zhaomin

\* 员工姓名

赵敏

\* 性 别

女

图 像

职 位

班主任

入职日期

2008-12-18

归属部门

学工部

保存

取消

当用户修改完数据之后，点击保存按钮，就需要将数据提交到服务端，然后服务端需要将修改后的数据更新到数据库中。

### 3.2.1 接口文档

- 基本信息

- 1

请求路径：/emps
- 2
- 3

请求方式：PUT
- 4
- 5

接口描述：该接口用于修改员工的数据信息

- 请求参数

参数格式：application/json

参数说明：

名称	类型	是否必须	备注
id	number	必须	id
username	string	必须	用户名
name	string	必须	姓名
gender	number	必须	性别，说明：1 男，2 女
image	string	非必须	图像
deptId	number	非必须	部门id
entrydate	string	非必须	入职日期
job	number	非必须	职位，说明：1 班主任, 2 讲师, 3 学工主管, 4 教研主管, 5 咨询师

请求数据样例：

```

1  {
2      "id": 1,
3      "image": "https://web-framework.oss-cn-
      hangzhou.aliyuncs.com/2022-09-03-07-37-38222.jpg",
4      "username": "linpingzhi",
5      "name": "林平之",
6      "gender": 1,
7      "job": 1,
8      "entrydate": "2022-09-18",
9      "deptId": 1
10 }

```

- 响应数据

参数格式: application/json

参数说明:

参数名	类型	是否必须	备注
code	number	必须	响应码, 1 代表成功, 0 代表失败
msg	string	非必须	提示信息
data	object	非必须	返回的数据

响应数据样例:

```

1  {
2      "code":1,
3      "msg":"success",
4      "data":null
5  }

```

### 3.2.2 实现思路



### 3.2.3 代码实现

- EmpMapper

```
1  @Mapper
2  public interface EmpMapper {
3      //修改员工信息
4      public void update(Emp emp);
5
6      //省略...
7  }
```

- EmpMapper.xml

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE mapper
3      PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4      "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5  <mapper namespace="com.itheima.mapper.EmpMapper">
6
7      <!--更新员工信息-->
8      <update id="update">
9          update emp
10         <set>
11             <if test="username != null and username != ''">
12                 username = #{username},
13             </if>
14             <if test="password != null and password != ''">
15                 password = #{password},
16             </if>
17             <if test="name != null and name != ''">
18                 name = #{name},
19             </if>
20             <if test="gender != null">
21                 gender = #{gender},
22             </if>
23             <if test="image != null and image != ''">
24                 image = #{image},
25             </if>
26             <if test="job != null">
27                 job = #{job},
28             </if>
29             <if test="entrydate != null">
30                 entrydate = #{entrydate},
31             </if>
```



```

32         <if test="deptId != null">
33             dept_id = #{deptId},
34         </if>
35         <if test="updateTime != null">
36             update_time = #{updateTime}
37         </if>
38     </set>
39     where id = #{id}
40 </update>
41
42     <!-- 省略... -->
43
44 </mapper>

```

- EmpService

```

1  public interface EmpService {
2      /**
3       * 更新员工
4       * @param emp
5       */
6      public void update(Emp emp);
7
8      //省略...
9  }

```

- EmpServiceImpl

```

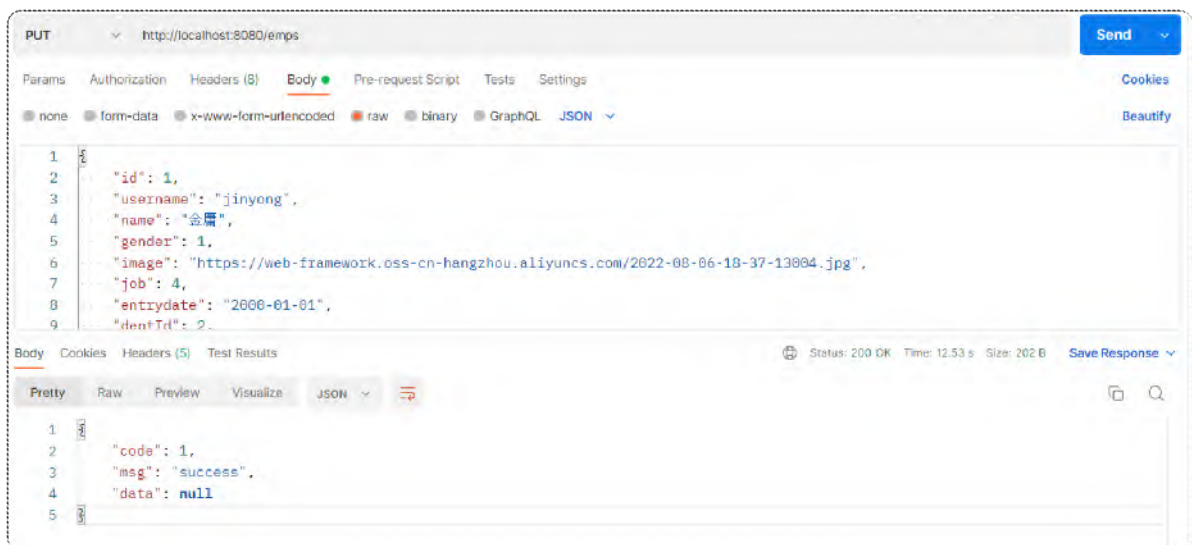
1  @Slf4j
2  @Service
3  public class EmpServiceImpl implements EmpService {
4      @Autowired
5      private EmpMapper empMapper;
6
7      @Override
8      public void update(Emp emp) {
9          emp.setUpdateTime(LocalDate.now()); //更新修改时间为当前时
          间
10
11          empMapper.update(emp);
12      }
13
14      //省略...
15  }

```

- EmpController

```
1  @Slf4j
2  @RestController
3  @RequestMapping("/emps")
4  public class EmpController {
5
6      @Autowired
7      private EmpService empService;
8
9      //修改员工
10     @PutMapping
11     public Result update(@RequestBody Emp emp) {
12         empService.update(emp);
13         return Result.success();
14     }
15
16     //省略...
17 }
```

### 3.2.4 postman测试



### 3.2.5 前后端联调测试



## 4. 配置文件

员工管理的增删改查功能我们已开发完成，但在我们所开发的程序中还有一些小问题，下面我们就来分析一下当前案例中存在的问题以及如何优化解决。

### 4.1 参数配置化

```
@Component
public class AliOSSUtils {
    private String endpoint = "https://oss-cn-hangzhou.aliyuncs.com";
    private String accessKeyId = "LTAI4GCH1vX6DKqJWxd6nEuW";
    private String accessKeySecret = "yBshYweHOpqDuhCArrVHwIiBKpyqSL";
    private String bucketName = "web-tljias";

    /**
     * 实现上传图片到OSS
     */
    public String upload(MultipartFile file) throws IOException {
```

在我们之前编写的程序中进行文件上传时，需要调用AliOSSUtils工具类，将文件上传到阿里云OSS对象存储服务当中。而在调用工具类进行文件上传时，需要一些参数：

- endpoint //阿里云OSS域名
- accessKeyId //用户身份ID
- accessKeySecret //用户密钥
- bucketName //存储空间的名字

关于以上的这些阿里云相关配置信息，我们是直接写死在java代码中了（硬编码），如果我们在做项目时每涉及到一个第三方技术服务，就将其参数硬编码，那么在Java程序中会存在两个问题：

1. 如果这些参数发生了变化了，就必须在源程序代码中改动这些参数，然后需要重新进行代码的编译，将Java代码编译成class字节码文件再重新运行程序。（比较繁琐）
2. 如果我们开发的是一个真实的企业级项目，Java类可能会有很多，如果将这些参数分散的定义在各个Java类当中，我们要修改一个参数值，我们就需要在众多的Java代码当中来定位到对应的位置，再来修改参数，修改完毕之后再重新编译再运行。（参数配置过于分散，是不方便集中的管理和维护）

为了解决以上分析的问题，我们可以将参数配置在配置文件中。如下：

```
1  #自定义的阿里云oss配置信息
2  aliyun.oss.endpoint=https://oss-cn-hangzhou.aliyuncs.com
3  aliyun.oss.accessKeyId=LTAI4GCH1vX6DKqJWxd6nEuW
4  aliyun.oss.accessKeySecret=yBshYweHOpqDuhCArrVHwIiBKpyqSL
5  aliyun.oss.bucketName=web-tlias
```

在将阿里云OSS配置参数交给properties配置文件来管理之后，我们的AliOSSUtils工具类就变为以下形式：

```
1  @Component
2  public class AliOSSUtils {
3      /*以下4个参数没有指定值（默认值：null）*/
4      private String endpoint;
5      private String accessKeyId;
6      private String accessKeySecret;
7      private String bucketName;
8
9      //省略其他代码...
10 }
```

而此时如果直接调用AliOSSUtils类当中的upload方法进行文件上传时，这4项参数全部为null，原因是因为并没有给它赋值。

此时我们是不是需要将配置文件当中所配置的属性值读取出来，并分别赋值给AliOSSUtils工具类当中的各个属性呢？那应该怎么做呢？

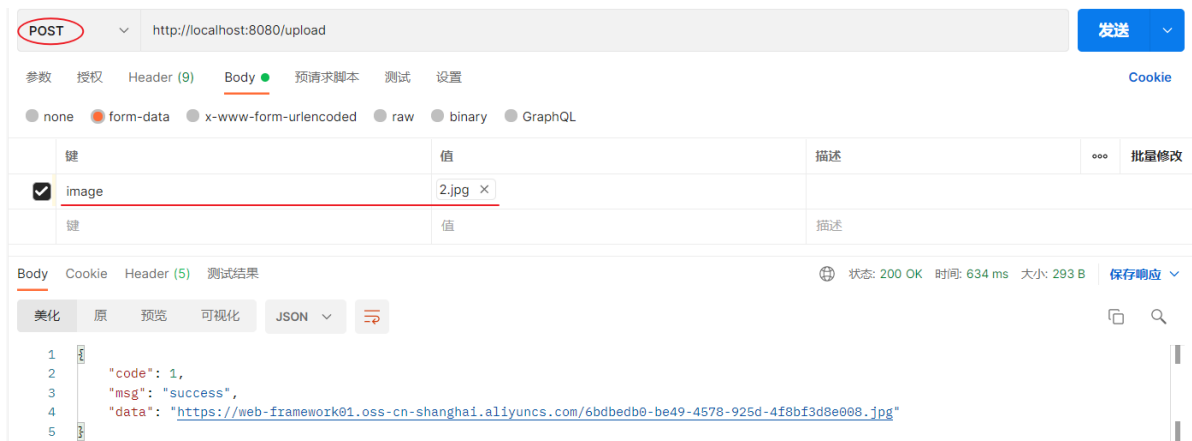
因为application.properties是springboot项目默认的配置文件，所以springboot程序在启动时会默认读取application.properties配置文件，而我们可以使用一个现成的注解：@Value，获取配置文件中的数据。

@Value 注解通常用于外部配置的属性注入，具体用法为：@Value("\${配置文件中的key}")

```
1  @Component
2  public class AliOSSUtils {
3
4      @Value("${aliyun.oss.endpoint}")
5      private String endpoint;
6
7      @Value("${aliyun.oss.accessKeyId}")
8      private String accessKeyId;
9
10     @Value("${aliyun.oss.accessKeySecret}")
11     private String accessKeySecret;
12
13     @Value("${aliyun.oss.bucketName}")
14     private String bucketName;
15
16     //省略其他代码...
17 }
```



使用postman测试：



## 4.2 yaml配置文件

前面我们一直使用springboot项目创建完毕后自带的application.properties进行属性的配置，那其实呢，在springboot项目当中是支持多种配置方式的，除了支持properties配置文件以外，还支持另外一种类型的配置文件，就是我们接下来要讲解的yaml格式的配置文件。

- application.properties

```
1  server.port=8080
2  server.address=127.0.0.1
```

- application.yml

```
1  server:
2    port: 8080
3    address: 127.0.0.1
```

- application.yaml

```
1  server:
2    port: 8080
3    address: 127.0.0.1
```

yaml 格式的配置文件，后缀名有两种：

- yaml （推荐）
- yml

常见配置文件格式对比：

XML配置文件:

```
<server>
  <port>8080</port>
  <address>127.0.0.1</address>
</server>
```

臃肿

properties配置文件:

```
server.port=8080
server.address=127.0.0.1
```

层级结构不清晰

yaml / yml配置文件:

```
server:
  port: 8080
  address: 127.0.0.1
```

简洁、数据为中心

properties配置文件格式:

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/tlias
spring.datasource.username=root
spring.datasource.password=1234
```

yaml / yml配置文件格式:

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/tlias
    username: root
    password: 1234
```

我们可以看到配置同样的数据信息，yaml格式的数据有以下特点:

- 容易阅读
- 容易与脚本语言交互
- 以数据为核心，重数据轻格式

简单的了解过springboot所支持的配置文件，以及不同类型配置文件之间的优缺点之后，接下来我们就了解下yaml配置文件的基本语法:

- 大小写敏感
- 数值前边必须有空格，作为分隔符
- 使用缩进表示层级关系，缩进时，不允许使用Tab键，只能用空格（idea中会自动将Tab转换为空格）
- 缩进的空格数目不重要，只要相同层级的元素左侧对齐即可
- # 表示注释，从这个字符一直到行尾，都会被解析器忽略

#配置服务器相关信息

```
server:
  port: 8080
  address: 127.0.0.1
```

了解完yaml格式配置文件的基本语法之后，接下来我们再来看下yaml文件中常见的数据格式。在这里我们主要介绍最为常见的两类:

1. 定义对象或Map集合
2. 定义数组、list或set集合

对象/Map集合



```
1  user:
2    name: zhangsan
3    age: 18
4    password: 123456
```

## 数组/List/Set集合

```
1  hobby:
2    - java
3    - game
4    - sport
```

熟悉完了yml文件的基本语法后，我们修改下之前案例中使用的配置文件，变更为application.yml  
配置方式：

1. 修改application.properties名字为： `_application.properties`（名字随便更换，只要加载不到即可）
2. 创建新的配置文件： `application.yml`

原有application.properties文件：

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/tlias
spring.datasource.username=root
spring.datasource.password=1234
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=100MB

mybatis.configuration.log-impl=org.apache.ibatis.logging.stdout.StdOutImpl
mybatis.configuration.map-underscore-to-camel-case=true

#自定义的阿里云OSS配置信息
aliyun.oss.endpoint=https://oss-cn-hangzhou.aliyuncs.com
aliyun.oss.accessKeyId=LTAI4GCH1vX6DKqJWxd6nEuW
aliyun.oss.accessKeySecret=yBshYweHOpqDuhCArrVHwIiBKpyqSL
aliyun.oss.bucketName=web-framework
```

新建的application.yml文件：

```
1  spring:
2    datasource:
3      driver-class-name: com.mysql.cj.jdbc.Driver
4      url: jdbc:mysql://localhost:3306/tlias
5      username: root
6      password: 1234
7    servlet:
8      multipart:
9        max-file-size: 10MB
```

```

10     max-request-size: 100MB
11
12     mybatis:
13         configuration:
14             log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
15             map-underscore-to-camel-case: true
16
17     aliyun:
18         oss:
19             endpoint: https://oss-cn-hangzhou.aliyuncs.com
20             accessKeyId: LTAI4GCH1vX6DKqJWxd6nEuW
21             accessKeySecret: yBshYweHOpqDuhCArrVHwIiBKpyqSL
22             bucketName: web-397

```

### 4.3 @ConfigurationProperties

讲解完了yml配置文件之后，最后再来介绍一个注解 `@ConfigurationProperties`。在介绍注解之前，我们先来看一个场景，分析下代码当中可能存在的问题：

```

aliyun:
  oss:
    endpoint: https://oss-cn-hangzhou.aliyuncs.com
    accessKeyId: LTAI4GCH1vX6DKqJWxd6nEuW
    accessKeySecret: yBshYweHOpqDuhCArrVHwIiBKpyqSL
    bucketName: web-framework

```

```

@Component
public class AliOSSUtils {

    @Value("${aliyun.oss.endpoint}")
    private String endpoint;

    @Value("${aliyun.oss.accessKeyId}")
    private String accessKeyId;

    @Value("${aliyun.oss.accessKeySecret}")
    private String accessKeySecret;

    @Value("${aliyun.oss.bucketName}")
    private String bucketName;
}

```

我们在application.properties或者application.yml中配置了阿里云OSS的四项参数之后，如果java程序中需要这四项参数数据，我们直接通过@Value注解来进行注入。这种方式本身没有什么问题，但是如果说需要注入的属性较多（例：需要20多个参数数据），我们写起来就会比较繁琐。

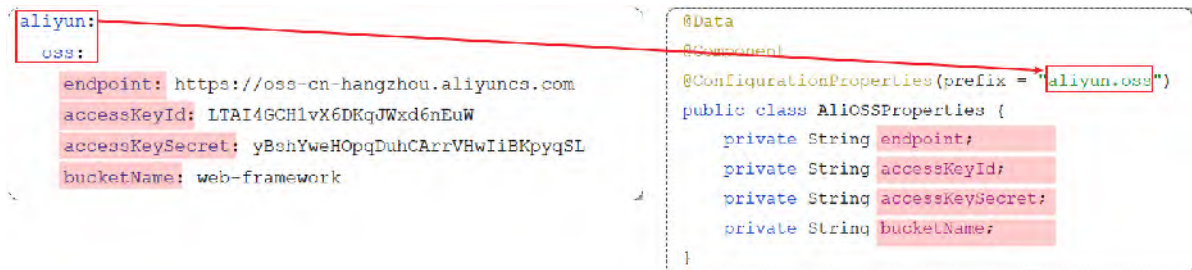
那么有没有一种方式可以简化这些配置参数的注入呢？答案是肯定有，在Spring中给我们提供了一种简化方式，可以直接将配置文件中配置项的值自动的注入到对象的属性中。

Spring提供的简化方式套路：

1. 需要创建一个实现类，且实体类中的属性名和配置文件当中key的名字必须要一致

比如：配置文件当中叫endpoints，实体类当中的属性也得叫endpoints，另外实体类当中的属性还需要提供 getter / setter方法

2. 需要将实体类交给Spring的IOC容器管理，成为IOC容器当中的bean对象
3. 在实体类上添加 `@ConfigurationProperties` 注解，并通过prefix属性来指定配置参数项的前缀



实体类: AliOSSProperties

```
1  import lombok.Data;
2  import
    org.springframework.boot.context.properties.ConfigurationProperties;
3  import org.springframework.stereotype.Component;
4
5  /*阿里云OSS相关配置*/
6  @Data
7  @Component
8  @ConfigurationProperties(prefix = "aliyun.oss")
9  public class AliOSSProperties {
10     //区域
11     private String endpoint;
12     //身份ID
13     private String accessKeyId ;
14     //身份密钥
15     private String accessKeySecret ;
16     //存储空间
17     private String bucketName;
18 }
```

AliOSSUtils工具类:

```
1  import com.aliyun.oss.OSS;
2  import com.aliyun.oss.OSSClientBuilder;
3  import org.springframework.beans.factory.annotation.Autowired;
4  import org.springframework.stereotype.Component;
```

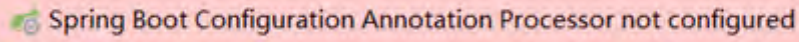
```

5  import org.springframework.web.multipart.MultipartFile;
6  import java.io.IOException;
7  import java.io.InputStream;
8  import java.util.UUID;
9
10 @Component //当前类对象由Spring创建和管理
11 public class AliOSSUtils {
12
13     //注入配置参数实体类对象
14     @Autowired
15     private AliOSSProperties aliOSSProperties;
16
17
18     /**
19      * 实现上传图片到oss
20      */
21     public String upload(MultipartFile multipartFile) throws
IOException {
22         // 获取上传的文件的输入流
23         InputStream inputStream = multipartFile.getInputStream();
24
25         // 避免文件覆盖
26         String originalFilename =
multipartFile.getOriginalFilename();
27         String fileName = UUID.randomUUID().toString() +
originalFilename.substring(originalFilename.lastIndexOf("."));
28
29         //上传文件到 oss
30         OSS ossClient = new
OSSClientBuilder().build(aliOSSProperties.getEndpoint(),
31             aliOSSProperties.getAccessKeyId(),
aliOSSProperties.getAccessKeySecret());
32         ossClient.putObject(aliOSSProperties.getBucketName(),
fileName, inputStream);
33
34         //文件访问路径
35         String url =aliOSSProperties.getEndpoint().split("//")[0] +
"//" + aliOSSProperties.getBucketName() + "." +
aliOSSProperties.getEndpoint().split("//")[1] + "/" + fileName;
36
37         // 关闭ossClient
38         ossClient.shutdown();
39         return url;// 把上传到oss的路径返回
40     }

```

```
41    }  
42
```

在我们添加上注解后，会发现idea窗口上面出现一个红色警告：

A red error banner from IntelliJ IDEA stating "Spring Boot Configuration Annotation Processor not configured".

这个警告提示是告知我们还需要引入一个依赖：

```
1  <dependency>  
2      <groupId>org.springframework.boot</groupId>  
3      <artifactId>spring-boot-configuration-processor</artifactId>  
4  </dependency>
```

当我们在pom.xml文件当中配置了这项依赖之后，我们重新启动服务，大家就会看到在properties或者是yml配置文件当中，就会提示阿里云 OSS 相关的配置项。所以这项依赖它的作用就是会自动的识别被 `@Configuration Properties` 注解标识的bean对象。

刚才的红色警告，已经变成了一个灰色的提示，提示我们需要重新运行springboot服务

@ConfigurationProperties注解我们已经介绍完了，接下来我们就来区分一下

@ConfigurationProperties注解以及我们前面所介绍的另外一个@Value注解：

相同点：都是用来注入外部配置的属性的。

不同点：

- @Value注解只能一个一个的进行外部属性的注入。
- @ConfigurationProperties可以批量的将外部的属性配置注入到bean对象的属性中。

如果要注入的属性非常的多，并且还想做到复用，就可以定义这么一个bean对象。通过

configuration properties 批量的将外部的属性配置直接注入到 bin 对象的属性当中。在其他的类当中，我要想获取到注入进来的属性，我直接注入 bin 对象，然后调用 get 方法，就可以获取到对应的属性值了