# 今日任务

1. Fastdfs的java客户端（重点）
2. 商品图片上传流程（难点）
3. 完成添加商品之上传图片功能
4. 完成添加商品之添加商品详情功能

# Java客户端

存储层

支撑层

fastdfs

图片服务器（对象层）

文件微服务层

客户端

http协议

Tcp协议

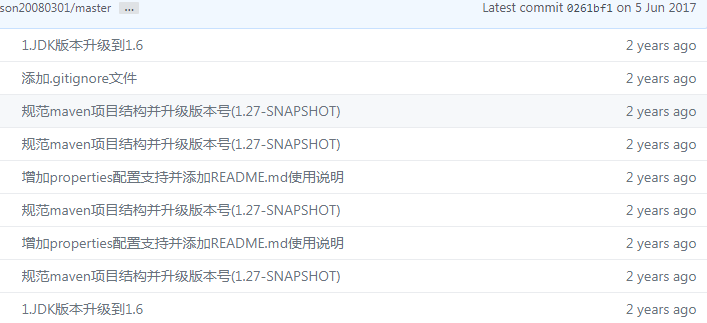
## 一次抉择

1. Fastdfs-java-client

官网版本：<https://github.com/happyfish100/fastdfs-client-java>

优点：余庆本人所写，权威性够强，文档还算全，无第三方包依赖

缺点：没有使用连接池，不利于高并发场景使用



已经2两年未更新了！！

1. FastDFS-Client

网址：<https://github.com/tobato/FastDFS_Client>

特点：

1，对关键部分代码加入了单元测试，便于理解与服务端的接口交易，提高接口质量

2，将以前对byte硬解析风格重构为使用 对象+注解 的形式，尽量增强了代码的可读性

3，支持对服务端的连接池管理(commons-pool2)

4，支持上传图片时候检查图片格式，并且自动生成缩略图

5，**在SpringBoot当中自动导入依赖**

**提示：上面都不是关键，关键是代码活跃度和*issues都*很高，这是选择开源软件的首要原则，再牛逼的项目，如果很长时间没更新，作为一个老鸟，劝你三思而行，除非你真的非常牛逼，出了问题，你有能力去修改源代码，自己修改bug，同学们去企业后一定要注意，很多时候我们的悲剧就是从一个不起眼的选择开始的。**

1. fastdfs-java-client

网址：<https://gitee.com/LiZhiW/fastdfs-java-client>

这个其实也不错，不过也2年没更新了，以前的同学一直用的这个。

## 总结

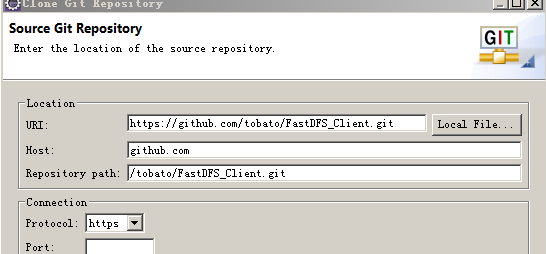
这三个是目前使用最多的fastdfs的java客户端了，经过综合的权衡，相信大家的选择应该会和我的一样。

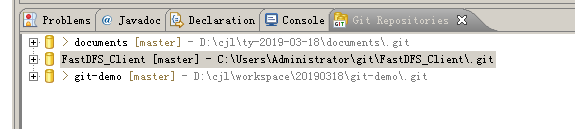
Java的开源软件实在是太多了，这是我认为java之所以几十年经久不衰的主要原因。想象一下，如果当年微软在功臣名就后主动开源，那今天的软件格局又会是怎样的呢？

开源软件多是好事，但如何选择也是烦恼，所以有必要通过一次开源软件的选择经历，给大家在使用开源软件时一些经验指导，让大家在以后的工作中少走弯路，也在面试时能侃侃而谈，让自己不只是简历上写的那样，而是只真正的久经沙场的老鸟。

# 自己动手

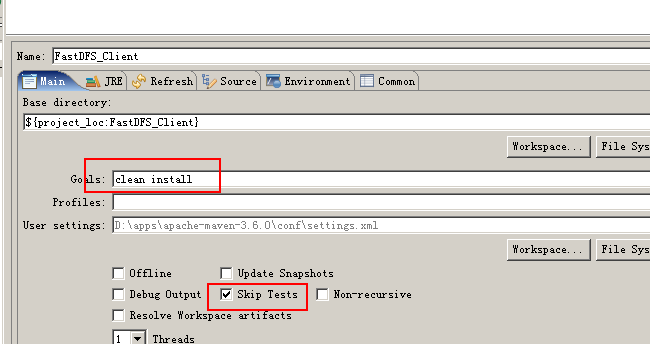
选择官网发布版：<https://github.com/tobato/FastDFS_Client/releases>

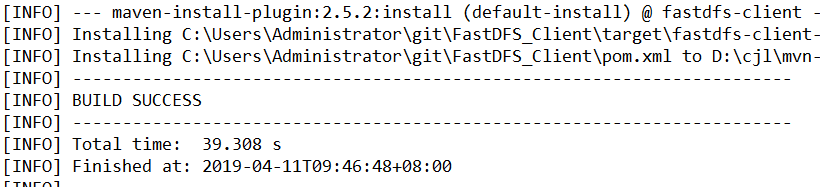




安装jar包到本地仓库，以便我们项目使用！！！

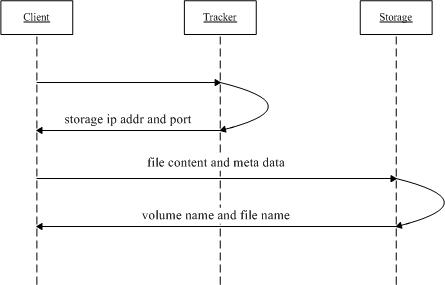
先切换到最新的发布版：1.26.5





# 上传流程

Uml：有八种图，下面是时序图。



1. client询问tracker上传到的storage，不需要附加参数；

2. tracker返回一台可用的storage；

3. client直接和storage通讯完成文件上传。

# 上传代码

## 测试代码

使用spring-boot。导入依赖。

|  |
| --- |
| <!-- fastdfs client -->  <dependency>  <groupId>com.github.tobato</groupId>  <artifactId>fastdfs-client</artifactId>  <version>1.26.5</version>  </dependency> |

配置文件

|  |
| --- |
| # ===================================================================  # 分布式文件系统FDFS配置  # ===================================================================  fdfs:  so-timeout: 1501  connect-timeout: 601  thumb-image: #缩略图生成参数  width: 150  height: 150  tracker-list: #TrackerList参数,支持多个  - 192.168.19.128:22122  #自定义配置  storage: 192.168.19.130 |

|  |
| --- |
| **package** com.go9.goods.file;  **import** java.io.File;  **import** java.io.FileInputStream;  **import** org.junit.Test;  **import** org.junit.runner.RunWith;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.boot.test.context.SpringBootTest;  **import** org.springframework.test.context.junit4.SpringJUnit4ClassRunner;  **import** com.github.tobato.fastdfs.domain.fdfs.StorePath;  **import** com.github.tobato.fastdfs.service.FastFileStorageClient;  @RunWith(SpringJUnit4ClassRunner.**class**)  @SpringBootTest  **public** **class** FileSerivceTest {  @Autowired  **private** FastFileStorageClient client;    @Test  **public** **void** testUpload() **throws** Exception {  File file = **new** File("D:\\Downloads\\go8\\go8-mall\\src\\main\\resources\\static\\images\\detail.jpg");  FileInputStream is = **new** FileInputStream(file);  StorePath path = client.uploadFile(is, file.length(), "jpg", **null**);  System.***out***.println(path);  }    @Test  **public** **void** testDelete() **throws** Exception {  client.deleteFile("group1/M00/00/00/wKg5HlyuqpuAdJoHAADWoNY6DV0660.jpg");  }  } |

## 服务层

|  |
| --- |
| package com.go9.goods.file;  import java.io.InputStream;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.beans.factory.annotation.Value;  import org.springframework.stereotype.Service;  import com.github.tobato.fastdfs.domain.fdfs.StorePath;  import com.github.tobato.fastdfs.service.FastFileStorageClient;  @Service  public class FileServiceImpl implements FileService {  @Autowired  private FastFileStorageClient client;  @Value("${file-server-ip}")  private String fileServerIp;    @Override  public String upload(InputStream inputStream, long fileSize, String fileExtName) {  StorePath storePath = client.uploadFile(inputStream, fileSize, fileExtName, null);  //group + path  String path = storePath.getFullPath();    return fileServerIp+path;  }  @Override  public void delete(String fullpath) {  client.deleteFile(fullpath);  }  } |

## Web层

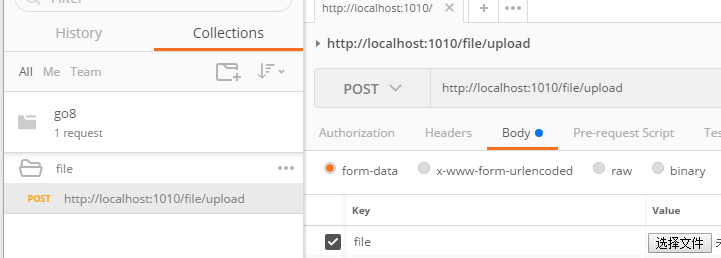
写代码：

如果你是使用了指定前端框架或者插件时，请一定看其文档，因为不同的框架要求返回前端的数据格式是不一样的。

这里api约定：后端返回数据格式：{“path”:””,”url”:””}

|  |
| --- |
| package com.go9.goods.file.controller;  import java.util.HashMap;  import java.util.Map;  import org.apache.commons.io.FilenameUtils;  import org.apache.commons.lang3.RegExUtils;  import org.apache.commons.lang3.StringUtils;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.DeleteMapping;  import org.springframework.web.bind.annotation.PostMapping;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RestController;  import org.springframework.web.multipart.MultipartFile;  import com.go9.common.ServiceMessage;  import com.go9.goods.file.FileService;  @RestController  @RequestMapping("/file")  public class FileController {  @Autowired  private FileService fileService;    @PostMapping("/upload")  public ServiceMessage upload(MultipartFile file){  try {  String fileExtName = FilenameUtils.getExtension(file.getOriginalFilename());  String url = fileService.upload(file.getInputStream(), file.getSize(), fileExtName);  Map<String,String> map = new HashMap<>();  //http://192.168.19.131/group1/M00/00/00/wKgTg1w2w66AZlqCAAJRZnDMCEk197.jpg  String path = StringUtils.substringAfter(url, "group");  path = "group"+path;  map.put("path", path);    //获取url，就是去掉其中的group[1-9]  url = RegExUtils.removePattern(url, "/group[1-9]");  map.put("url", url);  return ServiceMessage.ok(map);  } catch (Exception e) {  e.printStackTrace();  return ServiceMessage.error();  }  }    @DeleteMapping  public ServiceMessage delete(String path) {  try {  fileService.delete(path);  return ServiceMessage.ok();  } catch (Exception e) {  e.printStackTrace();  return ServiceMessage.error();  }  }  } |

## 测试



# 添加商品

前面主要完成了商品的基本信息的添加，商品图片上传功能还未完成，现在存储图片的服务器有了，就可以来实现该功能了。先来看下，整个的上传流程分为2个阶段来完成：

1.jpg

客户端

商品服务

Fastdfs服务器

Springmvc接受文件内容封装成MultipartFile，提取出inputstream用fastdfs客户端再上传至服务器

tracker

storage

http

tcp

上面的第二阶段前面刚刚完成，第一阶段的上传过程前面大家也学过，下面主要是再进行一下整合，以及对外的接口如何发布。

# 接口约定

从逻辑上讲，文件的上传、下载等管理应该是一个单独的原子服务，应该单独出来，由专人编写和维护。但是在项目初期，适当减少服务是个明智的选择，我们从业务上讲把它划到商品的服务里了。

|  |  |  |  |
| --- | --- | --- | --- |
| **业务类型** | **method** | **url** | **return body** |
| 文件上传 | post | files/upload | 返回一个统一的json  对象，  {  “code”: xxx,  “message”:”xxxx”,  “data”:{} or []  } |
| 文件删除 | delete | files?ids |

**特殊说明：文件上传的返回消息格式比较特殊，因为一般前端都会用专业的文件上传组件，进行大量的封装，对后端返回的消息格式有明确的规定，这时就不好再用统一的系统对象返回了，需要遵从指定的返回格式。**

例如大名鼎鼎的KindEditor中的上传插件要求后端返回的数据格式如下：

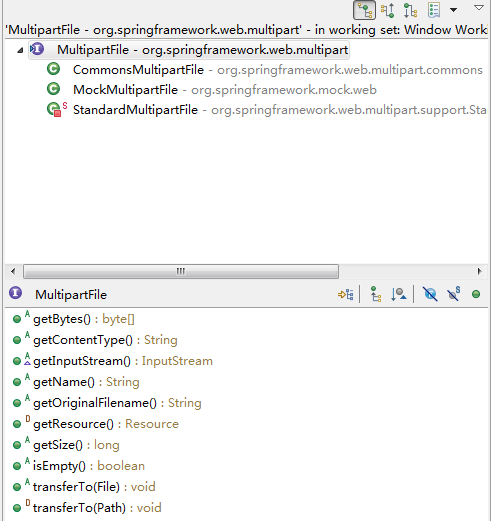
|  |
| --- |
| //成功时  {  "error" : 0,  "url" : "http://www.example.com/path/to/file.ext"  }  //失败时  {  "error" : 1,  "message" : "错误信息"  } |

Go8-admin使用的是 iview的Upload组件，对返回格式无特殊要求！！！

# 上传图片

## Springmvc

复习关键点，springmvc是如何封装要上传的文件对象的，这个封装技术的类组织结构是什么，有哪些关键的属性参数值？？



以上是eclipse下的继承关系，没有“图”，这一点确实要吐槽下eclipse了。要是有像idea一样的类图就好了。不过我个人还是习惯用eclipse，萝卜青菜各有所爱，不管什么IDE只要自己用着顺手就好，不要盲目的跟风，更不要去网上瞎吵吵。

## 代码

暂无

# 商品详情

完成富文本添加功能（补充在前面的代码中）。

对后端来说没有什么难点，同学们重点是要理解富文本编辑器的概念。

这样整个商品的添加功能才算完成了。