第 2-7 课：使用 Spring Boot 上传文件到 FastDFS

什么是 FastDFS

FastDFS 是一个开源的轻量级分布式文件系统，它解决了大数据量存储和负载均衡等问题，特别适合以中小文件（建议范围：4 KB < file_size < 500 MB）为载体的在线服务，如相册网站、视频网站等。在 UC 基于 FastDFS 开发向用户提供了网盘、社区、广告和应用下载等业务的存储服务。

FastDFS 由 C 语言开发，支持 Linux、FreeBSD 等 UNIX 系统类 Google FS，不是通用的文件系统，只能通过专有 API 访问，目前提供了 C、Java 和 PHP API，为互联网应用量身定做，解决了大容量文件存储问题，追求高性能和高扩展性，FastDFS 可以看做是基于文件的 Key Value Pair 存储系统，称作分布式文件存储服务会更合适。

FastDFS 特性

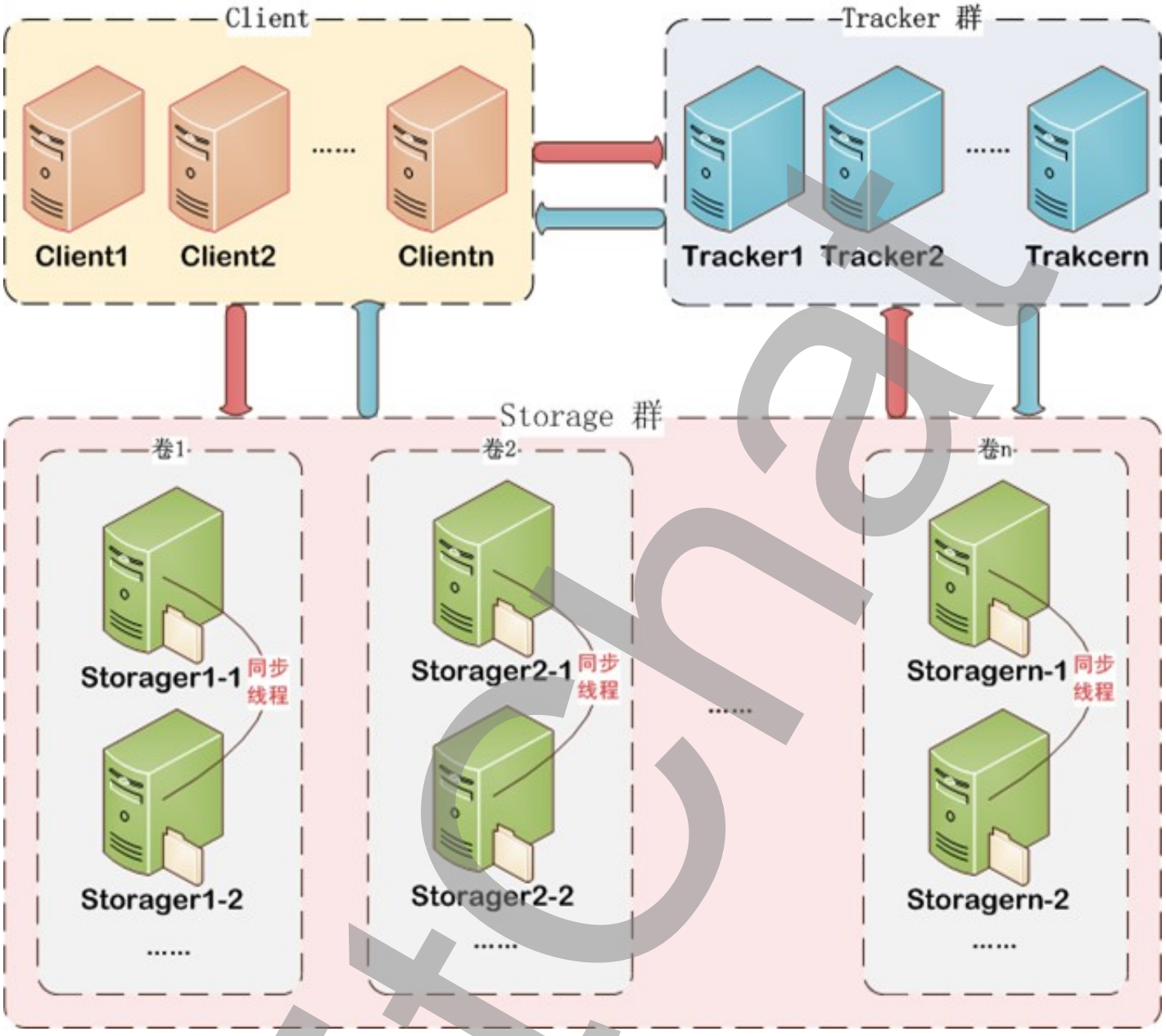
- 文件不分块存储，上传的文件和 OS 文件系统中的文件一一对应
- 支持相同内容的文件只保存一份，节约磁盘空间
- 下载文件支持 HTTP 协议，可以使用内置 Web Server，也可以和其他 Web Server 配合使用
- 支持在线扩容
- 支持主从文件
- 存储服务器上可以保存文件属性（meta-data）V2.0 网络通信采用 libevent，支持大并发访问，整体性能更好

FastDFS 相关概念

FastDFS 服务端有三个角色：跟踪服务器（Tracker Server）、存储服务器（Storage Server）和客户端（Client）。

- **Tracker Server**：跟踪服务器，主要做调度工作，起负载均衡的作用。在内存记录集群中所有存储组和存储服务器的状态信息，是客户端和数据服务器交互的枢纽。相比 GFS 中的 Master 更为精简，不记录文件索引信息，占用的内存量很少。
- **Storage Server**：存储服务器（又称存储节点或数据服务器），文件和文件属性（Meta Data）都保存到存储服务器上。Storage Server 直接利用 OS 的文件系统调用管理文件。
- **Client**：客户端，作为业务请求的发起方，通过专有接口，使用 TCP/IP 协议与跟踪器服务器或存储节点进行数据交互。FastDFS 向使用者提供基本文件访问接口，如 upload、download、append、delete 等，以客户端库的方式提供给用户使用。

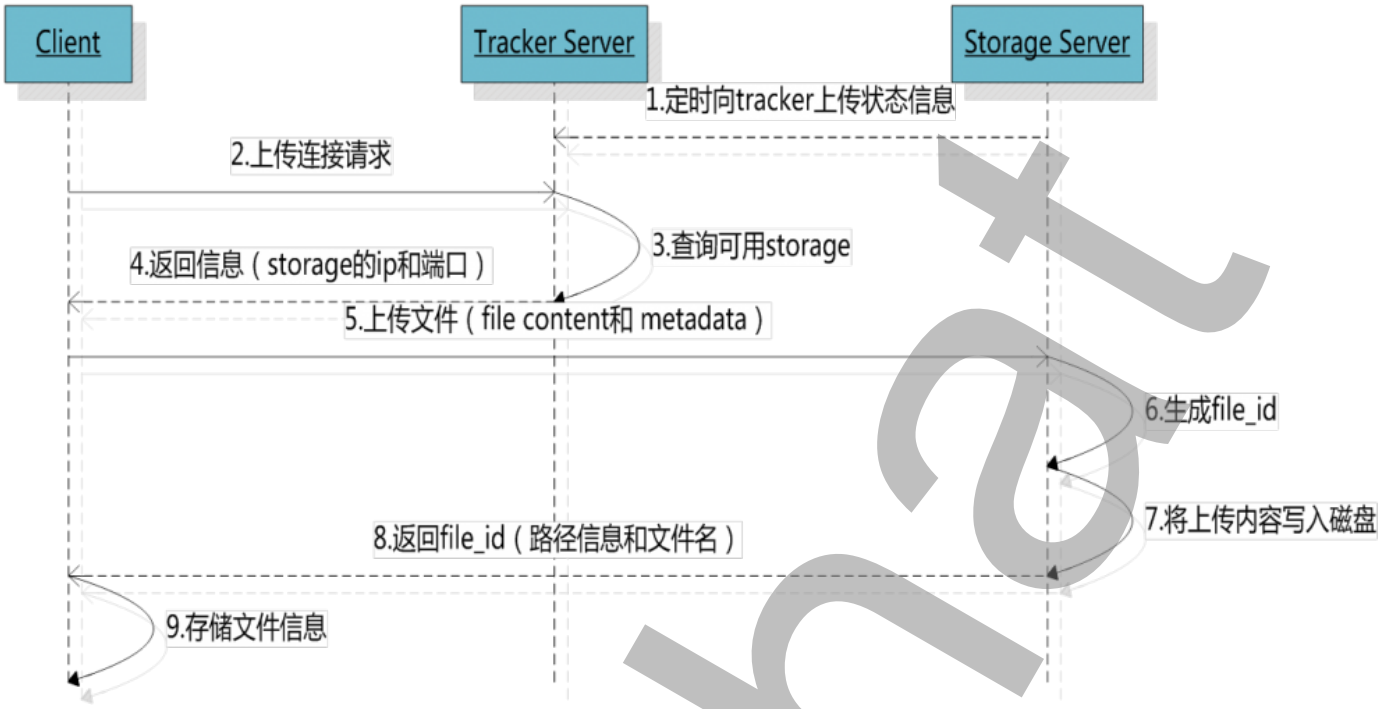
通过一张图来看一下 FastDFS 的运行机制：



Tracker 相当于 FastDFS 的大脑，不论是上传还是下载都是通过 Tracker 来分配资源；客户端一般可以使用 Ngnix 等静态服务器来调用或者做一部分的缓存；存储服务器内部分为卷（或者叫做组），卷与卷之间是平行的关系，可以根据资源的使用情况随时增加，卷内服务器文件相互同步备份，以达到容灾的目的。

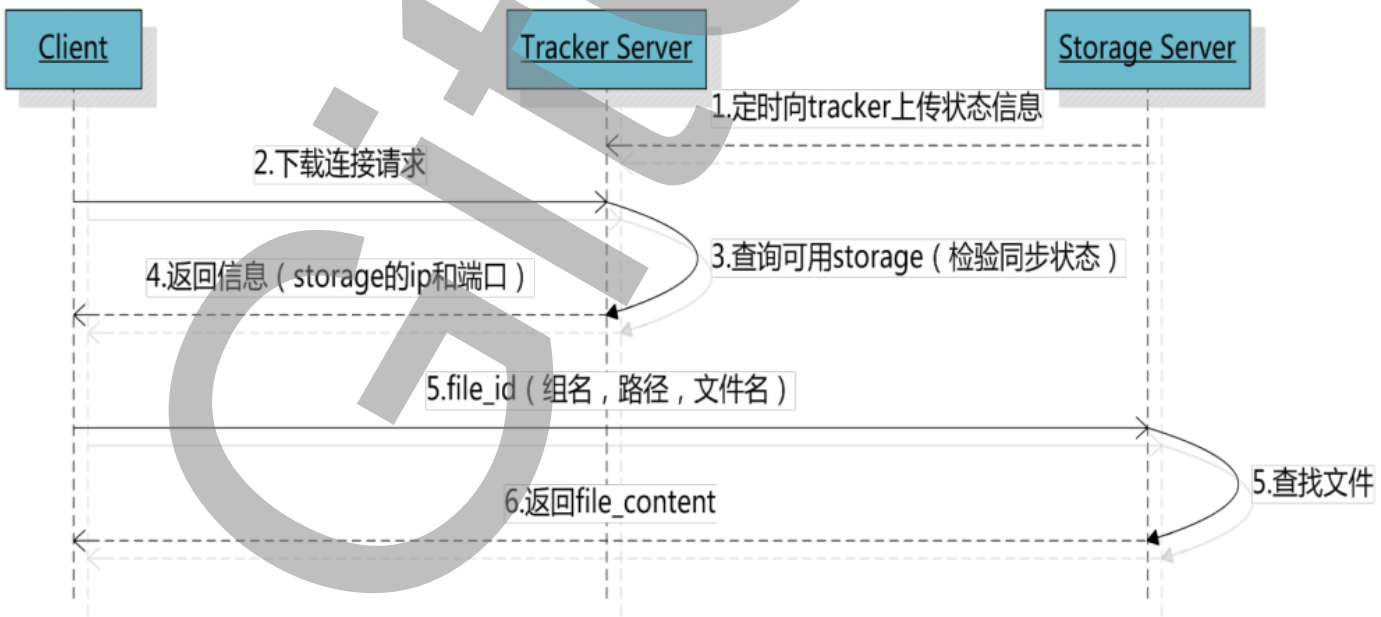
上传机制

首先客户端请求 Tracker 服务获取到存储服务器的 IP 地址和端口，然后客户端根据返回的 IP 地址和端口号请求上传文件，存储服务器接收到请求后生产文件，并且将文件内容写入磁盘并返回给客户端 file_id、路径信息、文件名等信息，客户端保存相关信息上传完毕。



下载机制

客户端带上文件名信息请求 Tracker 服务获取到存储服务器的 IP 地址和端口，然后客户端根据返回的 IP 地址和端口号请求下载文件，存储服务器接收到请求后返回文件给客户端。



Spring Boot 集成 FastDFS

本节示例项目会在上一节项目的基础上进行构建开发，从前端上传文件到后台后，直接传递到 FastDFS 集群中，并返回文件存储的地址。

pom 包配置

引入 FastDFS 的 Java 客户端：

```
<dependency>
    <groupId>org.csource</groupId>
    <artifactId>fastdfs-client-java</artifactId>
    <version>1.27-SNAPSHOT</version>
</dependency>
```

fastdfs-client-java 为 FastDFS 的 Java 客户端，用来和 FastDFS 集群进行交互。

FastDFS 配置

项目 resources 目录下添加 `fdfs_client.conf` 文件：

```
connect_timeout = 60 # 连接超时时间
network_timeout = 60 # 网络超时时间
charset = UTF-8 # 编码格式
http.tracker_http_port = 8080 #tracker 端口
http.anti_steal_token = no #token 防盗链功能
http.secret_key = 123456 #密钥
# tracer server 列表，多个 tracer server 的话，分行列出
tracker_server = 192.168.53.85:22122
tracker_server = 192.168.53.86:22122
```

配置文件设置了连接的超时时间、编码格式以及 tracker_server 地址等信息。

详细内容参考：fastdfs-client-java。

封装 FastDFS 上传工具类

封装 FastDFSFile，文件基础信息包括文件名、内容、文件类型、作者等。

```
public class FastDFSFile {
    private String name;
    private byte[] content;
    private String ext;
    private String md5;
    private String author;
    //省略getter、setter
}
```

接下来封装 FastDFSClient 类，FastDFSClient 主要封装最基础的操作，包含上传、下载、删除等方法。FastDFSFile 任务可以是 FastDFS 上传文件的封装，操作时每一个文件对应一个实例。

首先在类加载的时候读取配置信息，并进行初始化。

```
static {
    try {
        String filePath = new ClassPathResource("fdfs_client.conf").getFile().getA
bsolutePath();;
        ClientGlobal.init(filePath);
    } catch (Exception e) {
        logger.error("FastDFS Client Init Fail!",e);
    }
}
```

> ClientGlobal.init 方法会读取配置文件，并初始化对应的属性。

FastDFSClient 类中第一个方法——文件上传。

1.文件上传

使用 FastDFS 提供的客户端 storageClient 来进行文件上传，最后将上传结果返回。

```
public static String[] upload(FastDFSFile file) {
    logger.info("File Name: " + file.getName() + "File Length:" + file.getContent(
).length);

    //文件属性信息
    NameValuePair[] meta_list = new NameValuePair[1];
    meta_list[0] = new NameValuePair("author", file.getAuthor());

    long startTime = System.currentTimeMillis();
    String[] uploadResults = null;
    StorageClient storageClient=null;
    try {
        //获取
        storageClient = getStorageClient();
        //上传
        uploadResults = storageClient.upload_file(file.getContent(), file.getExt()
, meta_list);
    } catch (IOException e) {
        logger.error("IO Exception when uploadind the file:" + file.getName(), e);
    } catch (Exception e) {
        logger.error("Non IO Exception when uploadind the file:" + file.getName(),
 e);
    }
    logger.info("upload_file time used:" + (System.currentTimeMillis() - startTime
) + " ms");

    //验证上传结果
    if (uploadResults == null && storageClient!=null) {
        logger.error("upload file fail, error code:" + storageClient.getErrorCode(
));
    }
    //上传文件成功会返回 groupName。
    logger.info("upload file successfully!!!" + "group_name:" + uploadResults[0] +
 ", remoteFileName:" + " " + uploadResults[1]);
    return uploadResults;
}
```

其中：

- NameValuePair，主要存储文件的一些基础属性，如作者信息、创建时间等；
- getStorageClient()，封装了获取客户端的方法。

首先获取 TrackerServer 信息，使用 TrackerServer 构建出每次操作的客户端实例 StorageClient。详细代码如下：

```
private static StorageClient getStorageClient() throws IOException {
    TrackerServer trackerServer = getTrackerServer();
    StorageClient storageClient = new StorageClient(trackerServer, null);
    return  storageClient;
}
```

下面为封装获取 TrackerServer 的方法：

```
private static TrackerServer getTrackerServer() throws IOException {
    TrackerClient trackerClient = new TrackerClient();
    TrackerServer trackerServer = trackerClient.getConnection();
    return  trackerServer;
}
```

2.获取文件

根据 groupName 和文件名获取文件信息。group 也可称为卷，同组内服务器上的文件是完全相同的，同一组内的 storage server 之间是对等的，文件上传、删除等操作可以在任意一台 storage server 上进行。

```
public static FileInfo getFile(String groupName, String remoteFileName) {
    try {
        storageClient = new StorageClient(trackerServer, storageServer);
        return storageClient.get_file_info(groupName, remoteFileName);
    } catch (IOException e) {
        logger.error("IO Exception: Get File from Fast DFS failed", e);
    } catch (Exception e) {
        logger.error("Non IO Exception: Get File from Fast DFS failed", e);
    }
    return null;
}
```

3.下载文件

根据 storageClient 的 API 获取文件的字节流并返回：

```
public static InputStream downFile(String groupName, String remoteFileName) {
    try {
        StorageClient storageClient = getStorageClient();
        byte[] fileByte = storageClient.download_file(groupName, remoteFileName);
        InputStream ins = new ByteArrayInputStream(fileByte);
        return ins;
    } catch (IOException e) {
        logger.error("IO Exception: Get File from Fast DFS failed", e);
    } catch (Exception e) {
        logger.error("Non IO Exception: Get File from Fast DFS failed", e);
    }
    return null;
}
```

4.删除文件

根据文件名和组删除对应的文件。

```
public static void deleteFile(String groupName, String remoteFileName)
        throws Exception {
    StorageClient storageClient = getStorageClient();
    int i = storageClient.delete_file(groupName, remoteFileName);
    logger.info("delete file successfully!!!" + i);
}
```

当使用 FastDFS 时，直接调用 FastDFSClient 对应的方法即可。

编写上传控制类

从 MultipartFile 中读取文件信息，然后使用 FastDFSClient 将文件上传到 FastDFS 集群中，封装一个 saveFile() 方法用来调用上面封装的 FastDFS 工具类，将 MultipartFile 文件上传到 FastDFS 中，并返回上传后文件的地址信息。

```
public String saveFile(MultipartFile multipartFile) throws IOException {
    String[] fileAbsolutePath={};
    String fileName=multipartFile.getOriginalFilename();
    String ext = fileName.substring(fileName.lastIndexOf(".") + 1);
    byte[] file_buff = null;
    InputStream inputStream=multipartFile.getInputStream();
    if(inputStream!=null){
        int len1 = inputStream.available();
        file_buff = new byte[len1];
        inputStream.read(file_buff);
    }
    inputStream.close();
    FastDFSFile file = new FastDFSFile(fileName, file_buff, ext);
    try {
        fileAbsolutePath = FastDFSClient.upload(file);  //upload to fastdfs
    } catch (Exception e) {
        logger.error("upload file Exception!",e);
    }
    if (fileAbsolutePath==null) {
        logger.error("upload file failed,please upload again!");
    }
    String path=FastDFSClient.getTrackerUrl()+fileAbsolutePath[0]+ "/"+fileAbsolut
ePath[1];
    return path;
}
```

当上传请求传递到后端时，调用上面方法 saveFile()。

```
@PostMapping("/upload")
public String singleFileUpload(@RequestParam("file") MultipartFile file,
                               RedirectAttributes redirectAttributes) {
    if (file.isEmpty()) {
        redirectAttributes.addFlashAttribute("message", "Please select a file to u
pload");
        return "redirect:uploadStatus";
    }
    try {
        String path=saveFile(file);
        redirectAttributes.addFlashAttribute("message",
                "You successfully uploaded '" + file.getOriginalFilename() + "'");
        redirectAttributes.addFlashAttribute("path","file path url '" + path + "'"
);
    } catch (Exception e) {
        logger.error("upload file failed",e);
    }
    return "redirect:/uploadStatus";
}
```

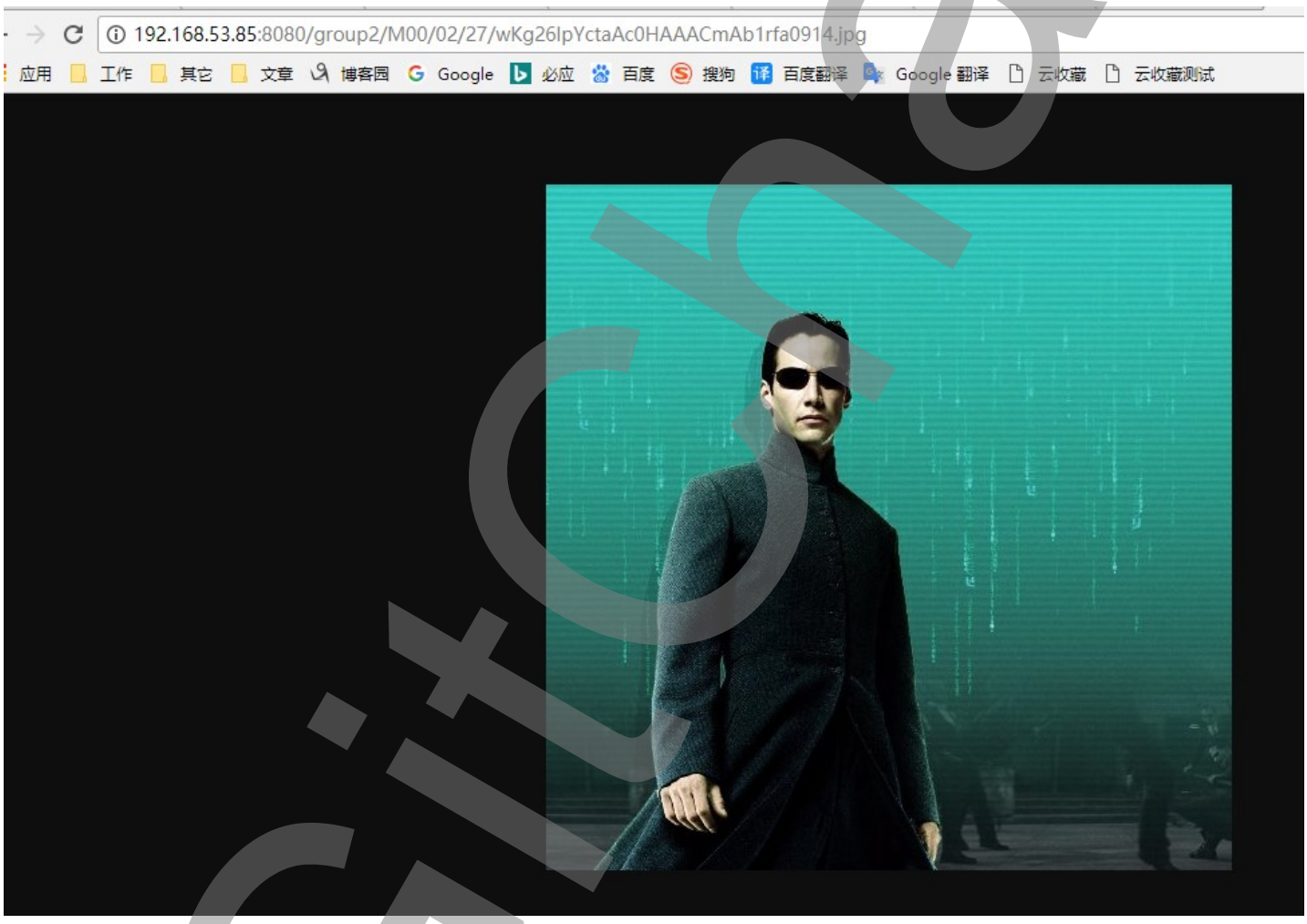
上传成功之后，将文件的路径展示到页面，效果图如下：

Spring Boot - Upload Status

You successfully uploaded 'neo.jpg'

file path url 'http://192.168.53.85:8080/group2/M00/02/27/wKg26lpYctaAc0HAAACmAb1rfa0914.jpg'

在浏览器中访问此 URL，可以看到成功通过 FastDFS 展示：



在实际项目使用中可以给 Tracker 配置好固定域名，将返回的地址信息存储到数据库中，前端业务调用时直接获取地址展示即可。

总结

整体上传逻辑：在页面上传文件后台由 MultipartFile 接收，接着将 MultipartFile 文件转发为 FastDFS 文件封装类 FastDFSFile，调用 FastDFSClient 类的封装方法，将文件（FastDFSFile）上传到 FastDFS 集群中，成功后集群中文件存储位置返回到页面。

FastDFS 是一款非常优秀的中小文件存储系统，结合 Spring Boot 的相关特性很容易将 FastDFS 集成到项目

中，方便前端业务进行处理。

> 点击这里下载源码。