电子科技大学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

本科学位论文

BACHELOR THESIS

论文题目 基于 tus 协议的断点续传客户端软件

学科专业 计算机科学与技术

学　　号 2014060107011

作者姓名 唐治中

指导老师 丘志杰　高级工程师

分类号 _____ 密级 _____

UDC 注 1 _____

学　位　论　文

基于 tus 协议的断点续传客户端软件

（题名和副题名）

唐治中

（作者姓名）

指导老师　丘志杰　高级工程师

（姓名、职称、单位名称）

申请学位级别　本科　学科专业　计算机科学与技术

提交论文日期 _____ 论文答辩日期 _____

学位授予单位和日期　电子科技大学

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

摘　要

为了适应当下日益增长的用户对文件传输的多样化需求，断点续传功能作为一项保证文件传输质量的技术得到了广泛的应用。本文以 tus 传输协议为研究课题，实现了基于 tus 协议的断点续传客户端软件的开发，重点实现了文件传输过程中按块传输的方法，通过配置文件记录传输位置以便继续传输，使用多线程传输文件等。本软件使用 JAVA 语言实现，实现内容分为四部分，分别是协议基本内容，具体功能和模块，软件的编程实现详述，以及测试软件功能和完善。

其中，tus 协议的基本内容是对 tus 协议的解读与理解，包含了请求格式的详细介绍。软件的具体功能和模块设计部分介绍了本软件的需求分析，类的详细划分，以及程序的主要流程和对应类间的调用关系。软件的编程实现内容中，主要展示了软件核心类的代码详述，对应各个功能模块的代码实现。在软件测试部分中，通过编写测试用例的方式测试了主要函数的功能正确性，以及对软件的稳定性的简单测试。

关键词： tus 协议，断点续传，JAVA，配置文件，多线程

ABSTRACT

In order to adapt to the ever-increasing user demand for the diversification of file transfer, the breakpoint resume function has been widely used as a technology for guaranteeing the file transfer quality. This article takes the tus transmission protocol as the research topic, realizes the development of the client software of the broken point resuming client based on the tus protocol, focuses on the method of block transmission in the file transmission process, records the transmission position through the configuration file to continue transmission, and uses multithreading transfer files and so on. The software is implemented in JAVA language. The implementation is divided into four parts, which are the basic contents of the agreement, specific functions and modules, detailed programming of the software, and testing software functions and improvements.

Among them, the basic content of the tus protocol is the interpretation and understanding of the tus protocol, including a detailed description of the request format. The software's specific function and module design section introduces the software's requirement analysis, the detailed division of the class, and the main flow of the program and the corresponding relationship between the classes. In the programming and implementation content of the software, the code of the core class of the software is shown in detail, corresponding to the code implementation of each functional module. In the software testing part, the function correctness of the main function is tested by writing test cases, and a simple test of the stability of the software is performed.

Keywords: file transfer, breakpoint resume, tus transmission protocol, transmission by block, multi-threaded transfer, programming environment

目 录

第一章 绪 论.....1
1.1 研究工作的背景.....1
1.2 断点续传的现状与意义.....1
1.3 实现本软件的主要工作内容.....2
1.4 本论文的结构安排.....2
第二章 tus 协议基础.....3
2.1 tus 核心协议.....3
2.1.1 tus 请求.....3
2.1.1.1 HEAD.....3
2.1.1.2 PATCH.....4
2.1.1.3 OPTION.....4
2.2 tus 协议的头部信息.....5
2.3 tus 协议扩展.....6
2.3.1 新建上传.....6
2.3.1.1 举例.....6
2.3.1.2 头部信息.....6
2.3.1.3 要求.....7
2.3.2 过期处理.....7
2.3.2.1 举例.....7
2.3.2.2 头部信息.....8
2.3.3 校验处理.....8
2.3.3.1 头部信息.....9
2.3.3.2 举例.....9
2.3.4 终止处理.....9
2.3.4.1 需求.....10
2.3.4.2 举例.....10
2.3.5 级联处理.....10
2.3.5.1 头部信息.....11
2.3.5.2 举例.....11
2.4 关于 base64.....13

2.5 JAVA 语言的环境配置 14
2.5.1 JAVA 工具包 JDK 14
2.5.2 JAVA 运行环境 JRE 14
2.5.3 JDK 和 JRE 的安装 15
2.5.3.1 关于 PC 配置要求 15
2.5.4 环境变量的配置 15
2.6 本章小结 16
第三章 软件的设计 17
3.1 需求分析与实现原理 17
3.1.1 软件的需求分析 17
3.2 软件的模块设计 19
3.2.1 用户交互模块 20
3.2.2 传输设置模块 20
3.2.3 异常处理模块 21
3.2.4 线程管理模块 21
3.2.5 分片传输模块 21
3.2.6 文件处理模块 22
3.3 主要类的设计 22
3.3.1 TusClient 类的设计 22
3.3.2 Window 类的设计 23
3.3.3 TusUpload 类的设计 23
3.3.4 JProcessBar 类的设计 24
3.3.5 TusUploadRun 类的设计 25
3.3.6 主要类间的关系 26
3.4 软件流程设计 27
3.4.1 UML 流程图 27
3.5 本章小结 29
第四章 软件的实现 30
4.1 程序主要类的介绍 30
4.1.1 Window 类 30
4.1.2 MyThreadFactory 类 32
4.1.3 TusUploadRun 类 32
4.1.4 TusUpload 类 34

4.1.5 TusUploader 类 35
4.1.6 关于配置文件 36
4.2 本章小结 37
第五章 软件的测试 38
5.1 软件测试的概念 38
5.2 本软件的测试 38
5.2.1 软件测试用例 38
5.2.2 软件测试环境 38
5.2.3 软件的详细测试 38
5.2.3.1 TestTusClient 测试类 38
5.2.3.2 TestTusUpload 测试类 40
5.3 软件的使用 42
5.4 本章小结 43
第六章 全文总结与展望 44
6.1 全文总结 44
6.2 后续工作展望 44
致 谢 45
参考文献 46
外文资料原文 47
外文资料译文 49

第一章 绪 论

1.1 研究工作的背景

HTTP 超文本传输协议，是一种用于分布式、协作式和超媒体信息系统的应用层协议。HTTP 是万维网的数据通信的基础。设计 HTTP 最初的目的是为了提供一种发布和接收 HTML 页面的方法。通过 HTTP 或者 HTTPS 协议请求的资源由统一资源标识符来标识。而如今的 http 协议的发展前景是，将 web 服务器内置到嵌入式工控节点，由网络服务器提供交互式 Internet 服务，例如提供符合 http 协议的用户远程监控界面和信息交互服务。在 web 嵌入式远程监控系统中[1]，将被监控的各底层状态定义成 HTML 语言许可的网络变量，然后利用这些变量生成网页，由网络服务器提供给远程用户，远程用户使用网络浏览器，下载服务器上的页面，以观察设备的实时运行情况，远程修改这些变量，完成对现场设备的监控。而对于这种 http 协议的嵌入式应用同样也需要断点续传的支持，使这些设备监控功能更加完善。

通过 HTTP，可以非常方便的实现断点续传。断点续传主要依赖于 HTTP 头部定义的 Range 来完成。具体 Range 的说明参见 RFC2616 中 14.35.2 节，在请求某范围内的资源时，可以更有效地对大资源发出请求或从传输错误中恢复下载。有了 Range，应用可以通过 HTTP 请求曾经获取失败的资源的某一个返回或者是部分，来恢复下载该资源。当然并不是所有的服务器都支持 Range，但大多数服务器是可以的。Range 是以字节计算的，请求的时候不必给出结尾字节数，因为请求方并不一定知道资源的大小。

1.2 断点续传的现状与意义

断点续传：指的是在上传/下载时，将任务（一个文件或压缩包）人为的划分为几个部分，每一个部分采用一个线程进行上传/下载，如果碰到网络故障，可以从已经上传/下载的部分开始继续上传/下载未完成的部分，而没有必要从头开始上传/下载。可以节省时间，提高速度。

有时用户上传/下载文件需要历时数小时，万一线路中断，不具备断点续传的 HTTP/FTP 服务器或下载软件就只能从头重传，比较好的 HTTP/FTP 服务器或下载软件具有断点续传能力，允许用户从上传/下载断线的地方继续传送，这样大大减少了用户的烦恼。

本课题的 tus 断点续传功能的实现对于现如今的网络时代有着非常重要的意

义，只要是通过网络进行交流的终端间，和使用 http 协议进行传输的终端间，就存在着因特殊情况而出现的传输中断，而断点续传功能的应用就可以有效的避免这些因为硬件故障或人为操作失误所带来的损失，从而提高用户间的交流效率，使网络空间更加高效便利可靠。

1.3 实现本软件的主要工作内容

本软件利用 JAVA 语言，以及 tus 官方提供的 tus 服务器端软件，实现了客户端的基本功能，主要工作内容如下：

本软件基本实现了 tus 协议规定的文件传输要求，例如按照 tus 协议内容规范请求报文以及返回信息格式，用 JAVA 实现了断点续传的基本功能，核心类封装了传输所需的信息，提供了参数及接口，便于在此基本功能上进行拓展。突出点在于使用了多线程技术管理传输过程。

1.4 本论文的结构安排

本文的章节结构安排如下：

第一章绪论主要介绍了本课题的工作背景，断点续传功能的现状及意义，本设计的主要工作等。第二章介绍了 tus 的协议基础，第三章详细描述了软件的功能及模块设计，第四章着重展示通过具体的编程语言来实现软件功能和模块的过程。第五章展示了软件的测试与成果展示。

第二章 tus 协议基础

tus 协议提供了一种通过 HTTP / 1.1（RFC 7230）[2] 和 HTTP / 2（RFC 7540）[3] 进行可恢复文件上传的机制。

2.1 tus 核心协议

核心协议介绍了如何恢复中断上传。它假定您已经有了上传的 URL，通常是通过创建扩展名创建的。所有客户端和服务器必须实施核心协议。本规范没有描述 URL 的结构，因为这是由具体实现决定的。本文档中显示的所有 URL 仅用于举例目的。另外，服务器决定执行认证和授权 [4]。

2.1.1 tus 请求

2.1.1.1 HEAD

即使偏移量为 0，或者上传已被视为已完成，服务器必须始终在 HEAD 请求的响应中包含上载偏移头。如果上传的大小已知，则服务器必须在响应中包含 Upload-Length 头。如果没有找到该资源，服务器应该返回 404 Not Found，410 Gone 或 403 Forbidden 状态，而没有 Upload-Offset 头。

服务器必须通过向响应添加 Cache-Control：no-store 头来防止客户端和/或代理缓存响应。

HEAD 请求用于确定应继续上传的偏移量。下面的示例显示了在传输 70 个字节后中断的 100 字节上传的继续。

Request

```
HEAD /files/24e533e02ec3bc40c387f1a0e460e216 HTTP/1.1
Host: tus.example.org
Tus-Resumable: 1.0.0
```

Response

```
HTTP/1.1 200 OK
Upload-Offset: 70
Tus-Resumable: 1.0.0
```

2.1.1.2 PATCH

服务器应该接受针对任何上传 URL 的 PATCH 请求，并在由上传偏移头指定的给定偏移量处应用消息中包含的字节。所有的 PATCH 请求必须使用 Content-Type：application / offset + octet-stream。

上传偏移报头的值必须等于资源的当前偏移量。为了实现并行上传，可以使用级联扩展。如果偏移量不匹配，服务器必须响应 409 冲突状态而不修改上传资源。

客户端应该在一个 PATCH 请求中发送所有剩余的上传字节，但也可以在需要的情况下连续使用多个小的请求。这种情况的一个例子是当使用 Checksum 扩展时。

服务器务必确认成功的 PATCH 请求 204 无内容状态。它必须包含包含新偏移量的 Upload-Offset 头。新的偏移量必须是 PATCH 请求之前的偏移量与当前 PATCH 请求期间接收和处理或存储的字节数之和。

客户端和服务器都应该尝试检测并处理可预测的网络错误。他们可以通过检查读/写套接字错误以及设置读/写超时来做到这一点。超时应该通过关闭底层连接来处理。

服务器应该总是尝试存储尽可能多的接收数据。

鉴于偏移量，客户端使用 PATCH 方法恢复上传：

Request

```
PATCH /files/24e533e02ec3bc40c387f1a0e460e216 HTTP/1.1
Host: tus.example.org
Content-Type: application/offset+octet-stream
Content-Length: 30
Upload-Offset: 70
Tus-Resumable: 1.0.0
```

Response

```
HTTP/1.1 204 No Content
Tus-Resumable: 1.0.0
Upload-Offset: 100
```

2.1.1.3 OPTION

OPTIONS 请求可以用来收集关于服务器当前配置的信息。由 204 无内容或 200 OK 状态指示的成功响应必须包含 Tus-Version 标头。它可能包含 Tus-Extension 和 Tus-Max-Size 标题。

客户端不应该在请求中包含 Tus-Resumable 头，服务器必须忽略头。

本示例阐明了对 OPTIONS 请求的响应。在请求和响应中使用的版本是 1.0.0，而服务器也能够处理 0.2.2 和 0.2.1。上传总数最多为 1GB，允许创建和过期的扩展。

Request

```
OPTIONS /files HTTP/1.1
Host: tus.example.org
```

Response

```
HTTP/1.1 204 No Content
Tus-Resumable: 1.0.0
Tus-Version: 1.0.0,0.2.2,0.2.1
Tus-Max-Size: 1073741824
Tus-Extension: creation,expiration
```

2.2 tus 协议的头部信息

头部信息 Headers 中包含了在发送请求和返回相应信息中的关键字。

1.Upload-Offset：Upload-Offset 请求和响应头指示资源内的字节偏移。该值必须是一个非负整数。

2.Upload-Length：上传长度请求和响应标头以字节表示整个上传的大小。该值必须是一个非负整数。

3.Tus-Version：Tus-Version 响应头必须是服务器支持的协议版本的逗号分隔列表。该列表务必按照服务器的首选项排序，其中第一个是最优选的。

4.Tus-Resumable：除 OPTIONS 请求外，Tus 可恢复头部必须包含在每个请求和响应中。该值必须是客户端或服务器使用的协议版本。

如果客户端指定的版本不被服务器支持，它必须响应 412 Precondition Failed 状态，并且必须在响应中包含 Tus-Version 标头。另外，服务器不能处理请求。

5.Tus-Extension：Tus-Extension 响应头必须是服务器支持的扩展名的逗号分隔列表。如果不支持扩展，则必须省略 Tus-Extension 头。

6.Tus-Max-Size：Tus-Max-Size 响应头必须是一个非负整数，指示整个上传允许的最大字节大小。如果存在已知的硬限制，服务器应该设置该标头。

7.X-HTTP-Method-Override：如果头部出现，X-HTTP-Method-Override 请求头必须是一个必须被服务器解释为请求方法的字符串。请求的实际方法必须被忽略。如果客户端环境不支持 PATCH 或 DELETE 方法，客户端应该使用这个头文件。

2.3 tus 协议扩展

鼓励客户和服务器尽可能多地实现扩展。功能检测应该通过客户端发送 OPTIONS 请求和服务器响应 Tus-Extension 头来实现。

2.3.1 新建上传

客户端和服务器应该实现上传创建扩展。如果服务器支持这个扩展，它必须添加创建到 Tus-Extension 头。

2.3.1.1 举例

一个空的 POST 请求用于创建一个新的上传资源。上传长度标题以字节表示整个上传的大小。

Request

```
POST /files HTTP/1.1
Host: tus.example.org
Content-Length: 0
Upload-Length: 100
Tus-Resumable: 1.0.0
Upload-Metadata: filename d29ybGRfZG9taW5hdGlvbl9wbGFuLnBkZg==
```

Response

```
HTTP/1.1 201 Created
Location: https://tus.example.org/files/24e533e02ec3bc40c387f1a0e460e216
Tus-Resumable: 1.0.0
```

新资源的隐含偏移量为 0，允许客户端使用核心协议来执行实际上传。

2.3.1.2 头部信息

1.Upload-Defer-Length：Upload-Defer-Length 请求和响应标头指示上传的大小当前不知道，并且稍后将被传输。它的值必须是 1. 如果上传的长度没有延迟，这个头部必须省略。

2.Upload-Metadata：上传元数据请求和响应头必须由一个或多个以逗号分隔的键值对组成。密钥和值必须用空格分隔。密钥不能包含空格和逗号，也不能为空。该键应该是 ASCII 编码的，并且该值必须是 Base64 编码的。所有密钥必须是唯一的。

2.3.1.3 要求

客户端必须发送一个针对已知上传创建 URL 的 POST 请求来请求一个新的上传资源。该请求必须包含以下标题之一：

a）上传长度以字节表示整个上传的大小。

b）上传 - 推迟 - 长度：如果上传大小当时未知，则为 1。一旦知道客户端必须在下一个 PATCH 请求中设置上传长度标题。一旦设定的长度不能改变。只要上传长度未知，服务器必须在对 HEAD 请求的所有响应中设置上传延迟长度：1。

如果服务器支持推迟长度，它必须添加创建 - 延迟长度到 Tus-Extension 报头。

客户端可以提供上传 - 元数据头部以向上传创建请求添加额外的元数据。服务器可以决定忽略或使用这些信息来进一步处理请求或拒绝它。如果上传包含额外的元数据，对 HEAD 请求的响应必须包含上传 - 元数据头及其在创建期间由客户指定的值。

如果上传的长度超过了最大值（可以使用 Tus-Max-Size 头部指定），那么服务器必须响应 413 请求实体太大的状态。

服务器必须通过 201 Created 状态确认成功上传创建。服务器必须将 Location 标头设置为所创建资源的 URL。这个网址可能是绝对的或相对的。

客户端必须使用核心协议执行实际上传。

2.3.2 过期处理

服务器可以在它们到期后删除未完成的上传。为了向客户端表明这种行为，服务器必须将过期添加到 Tus-Extension 报头。

2.3.2.1 举例

在 Upload-Expires 中指定的时间之前，未完成的上传可用。此日期之后，上传无法恢复。

Request

```
PATCH /files/24e533e02ec3bc40c387f1a0e460e216 HTTP/1.1
Host: tus.example.org
Content-Type: application/offset+octet-stream
Content-Length: 30
Upload-Offset: 70
Tus-Resumable: 1.0.0
```

Response

```
HTTP/1.1 204 No Content
```

```
Upload-Expires: Wed, 25 Jun 2014 16:00:00 GMT
Tus-Resumable: 1.0.0
Upload-Offset: 100
```

2.3.2.2 头部信息

1.Upload-Expires：Upload-Expires 响应标题指示未完成上载过期的时间。服务器可能希望在一段时间后删除不完整的上传，以防止被放弃的上传占用额外的存储空间。客户端应该使用这个头部来确定在尝试恢复上传之前上传是否仍然有效。

如果上传过期，该头部必须包含在每个 PATCH 响应中。如果在创建时已知过期，则 Upload-Expires 标头必须包含在对初始 POST 请求的响应中。它的值可能会随着时间而改变。

如果客户端尝试恢复已被服务器删除的上载，则服务器应该以 404 Not Found 或 410 Gone 状态进行响应。如果服务器正在跟踪过期的上传，则应使用后者。在这两种情况下，客户端都应该开始新的上传。

Upload-Expires 头的值必须采用 RFC 7231 datetime 格式。

2.3.3 校验处理

客户端和服务器可以实现并使用此扩展来验证每个 PATCH 请求的数据完整性。如果支持，服务器必须将校验和添加到 Tus-Extension 头。

客户端可以在 PATCH 请求中包含 Upload-Checksum 标头。一旦收到完整的请求，服务器必须使用指定的算法验证上传的块对照提供的校验和。取决于结果，服务器可以用以下状态码之一响应：1）如果服务器不支持校验和算法，则为 400 错误请求; 2）460 校验和不匹配，如果校验和不匹配; 或者 3）204 如果校验和匹配和数据处理成功。在前两种情况下，必须丢弃上传的块，并且不得更新上传及其偏移量。

服务器必须至少支持由 sha1 标识的 SHA1 校验和算法。校验和算法的名称只能由 ASCII 字符组成，修改后不包括大写字符。

Tus-Checksum-Algorithm 头必须包含在 OPTIONS 请求的响应中。

如果散列不能在上传开始时计算，则可以将其作为预告片加入。如果服务器可以处理预告片，则必须通过向 Tus-Extension 头添加校验和预告片来宣布此行为。预告片，也称为尾部标题，是在请求正文已经传输之后发送的标题。遵循 RFC 7230，它们必须使用尾部标头进行宣布，并且只允许在分块传输中使用。

2.3.3.1 头部信息

1.Tus-Checksum-Algorithm：Tus-Checksum-Algorithm 响应头必须是服务器支持的校验和算法的逗号分隔列表。

2.Upload-Checksum：Upload-Checksum 请求标头包含有关当前主体有效负载校验和的信息。头部必须由用空格分隔的校验和算法的名称和 Base64 编码的校验和组成。

2.3.3.2 举例

Request

```
OPTIONS /files HTTP/1.1
Host: tus.example.org
Tus-Resumable: 1.0.0
```

Response

```
HTTP/1.1 204 No Content
Tus-Resumable: 1.0.0
Tus-Version: 1.0.0
Tus-Extension: checksum
Tus-Checksum-Algorithm: md5,sha1,crc32
```

Request

```
PATCH /files/17f44dbe1c4bace0e18ab850cf2b3a83 HTTP/1.1
Content-Length: 11
Upload-Offset: 0
Tus-Resumable: 1.0.0
Upload-Checksum: sha1 Kq5sNclPz7QV2+lfQIuc6R7oRu0=

hello world
```

Response

```
HTTP/1.1 204 No Content
Tus-Resumable: 1.0.0
Upload-Offset: 11
```

2.3.4 终止处理

该扩展为客户端定义了一种终止已完成和未完成的上传的方式，允许服务器释放已用资源。

如果服务器支持该扩展，则必须通过向 Tus-Extension 头添加终止来宣布。

2.3.4.1 需求

当接收到一个现有上传的 DELETE 请求时，服务器应该释放相关的资源，并且必须响应 204 无内容状态，确认上传已经终止。对于将来对这个 URL 的所有请求，服务器应该用 404 Not Found 或 410 Gone 状态进行响应。

2.3.4.2 举例

Request

```
DELETE /files/24e533e02ec3bc40c387f1a0e460e216 HTTP/1.1
Host: tus.example.org
Content-Length: 0
Tus-Resumable: 1.0.0
```

Response

```
HTTP/1.1 204 No Content
Tus-Resumable: 1.0.0
```

2.3.5 级联处理

该扩展可用于将多个上传连接为一个上传，使客户端可以执行并行上传并上传不连续的块。如果服务器支持这个扩展，它必须添加连接到 Tus-Extension 头。

部分上传代表一个文件块。它通过包含 Upload-Concat：部分头部来创建，同时使用创建扩展创建新的上传。多个部分上传按指定顺序连接成最终上传。服务器不应该处理这些部分上传，直到它们连接在一起形成最终上传。最终上传的长度必须是所有部分上传长度的总和。

为了创建新的最终上传，客户端必须将 Upload-Concat 头添加到上传创建请求中。该值必须是最后的，后跟需要连接的部分上传 URL 的分号和空格分隔的列表。部分上传必须按照列表中指定的顺序进行连接。这个连接请求应该在所有相应的部分上传完成之后发生。客户端不得在最终的上传创建中包含上传长度标题。

当部分上传仍在进行中时，客户端可以发送连接请求。这个特性必须由服务器通过向 Tus 扩展头添加未完成的连接来明确地宣布。

当创建新的最终上传时，部分上传的元数据不会被传输到新的最终上传。所有元数据都应该使用 Upload-Metadata 头部包含在串联请求中。

服务器可以在连接后删除部分上传。然而，它们可能会被多次使用以形成最终资源。

服务器必须以 403 禁止状态响应最终上传 URL 的 PATCH 请求，并且不得修改最终上传或其部分上传。

对最终上载的 HEAD 请求的响应不应包含 Upload-Offset 标头，除非级联已成功完成。成功连接后，必须设置 Upload-Offset 和 Upload-Length，它们的值必须相等。没有为最终上载定义连接前的 Upload-Offset 头的值。

对 HEAD 请求进行部分上传的响应必须包含上传偏移头。

如果可以在请求时计算最终资源的长度，则必须包含上传长度标题。针对部分或最终上传的 HEAD 请求的响应必须包含上传创建请求中收到的 Upload-Concat 标头及其值。

2.3.5.1 头部信息

1.Upload-Concat：Upload-Concat 请求和响应头部必须在部分和最终上传创建请求中设置。它指示上传是部分还是最终上传。如果上传是部分内容，则标头值必须是部分的。在最终上传的情况下，它的值必须是最后的，后跟一个分号和空格分隔的部分上传 URL 列表，它们将被连接起来。部分上传的 URL 可能是绝对的或相对的，并且不能包含 RFC 3986 中定义的空格。

2.3.5.2 举例

在下面的例子中，为了便于阅读，省略了 Host 和 Tus-Resumable 头文件，尽管它们是规范要求的。在开始时创建两个部分上传：

第一部分

```
POST /files HTTP/1.1
Upload-Concat: partial
Upload-Length: 5

HTTP/1.1 201 Created
Location: https://tus.example.org/files/a
```

第二部分

```
POST /files HTTP/1.1
Upload-Concat: partial
Upload-Length: 6

HTTP/1.1 201 Created
Location: https://tus.example.org/files/b
```

现在可以使用 PATCH 请求将数据上传到两个部分资源：

第一部分

```
PATCH /files/a HTTP/1.1
Upload-Offset: 0
Content-Length: 5

hello

HTTP/1.1 204 No Content
```

第二部分

```
PATCH /files/b HTTP/1.1
Upload-Offset: 0
Content-Length: 6

 world

HTTP/1.1 204 No Content
```

在第一个请求中，字符串 hello 已上传，而第二个文件现在包含带有前导空格的“world”。

下一步是创建包含两个较早生成的部分上传的最终上传。在接下来的请求中，没有呈现上传长度标题。

Request and Response

```
POST /files HTTP/1.1
Upload-Concat: final;/files/a /files/b

HTTP/1.1 201 Created
Location: https://tus.example.org/files/ab
```

最终资源的长度现在是 11 个字节，由字符串 hello world 组成。

Request and Response

```
HEAD /files/ab HTTP/1.1

HTTP/1.1 200 OK
Upload-Length: 11
Upload-Concat: final;/files/a /files/b
```

2.4 关于 base64

Base64 是一种基于 64 个可打印字符来表示二进制数据的表示方法。由于 2^6=64，所以每 6 个比特为一个单元，对应某个可打印字符。3 个字节有 24 个比特，对应于 4 个 Base64 单元，即 3 个字节可由 4 个可打印字符来表示。它可用来作为电子邮件的传输编码。在 Base64 中的可打印字符包括字母 A-Z、a-z、数字 0-9，这样共有 62 个字符，此外两个可打印符号在不同的系统中而不同。一些如 uuencode 的其他编码方法，和之后 BinHex 的版本使用不同的 64 字符集来代表 6 个二进制数字，但是不被称为 Base64[5]。

文本	M								a								n							
ASCII 编码	77								97								110							
二进制位	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0
索引	19								22				5				46							
Base64 编码	T								W				F				u							

图 2-1 编码“Man”

如果要编码的字节数不能被 3 整除，最后会多出 1 个或 2 个字节，那么可以使用下面的方法进行处理：先使用 0 字节值在末尾补足，使其能够被 3 整除，然后再进行 Base64 的编码。在编码后的 Base64 文本后加上一个或两个 = 号，代表补足的字节数。也就是说，当最后剩余两个八位字节（2 个 byte）时，最后一个 6 位的 Base64 字节块有四位是 0 值，最后附加上两个等号；如果最后剩余一个八位字节（1 个 byte）时，最后一个 6 位的 base 字节块有两位是 0 值，最后附加一个等号。

文本 (1Byte)	A																							
二进制位	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
二进制位 (补 0)	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Base64 编码	Q								Q				-				-							
文本 (2Byte)	B								C															
二进制位	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0
二进制位 (补 0)	0	1	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0
Base64 编码	Q								K				m				-							

图 2-2 特殊处理

数值	字符	数值	字符	数值	字符	数值	字符
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
0	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

图 2-3 Base64 索引表

2.5 JAVA 语言的环境配置

Java 是一种广泛使用的电脑程式设计语言，拥有跨平台、物件导向、泛型程式设计的特性，广泛应用于企业级 Web 应用开发和移动应用开发。

Java 编程语言的风格十分接近 C++ 语言。继承了 C++ 语言面向对象技术的核心，Java 舍弃了 C++ 语言中容易引起错误的指针，改以引用取代，同时移除了 C++ 中的运算符重载和多重继承特性，改用接口取代，增加垃圾回收器功能。在 Java SE 1.5 版本中引入了泛型编程、类型安全的枚举、不定长参数和自动装/拆箱特性。昇阳电脑对 Java 语言的解释是：“Java 编程语言是个简单、面向对象、分布式、解释性、健壮、安全与系统无关、可移植、高性能、多线程和动态的语言”[6]

2.5.1 JAVA 工具包 JDK

Java Development Kit（JDK）是昇阳电脑针对 Java 开发人员发布的免费软件开发工具包（SDK，Software development kit）。自从 Java 推出以来，JDK 已经成为使用最广泛的 Java SDK。由于 JDK 的一部分特性采用商业许可证，而非开源[7]。

2.5.2 JAVA 运行环境 JRE

Java 执行环境（Java Runtime Environment，简称 JRE）是一个软体，由昇阳电脑所研发，JRE 可以让电脑系统执行 Java 应用程式（Java Application）。

JRE 的内部有一个 Java 虚拟机器（Java Virtual Machine，JVM）以及一些标准

的类别函数库（Class Library）[8]。

2.5.3 JDK 和 JRE 的安装

首先下载[9]安装[10]JDK 和 JRE 的安装包。

2.5.3.1 关于 PC 配置要求

对于 64 位 Windows 平台，JDK 和 JRE 具有最低的处理器，磁盘空间和内存要求。

在 64 位 Windows 平台上安装 JDK 或 JRE 之前，必须验证它是否满足以下最低处理器，磁盘空间和内存要求。

JDK	Installed Image
Development Tools: 64-bit platform	500 MB
Source Code	54.2 MB

图 2-4 安装 jdk 的硬盘要求

JRE	Installed Image
Public Java Runtime Environment	200 MB
Java Update	2 MB

图 2-5 安装 jre 的硬盘要求

2.5.4 环境变量的配置

添加 JAVAHOME 变量：

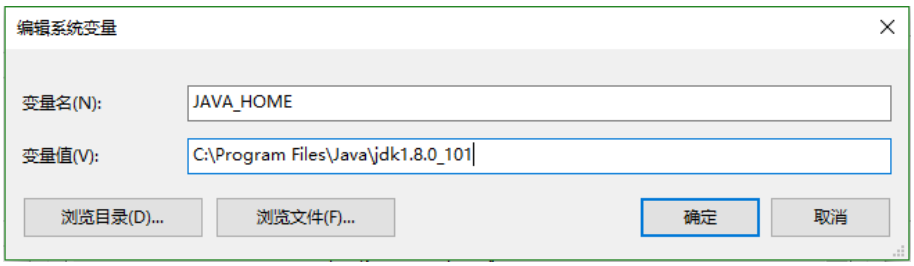

图 2-6 JAVAHOME 变量的添加

添加 Path 变量（如图 3-4 所示）：

在添加好环境变量后，可通过 Windows 的 DOS 窗口或者 PowerShell 使用 java 命令（如图 3-5 所示）：

图 2-7 Path 变量的添加

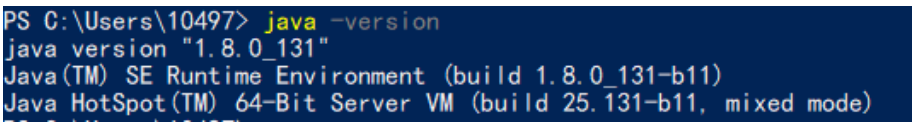

图 2-8 在 Windows 的 PowerShell 中查看 java 命令

2.6 本章小结

本章首先从 tus 协议本身开始，介绍了本断点续传软件遵循的协议理论基础，为软件的功能模块设计以及代码实现做好铺垫。

第三章 软件的设计

3.1 需求分析与实现原理

软件设计是程序员按照特定顺序撰写计算机数据和指令的集合。“软件设计”可以是撰写最基础的二进制 0 和 1 比特；也可以是创建在比特之上的各类软件语言、算法、架构、程序、图像化代码来进行[11]。

3.1.1 软件的需求分析

本软件的结构原理图如图 3-1： 如结构图所示，本客户端软件主要与服务器

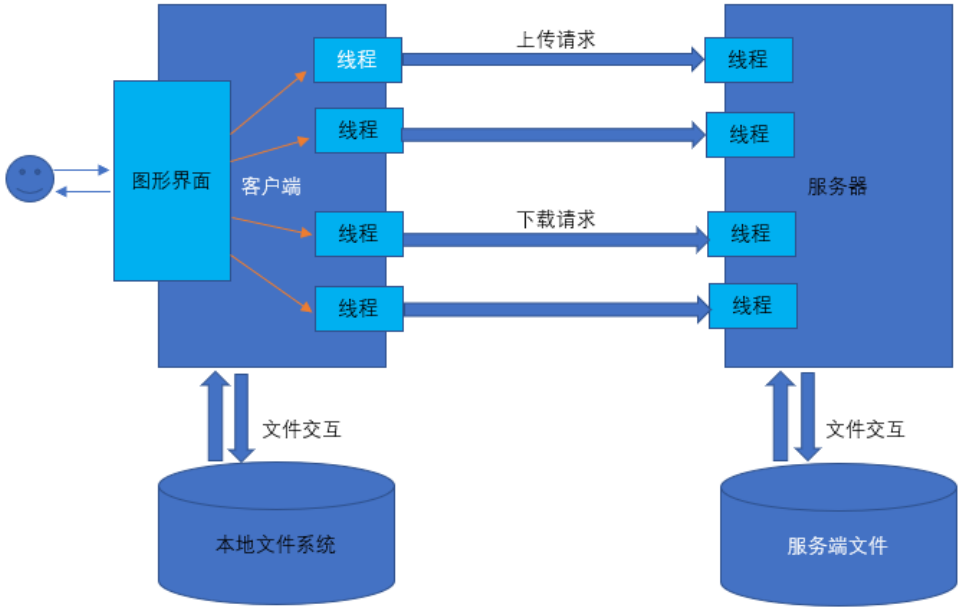

图 3-1 文件传输软件结构图

进行文件的交互，交互过程遵循 tus 协议，其中主要需求为：

1. 需要图形界面，与用户进行交互。

2. 需要管理并使用多线程与服务器发送请求和接受回复。

3. 需要与本地文件系统交互，完成文件的上传和下载，并包含与配置文件相关的操作。

4. 图 3-1 展示出的仅为普通文件传输的结构，要实现断点续传的功能，必须设计出循环按数据块传输文件的操作。

补充说明：由于在这次毕业设计测试中，服务端选择的是官方提供的 tusd 软件，而 tusd 中已经包含了多线程的处理、遵循 tus 协议的请求与回复处理以及对

于相同文件同时请求的锁处理，所以在客户端的编写中就没有进行文件的锁处理，只主要负责了相应的请求，文件的操作，配置文件的设计以及块传输的编写。

关于按块传输：按块传输是断点续传功能的核心，将整个文件划分成固定大小的数据块，可将文件的传输作为一个循环，每次传输固定大小的数据块内容，这样才能在每次循环中记录下 offset 的值，以便在继续传输的时候使用 offset 作为传输的起点。

普通的文件传输如下图 3-2 所示，inputstream 的输入流是连续的概念，于是当中断发生时无法高效的记录此刻文件的传输位置即 offset 的值，这样当下次向继续传输时便无从下手。

图 3-2 普通的文件传输

于是提供了如图 3-3 所示的分块传输的设计：将文件分成固定大小每块相等的数据块，并循环按块传输。这样的好处在于没传输一个数据块的数据之后就可以更新并记录文件传输位置即 offset 的值，当中断发生时，如 3-3 所示的情况，中断在数据块 3 传输过程中发生，这时 offset 记录的值为前一块数据块 2 传输完成后记录的值。此时虽然发生了中断，但程序成功记录了文件传输的 offset 值。

因此，在下次需要继续传输的时候就可以通过 offset 值使传输起点从块 2 之后，块 3 之前的位置开始，完成断点续传的任务。

而且，值得注意的是，为了保证断点续传软件的功能完整性，应该考虑到一些特殊情况下的 offset 记录设计。比如设置单独的传输暂停按钮和恢复传输按钮，使软件的功能更加完善。

此外，tusd 官方服务器提供了文件上传时服务端文件 offset 的记录过程，断点续传客户端只需在文件下载过程中提供文件传输位置 offset 的记录过程。

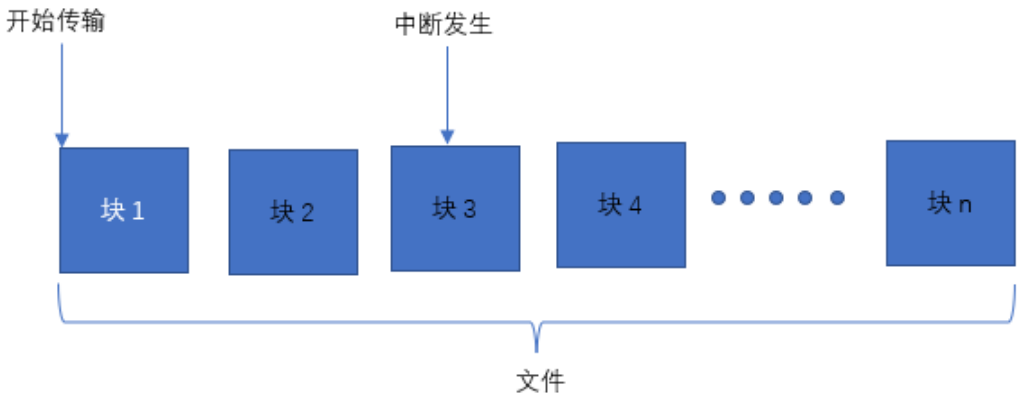

图 3-3 分块的循环文件传输

3.2 软件的模块设计

模块化编程（modular programming）是一种软件设计技术，它将软件分解为若干独立的、可替换的、具有预定功能的模块，每个模块实现一个功能，各模块通过接口（输入输出部分）组合在一起，形成最终程序。对于简单问题，可以直接构建单一模块的程序。而对于复杂问题，则可以先创建若干个较小的模块，然后将它们组装、链接在一起，从而构成复杂的软件系统。模块化编程具有以下优点：

易设计：较大的复杂问题分解为若干较小的简单问题，使我们可以从抽象的模块功能角度而非具体的实现角度去理解软件系统，从而整个系统的结构非常清晰、容易理解，设计人员在设计之初可以更加关注系统的顶层逻辑而非底层细节。

易实现：模块化设计适合团队开发，因为每个团队成员不需要了解系统全貌，只需关注所分配的小任务。另外团队可以灵活地增加人手，新人只需直接接手某个模块，不会影响系统其他模块的开发。

易测试：每个模块不但可以独立开发，也可以独立测试，最后组装时再进行联合测试。

易维护：如果需要修改系统或者扩展系统功能，只需针对特定模块进行修改或者添加新模。

可重用：很多模块的代码都可以不加修改地用于其他程序的开发。

模块化编程实际上是一条抽象设计原则的具体体现，即分离关注点（Separation of Concerns，缩写为 SoC）原则。所谓关注点，是指设计者关心的某个系统特性或行为；而分离关注点是指将系统分解为互不重叠的若干单元，每个单元对应于一个关注点。在模块化编程中，以程序的各个功能作为关注点，模块划分就是分离关注点的结果。一个模块可以使用另一个模块来实现自己的功能，但除此之外模

块之间最好没有交互，这是 SoC 原则的理想目标。

根据图 3-1 的软件结构图，断点续传软件的功能模块应该包含以下几块：用户交互模块，传输设置模块，异常处理模块，线程管理模块，分片传输模块。

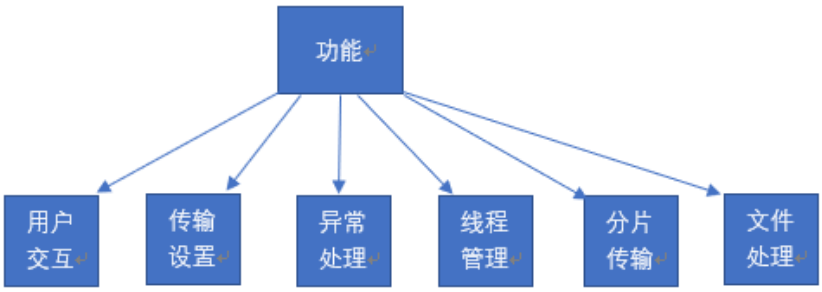

图 3-4 软件功能图

3.2.1 用户交互模块

如图 3-4 的软件功能图所示，展现的是断点续传软件的主要功能模块。其中，对于用户交互模块，需要实现的内容包括图形界面的设计，需要实现符合用户要求的图形界面接口，其中应包括主界面的设置，上传按钮的实现及对应的按钮监听的实现，下载按钮的实现以及对应的监听。对于文件上传部分，用户交互的要求还需要实现本地文件选择的界面窗口当选择好要上传的文件后，还需实现文件传输的实时进度条的图形界面。

对于文件下载部分，与文件上传略有区别，首先需要实现从服务器上选择已有完整文件的图形界面，在选择好待下载文件后同样需要显示文件实时传输进度的图形界面进度条。

3.2.2 传输设置模块

由于此断点续传软件需要遵守 tus 协议的基本传输要求，因此在软件功能中应该包含一块专门负责符合 tus 协议内容的文件传输设置的功能模块。此模块应该设置包含 tus 请求的基本格式，例如”HEAD”,”PATCH”,”OPTION” 这样的请求字段，例如：如果要实现文件的新建上传功能，则应设置请求为”POST”；如果要完成文件恢复上传的功能，则应设置请求为”HEAD”，下载时与上传类似。除了设置 tus 请求，还要在与服务器的交互中附有 tus 协议标准的请求头部信息，例如：在文件的上传过程中，发送给服务器的请求里应包含 Upload-Offset 的字段以及对应的值，包含 Upload-Metadata 的字段以及对应值，Tus-Resumable 的字段以及对应的值，以及客户端与服务端连接的 timeout 值设置。

3.2.3 异常处理模块

由于此软件设计使用 JAVA 语言实现，在此断点续传软件的运行过程中，以及代码的实现中，会出现不同种类的异常，例如：在操作文件时会出现 IOException，NullPointException 等异常，实现异常处理模块时，应针对可能出现某个特定异常的代码段中，利用 JAVA 语言的异常处理机制，添加对某个特定异常的捕获以及捕获异常后的程序段处理，这样使程序不会以为异常的出现，以及没有代码的处理，最终被 JAVA 虚拟机不断抛向上层函数并出现终端的报错信息。

另外，对于可能出现的不符合 tus 协议标准的请求字段以及不在正常范围内的返回代码，也可以通过自定义异常的方式进行设置，不仅方便了对代码的管理，也对异常处理有了更标准的完善；针对软件的断点续传功能，本软件还设置了恢复选项异常，这个异常的意义在于：为文件的传输提供了可恢复选项，例如被定义为不可恢复传输的文件如果试图使用断点续传功能，则要求抛出这个恢复选项异常。

3.2.4 线程管理模块

此断点续传软件应为用户提供多线程的运行功能，如软件结构图 3-1 所示，用户通过图形界面来选择创建上传文件或下载文件的进程，每个被用户选择创建的新线程应拥有自己的线程 id，以方便程序运行中报错及对线程操作的相关处理。由于断点续传软件所创建的线程的目的都是进行文件的传输，因此有很大的相似性，可以统一通过 JAVA 语言中的线程工厂对需要新创建的线程进行初始化，来减少代码的重复性，增加可读性。软件的主进程应拥有对各个子线程的控制权，包括线程的暂停，线程的恢复等。

3.2.5 分片传输模块

分片传输的功能模块是此断点续传软件的核心模块，他实现了文件的断点续传共能。如图 3-3 所示的分片传输原理图，通过对文件的分片来实现文件传输位置 offset 的等间隔记载，在恢复文件传输的工程中再使用 offset 的值，找到目标文件以及本地文件的传输位置，实现文件的可恢复传输功能。

此外，文件分片的大小应有科学的设置：如果文件分片过大，则对于较小的文件，尤其是小于文件分片大小的文件，则失去了可恢复性传输的功能；如果文件分片过小，则程序运行时将在每个小文件分片传输完成后运行记录 offset 值得代码片段，而这个负责记录 offset 值得代码涉及到文件的写操作，因此，过大频率的写文件操作将给程序的运行带来负担，影响程序运行速度。所以，文件分片大小的设置应以符合用户的日常需求为准。

3.2.6 文件处理模块

此断点续传软件的功能实现过程中还将涉及到文件处理的内容，例如：在文件上传过程中，文件处理环节体现在本地文件的选取，并从客户端软件中提取此文件对上传所需要的信息，像文件长度、文件在本地系统中的绝对路径、文件名称等信息、以及若是恢复上传过程，在从服务器获取到文件上传位置 offset 的值之后，将在客户端程序中的文件对象上找到 offset 的位置，从这个暂停位置作为新一次上传的起始位置，获取到输入流当中，同时上传过程对服务器的文件也进行着操作，它体现在需要从客户端与服务端的连接中获取服务端目标文件的输出流，来进行新一次的文件写操作。由于 tus 协议本身在服务端并不提供服务端已有文件的查看接口，因此在完成一个文件的完整上传时，还要在本地做记录。

在文件下载操作中，文件处理环节体现在本地文件的创建、从连接中获取服务端目标文件输入流的操作、从本地新建或已存在文件中提取输出流、在异常中断发生时或用户手动暂停下载时应有在本地记录文件传输位置 offset 值的处理。

3.3 主要类的设计

本传输软件的类设计符合软件需求的基本要求，主要包括显示客户端图形界面的界面类，负责文件传输及 offset 值记录的文件传输类，负责管理文件传输参数的客户端类等。

3.3.1 TusClient 类的设计

TusClient 类主要负责通过接收的用户请求，在这里进行处理，完成 uploader 类 doanloader 类的初始化，为文件的传输功能做好准备。其中 prepareConnection 函数设置好了符合 tus 协议要求的请求格式，然后在创建上传函数, 创建下载函数, 回复上传函数, 恢复函数对应不同的要求再单独添加的服务器请求条件。这样的设计减少了代码的重复，简化了结构，提高了可读性。

其中的成员变量 TUSVERSION 记录了现在 tus 协议需要添加在请求中的 tus 版本号，因为在每次本地与服务器的交互中，请求头中都要添加这个内容，所以将他定义成了客户类中的一个成员变量，因为客户端程序中的从线程的打开只新建一个客户端类，并在这个客户端类的实例对象中进行对文件的各种操作，所以其他的文件传输设置都可以似乎用这个在外部的变量。uploadCreationURL 变量记录了服务器 ip 地址及端口，resumingEnabled 这个 bool 型变量记录了这个客户类及对应的任务线程是否允许开启恢复性传输。TusURLStore 变量用来记录这个线程对应的本地文件与服务器文件的键值对，这个变量值存于客户端 PC 内存中。

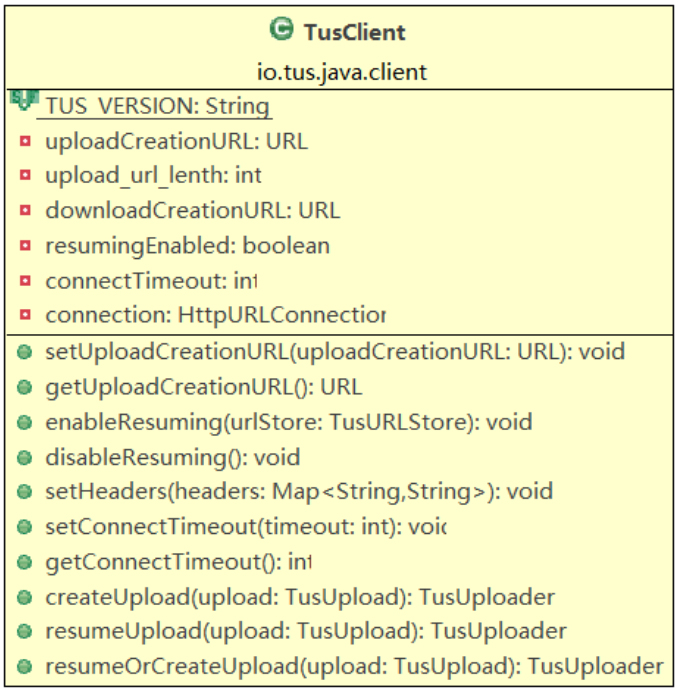

图 3-5 TusClient 类图

客户类的成员函数中包括对可恢复属性的开启、关闭以及获取属性的操作。resumeOrCreateUpload 函数通过判断服务器是否有此本地文件对应的未完成上传的部分文件，来确定此次文件上传是新建上传还是恢复上传，后面的判断恢复或新建下载与此类似。与这个函数对应的两个函数分别时 createUpload 函数和 resumeUpload 函数，这两个函数分别完成对文件新建上传和文件恢复上传的信息初始化设置，并返回用于文件分片传输功能的 uploader 实例。setConnectionTimeout 函数用于在此客户类实例外部设置连接超时时间，并且还包含获取超时时间接口。

3.3.2 Window 类的设计

Window 类的功能是负责用户图形界面的显示（如图 3-6，3-7），其中 Window 初始化函数中设置了 java 中 JFrame 的窗口位置，窗口大小及相关字体的设置。MyListener 类定义在 Window 类中，他的主要功能为监听 Window 窗口中的上传及下载按钮，并在这个监听类的功能函数中完成新建线程的处理。

3.3.3 TusUpload 类的设计

TusUpload 类用于封装选择的要上传的文件，并封装与文件有关的信息。在这个类中，size 变量用于记录此待上传文件的大小，input 变量由于设置文件的本地输入流。File 类型的 file 变量用来记录这个文件的 JAVA 类对象，可以通过这个变

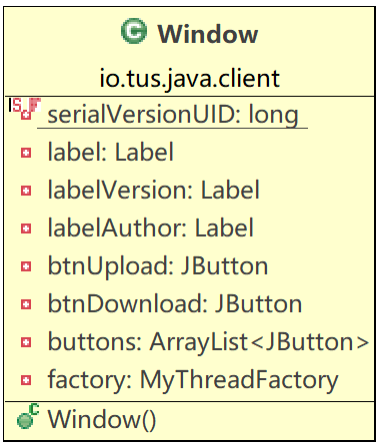

图 3-6 Window 类图

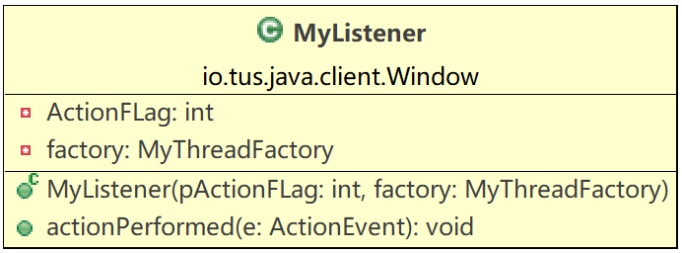

图 3-7 WindowListener 类图

量来获取待上传文件的文件名、文件大小、绝对路径等信息。metadata 这个 Map 型的变量中，则记录了该文件的文件名，并在之后的连接服务器操作中附加在请求头部中。

在成员函数中，带有 File 参数的 TusUpload 初始化函数通过待上传文件对象来初始化成员变量中的各个值，包括 size，input，file，metadata，fingerprint 等变量。其他函数均为设置或获取 upload 对象中封装的待上传文件信息的操作，因为这个类的设计思路就是在线程运行中初步封装待上传的文件信息，然后通过客户类中的功能函数来最终确定文件分片传输所需要的 uploader 对象，再通过调用 uploader 中的函数来完成分片上传的软件功能。

3.3.4 JProcessBar 类的设计

JProcessBar 类主要负责将实时的 offset 值用进度条的图形方式显示出来。进度条窗口应对应着一个已创建的线程，为这个线程所运行的文件上传或文件下载过程提供实时的进度条图形界面显示。此外，这个进度条类还应包含着用于控制这个线程文件传输过程的暂停传输和恢复传输按钮。

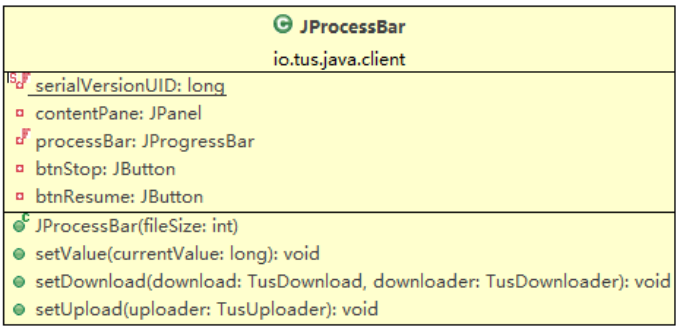

图 3-8 JProcessBar 类图

在此类的成员变量中，应包含用于显示此进度条的 JAVA 窗口容器 JPanel 对象。btnStop 变量和 btnResumme 变量均为 JButton 类型的对象，用于表示并实现暂停文件传输和恢复文件传输的按钮，在这个进度条类中还分别对应暂停传输和恢复传输按钮编写监听类，并在对应的监听类中实现当前线程的结束以及建立新的线程以实现恢复文件传输的操作。

在此类的成员函数中，带有整型变量 fileSize 的参数的初始化函数通过传入的目标文件的大小来确定构成进度条中的总数据大小。而于此对应，需要 override 的函数 setValue 将在每次分块传输完成时将此时此刻的文件传输进度 offset 的值传入并调用此函数，以此来实现进度条对文件传输位置的实时图形显示。

3.3.5 TusUploadRun 类的设计

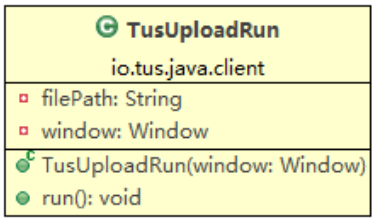

图 3-9 TusUploadRun 类图

这个类负责为线程提供用于本软件文件传输功能的接口，TusUploadRun 类与 TusDownloadRun 类基本相似。

在这个类中的功能函数 TusUploadRun 中，将完成一个新建线程中从 TusClient 类的初始化，TusUpload 类的对文件的封装过程，进度条类的初始化，到调用客户类的功能函数，最终生成 uploader 对象或 downloader 对象的分片传输所需的内容。在生成了传输容器对象后，还将通过循环和对容器对象分片传输函数的调用来实现文件的分片传输功能并在每次分片传输完成时进行配置文件的写操作和进度条

的更新操作。这个类功能函数的实现将在第四章中呈现。

3.3.6 主要类间的关系

如图 3-10，展示了 Window 类，TusUploadRun 类间的调用关系。window 中的监听按钮收到信号时，将新建上传线程，而 TusUploadRun 则是线程运行的接口。

在 Window 类所创建的断点续传软件主窗口中，分别包含着下载和上传按钮，用户同过这两个按钮的点击来出发监听类的运行，监听类通过判断用户操作来通过 MyThreadFactory 线程工厂类创建新的线程，同时确定用户所需的上传或下载操作。如果是上传操作，则对应将 TusUploadRun 接口传入到线程工厂中，从而在这个新建线程里运行的就是接口里定义的文件上传的一系列操作，正如上一节所描述的 TusUploadRun 类的介绍那样；如果是下载操作，同理，监听类将把文件下载运行接口传入线程工厂中，从而运行接口里定义的文件下载的一系列操作。在图 3-10 中由于空间大小，没有画出 TusDownRun 类与他们间的关系，但此类应与 TusUploadRun 类的关系相似。

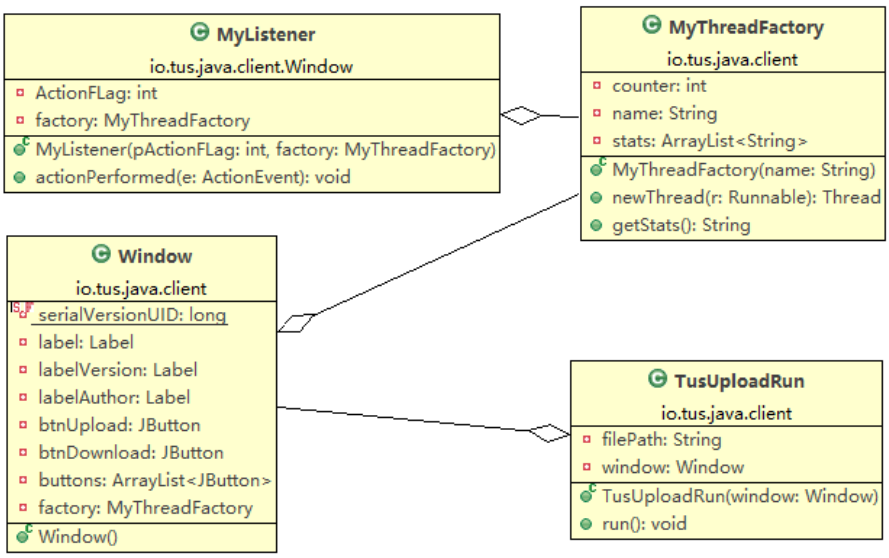

图 3-10 Window 类，TusUploadRun 类的关系

如图 3-11 所示，展示了 TusClient 类与 TusUploader 类、TusDownloader 类的关系。在 TusClient 类中，通过对象的成员函数例如开始上传函数、恢复上传函数、开始下载函数、恢复下载函数。在上传过程中，程序将通过服务器上是否有相同文件来决定是否新建上传或恢复上传，并返回一个 TusUploader 类型的对象，在这个对象中包含着上传此文件所需的信息；在下载过程中，程序将通过本地是否有与选择文件相同的文件来决定是否新建下载或恢复下载，并返回一个 Tus 下载

容器类型的对象，这个对象的私有成员变量中包含着下载此文件所需的所有信息。所以传输文件所需的内容，包含服务器 url、目标文件长度、文件传输进度 offset 的值、文件存储位置等，这些都分别包含在了两个上传、下载容器中，这两个容器同时拥有着对文件传输起着至关重要作用的分片传输功能函数，而客户端类的作用就是对这两个容器类进行配置和初始化，以此来完成对传输的控制。

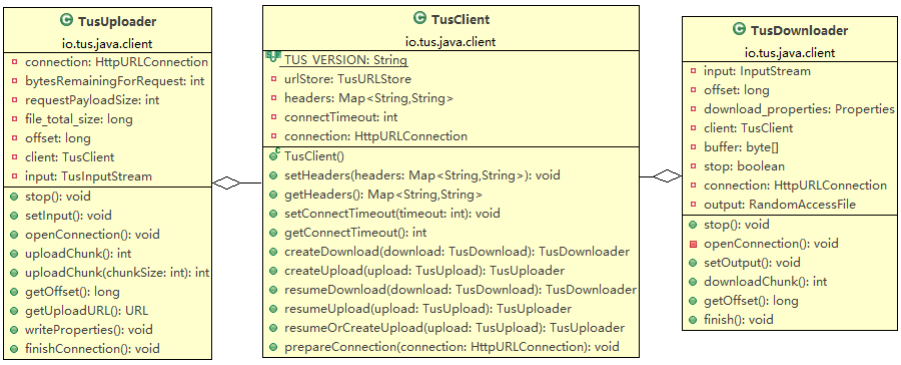

图 3-11 客户类和上传下载类的联系

3.4 软件流程设计

此软件为用户创建了图形界面，在界面中可自行选择上传文件或下载文件的操作。

当使用上传功能时，程序会通过 java 提供的文件选择工具选择上传文件，并通过与服务器的连接并获取此文件在服务器上是否有部分或完整的镜像，若只有部分上传则会通过服务器返回的 response 回复提供给此客户端程序上传的进度信息，即 offset 字段。

当使用下载功能时，程序会通过本地记录的配置文件获取此时服务器上已有的完整文件名，以供下载选择，此时程序会提供用户选择本地文件来开始或继续下载此文件，程序通过配置文件获取本地未完成下载文件的 offset 值，并继续下载。

3.4.1 UML 流程图

统一建模语言（英语：Unified Modeling Language，缩写 UML）是非专利的第三代建模和规约语言。UML 是一种开放的方法，用于说明、可视化、构建和编写一个正在开发的、面向对象的、软件密集系统的制品的开放方法。UML 展现了一系列最佳工程实践，这些最佳实践在对大规模，复杂系统进行建模方面，特别是在软件架构层次已经被验证有效。

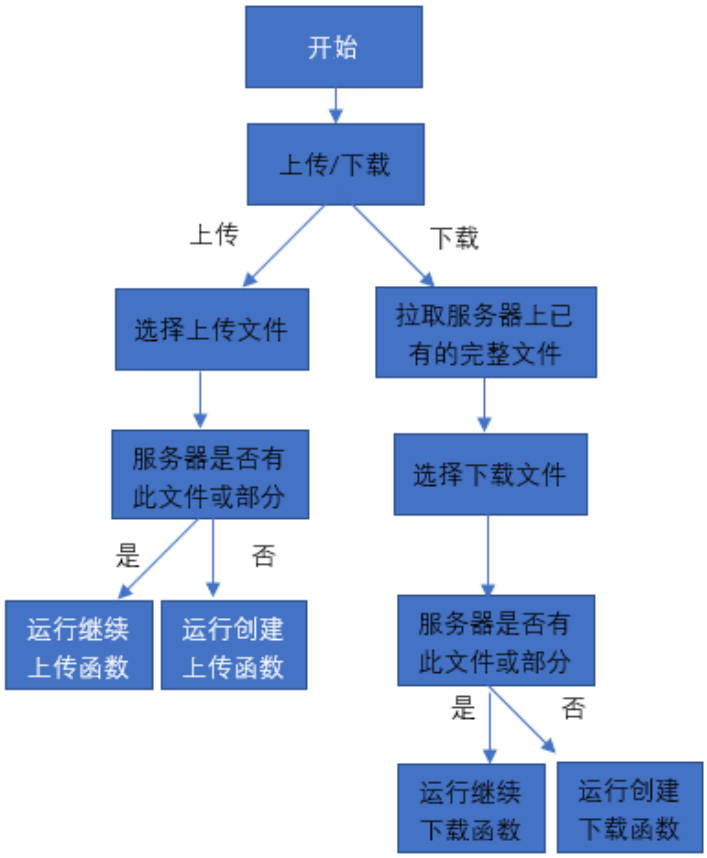

图 3-12 断点续传软件流程图

如图 3-12 所示的断点续传软件流程图，程序开始运行，将首先显示出 Window 类展现的主界面，上面要包含着上传文件和下载文件两个按钮。通过这两个按钮的监听类，程序将确定用户的选择，并进行分别的处理。就如图 3-10 所示的类图关系中的调用一样：如果用户选择的是上传操作，程序在运行 TusUploadRun 接口定义的线程时，就等于运行了如图 3-12 左边上传流程，其中服务器是否有此文件或部分的内容，包含在了初始化 TusClient 类以及调用这个客户类中的判别函数来实现，确定了这个之后，就是通过调用两个传输容器 uploader 以及 downloader 类中的分片传输函数来实现文件的分片传输过程。

如果用户选择的是下载操作，则程序将运行图 3-12 右边下载部分的流程。程序首先将为用户用图形界面的方式提供服务器上已存在的完整文件以供用户选择，选择好待下载文件后，程序同样初始化 TusClient 类，并调用判别创建下载或恢复下载函数来确定接下来需要进行的操作，同时将文件信息封装在 TusDownloader 类的实例中，后面的代码与上传函数一样，实现文件的分片传输。以上流程和程序代码的实现都包含于 TusDownloadRun 接口的运行函数中，它定义了这个新建

线程的操作。

3.5 本章小结

本章主要介绍了断点续传软件的需求分析，运行原理，和 java 代码的分类设计，分别对主要类进行了关键函数说明。可以看到，软件在遵循 tus 协议文件传输规则的同时，通过分片传输机制实现了断点续传功能，这也是此次毕业设计的核心任务。

第四章 软件的实现

4.1 程序主要类的介绍

此程序的主要文件传输功能函数封装在了 tusuploader 和 tusdownloader 两个类中，其他类完成其他基本参数传递，配置文件以及图形界面的需求。

4.1.1 Window 类

客户端界面的基本设置：

主界面的设置

```
1         public Window() {
2                 super("tus断点续传客户端");
3                 this.setBounds(0, 0, 500, 260);
4                 this.setVisible(true);
5                 this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
6                 this.getContentPane().setLayout(null);
7                 this.setLocationRelativeTo(null);
8                 buttons.add(btnUpload);
9                 buttons.add(btnDownload);
10                label.setBounds(100, 10, 260, 30);
11                label.setText("tus java client");
12                label.setFont(new java.awt.Font("MS Tang", Font.BOLD, 20));
13                this.getContentPane().add(label);
14                labelVersion.setBounds(360, 18, 50, 20);
15                labelVersion.setText("v1.0");
16                labelVersion.setFont(new java.awt.Font("MS Tang", Font.BOLD,
                      15));
17                this.getContentPane().add(labelVersion);
18                labelAuthor.setBounds(350, 30, 150, 30);
19                labelAuthor.setText("2018.4     by Tzz");
20                labelAuthor.setFont(new java.awt.Font("MS Tang", Font.ITALIC,
                      10));
21                this.getContentPane().add(labelAuthor);
22                btnUpload.setBounds(50, 60, 180, 30);
23                this.getContentPane().add(btnUpload);
24                btnDownload.setBounds(50, 100, 180, 30);
25                this.getContentPane().add(btnDownload);
26                //设置监听
27                for (int i = 0; i < buttons.size(); i++)
28                {
29                        buttons.get(i).addActionListener(new MyListener(i,
```

```
                factory));
30          }
31      }
```

Window 类继承了 JFrame 类，这个公共的 Window 类构造函数完成了基本的界面设置，8-9 行将上传和下载按钮添加到了 ArrayList<JButton> buttons 中，便于管理，并在 23，25 行将两个按钮添加到了 window 容器中。用 JButton 类中的 addActionListener 函数为两个按钮添加监听。监听类如下所示：

监听的编写

```
1  private class MyListener implements ActionListener {
2          private int ActionFLag = -1;
3          private MyThreadFactory factory = new MyThreadFactory("
               MyThreadFactory");
4          public MyListener(int pActionFLag, MyThreadFactory factory)
5          {
6                  this.ActionFLag = pActionFLag;
7                  this.factory = factory;
8          }
9
10         @Override
11         public void actionPerformed(ActionEvent e) {  //上传
12                 switch (this.ActionFLag) {
13                 case 0:  //上传
14                         TusUploadRun run1 = new TusUploadRun(Window.
                               this);
15                         Thread thread1;
16                         thread1 = factory.newThread(run1);
17                         thread1.start();
18                         break;
19
20                 case 1:  //下载
21                         TusDownloadRun run2 = new TusDownloadRun();
22                         Thread thread2;
23                         thread2 = factory.newThread(run2);
24                         thread2.start();
25                         break;
26
27                 default:
28                         break;
29                 }
30         }
31     }
```

通过 MyListener 类中的整形变量 ActionFlag 来确定应响应上传操作还是下载操作，并在前面的 Window 类构造函数中进行对应的初始化。上传和下载模块分别初始化了 TusUploadRun 和 TusDownloadRun 两个继承了 Runable 类的对象，并通过线程工厂分别创建了负责文件传输的线程。

4.1.2 MyThreadFactory 类

MyTreadFactory 构造函数：

MyThreadFactory 的构造函数

```
public MyThreadFactory(String name) {
                counter = 0;
                this.name = name;
                stats = new ArrayList<String>();
        }
```

构造函数中初始化了计数器 counter，用于记录线程工厂创建的线程数量，工厂类名字，和用于记录每个创建出的线程状态的 ArrayList。

MyThreadFactory 的创建新线程函数

```
@Override
public Thread newThread(Runnable r) {
        Thread t = new Thread(r, name + "-Thread_" + counter);
        counter++;
        stats.add(String.format("created thread %d with name %s on %s\n", t.
            getId(), t.getName(), new Date()));
        return t;
        }
```

这个 Override 函数为线程工厂中最重要的功能函数，他接受一个 Runnable 对象参数，并在函数中通过传递 Runnable 接口的方式创建新线程，此线程的 start（）方法运行的便是 Runnable 接口中定义的过程。这个线程工厂基本完成了软件设计阶段所需要的需求。

4.1.3 TusUploadRun 类

TusUploadRun 类继承了 java 提供的 Runnable 类，这个类可以通过接口传给新建线程，使线程完成此类里完成的功能。

TusUploadRun 类中的线程运行函数

```
@Override
public void run() {
```

```
3         JFileChooser chooser = new JFileChooser();
4         chooser.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
5         int r = chooser.showOpenDialog(window);
6         if (r == JFileChooser.APPROVE_OPTION) {
7                 // 设置文件路径
8                 filePath = chooser.getSelectedFile().getPath();
9         }
10        try {
11                System.setProperty("http.strictPostRedirect", "true");
12                final TusClient client = new TusClient();
13                client.setUploadCreationURL(new URL("http://localhost:1081/
                      files"));
14                File file = new File(filePath);
15                final TusUpload upload = new TusUpload(file);
16                TusURLMemoryStore tus_url_store = new TusURLMemoryStore();
17                client.enableResuming(tus_url_store);
18                System.out.println("Starting upload...");
19                TusUploader uploader = client.resumeOrCreateUpload(upload);
20                uploader.setChunkSize(1024);
21                // 设置进度条
22                 JProcessBar process_bar = new JProcessBar((int)upload.getSize
                      ());
23                 process_bar.setUpload(uploader);
24                do {
25                        process_bar.setValue(uploader.getOffset());
26                } while (uploader.uploadChunk() > -1);
27                uploader.finishConnection();;
28                if (uploader.getOffset() == uploader.getFileTotalSize()) {
29                        System.out.println("Upload finished.");
30                        System.out.format("Upload available at: %s", uploader.
                              getUploadURL().toString());
31                        uploader.writeProperties();
32                }
33        } catch (Exception e1) {
34                e1.printStackTrace();
35        }
36}
```

此类中包含的 run 函数即为线程运行的代码，其中包括弹出文件选择窗口，将选中文件传递给设置函数。其中 run 函数的第 13 行规定了服务端的 ip 地址和端口及文件存储路径，由于没有进行过远程测试，因此这个服务器地址没有给出用户接口，而仅仅在这里进行了规定。第 19 行调用的函数为 TusClient 类中的关键函数，此函数用来运行恢复上传或新建上传的准备工作，包括恢复上传中的从服务

器获取 offset 值以及新建上传中写配置文件等。client 对象调用的这个函数返回了一个 TusUploader 类的对象，这个对象用来在下面的循环按块上传中调用传输函数。第 22 行中的 JProcessBar 类为自定义类，这个类创建了一个新的窗口，用来实时显示文件传输的进度，并在第 25 行这个 while 循环中每次更新进度条的显示值。第 26 行为循环上传的判断条件，同时也是进行数据传输的函数。

由于在新弹出的任务窗口中提供了以中断与服务器连接的方式中断传输，并因此调出这个 while 循环，所以要在第 28 对是否完成传输进行判断。

4.1.4 TusUpload 类

TusUpload 类主要完成文件上传的输入流设置，以及 base64 的文件名加密操作。

TusUpload 类构造函数

```
1  public TusUpload(@NotNull File file) throws FileNotFoundException {
2          size = file.length();
3          this.file = file;
4          setInputStream(new FileInputStream(file));
5          fingerprint = String.format("%s-%d", file.getAbsolutePath(), size);
6          metadata = new HashMap<String, String>();
7          metadata.put("filename", file.getName());
8          local_file_path = file.getAbsolutePath();
9      }
```

在这个构造函数中，主要初始化了输入流，这里的输入流为自定义的 TusInputStream 类，这个类对普通的文件输入流进行了封装和简单的添加功能。

base64 加密函数

```
1  static String base64Encode(byte[] in)       {
2          StringBuilder out = new StringBuilder((in.length * 4) / 3);
3          String codes = "
               ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=";
4          int b;
5          for (int i = 0; i < in.length; i += 3)  {
6              b = (in[i] & 0xFC) >> 2;
7              out.append(codes.charAt(b));
8              b = (in[i] & 0x03) << 4;
9              if (i + 1 < in.length)      {
10                 b |= (in[i + 1] & 0xF0) >> 4;
11                 out.append(codes.charAt(b));
12                 b = (in[i + 1] & 0x0F) << 2;
13                 if (i + 2 < in.length)  {
```

```
                    b |= (in[i + 2] & 0xC0) >> 6;
                    out.append(codes.charAt(b));
                    b = in[i + 2] & 0x3F;
                    out.append(codes.charAt(b));
                } else  {
                    out.append(codes.charAt(b));
                    out.append('=');
                }
            } else      {
                out.append(codes.charAt(b));
                out.append("==");
            }
        }
        return out.toString();
    }
```

虽然 jdk 中 java.util.Base64 已经包含类 base64 的加密方法，但这里还是用 java 代码简单实现了 base64 的功能。

4.1.5 TusUploader 类

Uploader 类中封装了上传文件所需要的所有信息，只要传递这个类，就可以灵活的其他位置进行传输。其中最重要的成员函数为 uploadChunk () 函数。

上传函数

```
public int uploadChunk() throws IOException, ProtocolException {
        openConnection();
        if (stop) {
                stop = false;
                return -1;
        }
        int bytesToRead = 1024;
        int bytesRead = input.read(buffer, bytesToRead);
        if(bytesRead == -1) {
                return -1;
        }
        output.write(buffer, 0, bytesRead);
        output.flush();
        offset += bytesRead;
        bytesRemainingForRequest -= bytesRead;
        if (input == null) return -1;
        return bytesRead;
    }
```

当 boolean 型变量 stop 的值为 true 时，将退出这个块传输循环。传输过程为，在输入流中读取设置的块大小 1KB 的数据，读入 buffer 缓冲区中，然后再从 buffer 传入输出流中。

从连接中获取输出流

```
output = connection.getOutputStream();
```

这里，上传的输出流为从与服务器的连接类中获取的。下载过程中的 TusDownload 类和 TusDownloader 类与上传类似，最大的区别在于下载的输入流来自与服务器连接中，而输出流为本地的用户创建的文件。

tus 协议中的新建上传 POST

```
HttpURLConnection connection = (HttpURLConnection) uploadCreationURL.
    openConnection();
connection.setRequestMethod("POST");
```

上传文件的请求也遵循了 tus 协议的内容，设置了 POST 方法。而下载时则是 GET。

4.1.6 关于配置文件

程序一共管理了三个配置文件：

1.config.properties 文件：记录了本地文件绝对路径与服务器对应传输文件的键值对，便于在恢复上传时查询此文件，找到服务器对应的未完成文件。

2.serverfile.properties 文件：每当程序检查完成一个文件的完整上传时，会记录该文件在服务器上的绝对路径。

记录服务器中存在的完整文件

```
Properties properties = new Properties();
properties.load(new FileInputStream("F:\\Workspace_for_jee\\tus-java-client-
    master\\server_file.properties"));
properties.setProperty(uploadURL.toString(), "ture");
properties.store(new FileOutputStream("F:\\Workspace_for_jee\\tus-java-client-
    master\\server_file.properties"), "上传完成的文件");
```

downloadoffset.properties 文件：用于在下载文件时在本地保存文件下载位置 offset 的值，文件中保存的是本地保存文件的绝对路径和此文件 offset 值的键值对。因为在程序运行时，若每次块下载循环完成就写一次 offset 值，这样的方式严重影响了下载速度，所以设计为每下载 10MB 的文件大小时再记录一次 offset，如果通过展厅按钮方式暂停下载时就即刻记录 offset。

4.2 本章小结

本章节详细介绍了基于 tus 协议的断点续传客户端软件的核心类的功能及代码实现，可以看出代码的编写基本符合了软件需求的设定，完成了 tus 协议规定的报文标准、通过分块传输以实现断点续传功能、以及多线程管理的实现。

第五章 软件的测试

5.1 软件测试的概念

软件测试（英语：software testing），描述一种用来促进鉴定软件的正确性、完整性、安全性和品质的过程。据此，您可能会想，软件测试永远不可能完整的确立任意电脑软件的正确性。然而，在可计算理论（计算机科学的一个支派）一个简单的数学证明推断出下列结果：不可能完全解决所谓“死机”，指任意计算机程序是否会进入死循环，或者罢工并产生输出问题。换句话说，软件测试是一种实际输出与预期输出间的审核或者比较过程。

软件测试的经典定义是：在规定的条件下对程序进行操作，以发现程序错误，衡量软件品质，并对其是否能满足设计要求进行评估的过程。

5.2 本软件的测试

5.2.1 软件测试用例

测试用例（Test Case）是为某个特殊目标而编制的一组测试输入、执行条件以及预期结果，以便测试某个程序路径或核实是否满足某个特定需求。

5.2.2 软件测试环境

本软件的测试环境为 windows10，64 位系统，IDE 使用的是 eclipse java mars，测试工具为 JUnit，JUnit 是一个 Java 语言的单元测试框架。它由肯特 • 贝克和埃里希 • 伽玛（Erich Gamma）建立，逐渐成为源于 Kent Beck 的 sUnit 的 xUnit 家族中为最成功的一个。JUnit 有它自己的 JUnit 扩展生态圈。

5.2.3 软件的详细测试

此断点续传软件中，对一些重点类进行了测试函数的编写，内容如下。

5.2.3.1 TestTusClient 测试类

testTusClientURL 测试函数

```
1    @Test
2    public void testTusClientURL() throws MalformedURLException {
3        TusClient client = new TusClient();
4        client.setUploadCreationURL(creationUrl);
5        assertEquals(client.getUploadCreationURL(), creationUrl);
```

```
6     }
```

testTusClientURL 函数用于检查 client 对象中的上传地址是否正确。

testSetUploadCreationURL 测试函数

```
1     @Test
2     public void testSetUploadCreationURL() throws MalformedURLException {
3         TusClient client = new TusClient();
4         client.setUploadCreationURL(new URL("http://master.tus.io"));
5         assertEquals(client.getUploadCreationURL(), new URL("http://master.tus.
              io"));
6     }
```

testSetUploadCreationURL 函数用于测试 TusClient 类中的 setUploadCreationURL 函数的可用性。

testEnableResuming 测试函数

```
1     @Test
2     public void testEnableResuming() {
3         TusClient client = new TusClient();
4         assertEquals(client.resumingEnabled(), false);
5
6         TusURLStore store = new TusURLMemoryStore();
7         client.enableResuming(store);
8         assertEquals(client.resumingEnabled(), true);
9
10        client.disableResuming();
11        assertEquals(client.resumingEnabled(), false);
12    }
```

testEnableResuming 函数用于测试 TusClient 类中的 EnableResuming 函数。当初始化 TusClient 类的对象时，对象中的可恢复属性初试为 false，当用 TusURLStore 类进行服务器文件地址 URL 存储时，将自动将 client 对象中的可恢复上传属性设置为 true。当调用 disableResuming 函数时，可恢复属性又变回 false。

testPrepareConnection 测试函数

```
1     @Test
2     public void testPrepareConnection() throws IOException {
3         HttpURLConnection connection = (HttpURLConnection) mockServerURL.
              openConnection();
4         TusClient client = new TusClient();
5         client.prepareConnection(connection);
6
```

```
7         assertEquals(connection.getRequestProperty("Tus-Resumable"), TusClient.
              TUS_VERSION);
8     }
```

testPrepareConnection 函数用于测试 TusClient 类中的 tPrepareConnection 函数是否可用。此测试使用 mockServerURL 地址打开连接，初始化客户端类后调用 prepareConnection 函数，并对照从连接中获取的 tus 版本号与本地新建的客户端类 tus 版本号是否一致。

testSetHeaders 测试函数

```
1     @Test
2     public void testSetHeaders() throws IOException {
3         HttpURLConnection connection = (HttpURLConnection) mockServerURL.
              openConnection();
4         TusClient client = new TusClient();
5
6         Map<String, String> headers = new HashMap<String, String>();
7         headers.put("Greeting", "Hello");
8         headers.put("Important", "yes");
9         headers.put("Tus-Resumable", "evil");
10
11        assertNull(client.getHeaders());
12        client.setHeaders(headers);
13        assertEquals(headers, client.getHeaders());
14
15        client.prepareConnection(connection);
16
17        assertEquals(connection.getRequestProperty("Greeting"), "Hello");
18        assertEquals(connection.getRequestProperty("Important"), "yes");
19    }
```

testSetHeaders 函数用于测试 TusClient 类中的 SetHeaders 函数的功能。在代码的 7，8，9 行中分别对 headers 进行内容填充，然后分别从本地初始化的客户端类和建立的与服务器的连接中获取键值对，查看是否与预期值对应。

用 JUnit 运行的 TestTusClient 类测试结果如图 5-1 所示：

5.2.3.2 TestTusUpload 测试类

testTusUploadFile 测试函数

```
1     @Test
2     public void testTusUploadFile() throws IOException {
3         String content = "hello world";
```

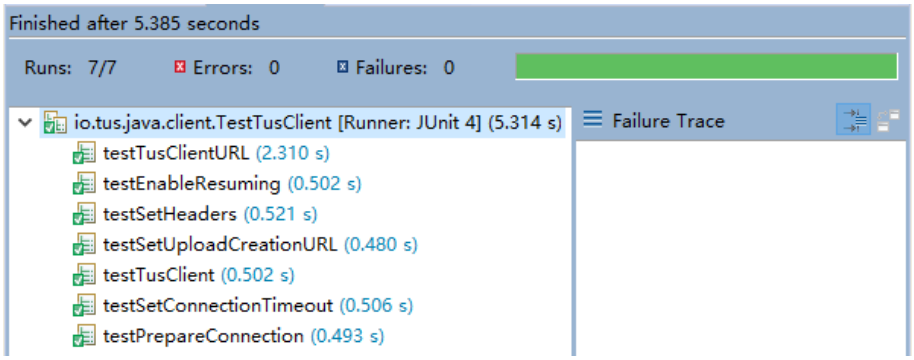

图 5-1 TestTusClient 测试类的运行例子

```
4
5         File file = File.createTempFile("tus-upload-test", ".tmp");
6         OutputStream output = new FileOutputStream(file);
7         output.write(content.getBytes());
8         output.close();
9
10        TusUpload upload = new TusUpload(file);
11
12        Map<String, String> metadata = new LinkedHashMap<String, String>();
13        metadata.put("foo", "hello");
14        metadata.put("bar", "world");
15        metadata.putAll(upload.getMetadata());
16
17        assertEquals(metadata.get("filename"), file.getName());
18
19        upload.setMetadata(metadata);
20        assertEquals(upload.getMetadata(), metadata);
21        assertEquals(
22                upload.getEncodedMetadata(),
23                "foo aGVsbG8=,bar d29ybGQ=,filename " + TusUpload.base64Encode(
                      file.getName().getBytes()));
24
25        assertEquals(upload.getSize(), content.length());
26        assertNotSame(upload.getFingerprint(), "");
27        byte[] readContent = new byte[content.length()];
28        assertEquals(upload.getInputStream().read(readContent), content.length
              ());
29        assertEquals(new String(readContent), content);
30        }
```

TestTusUpload 测试类和 testTusUploadFile 测试函数用于测试 upload 类中的管理文件信息的函数的正确性。测试函数首先在本地创建一个用于测试的临时

文件 tus-upload-test.tmp，将 hello world 写如此文件中，然后通过 upload 对象的 getMetadata 函数提取初始化 upload 时设置的信息，与本地新建的文件信息对比。

测试类的运行结果如图 5-2 所示：

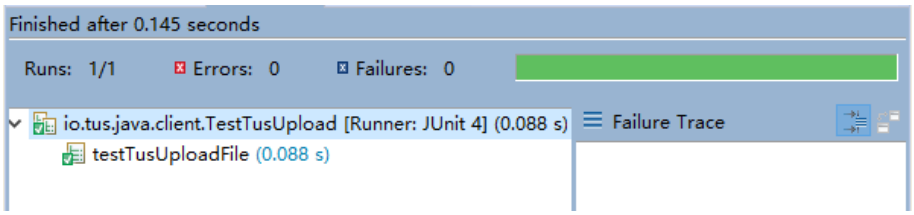

图 5-2 TestTusUpload 测试类的运行例子

5.3 软件的使用

如图 5-3 所示，展示的是客户端软件在选择本地待上传文件时的图形界面。

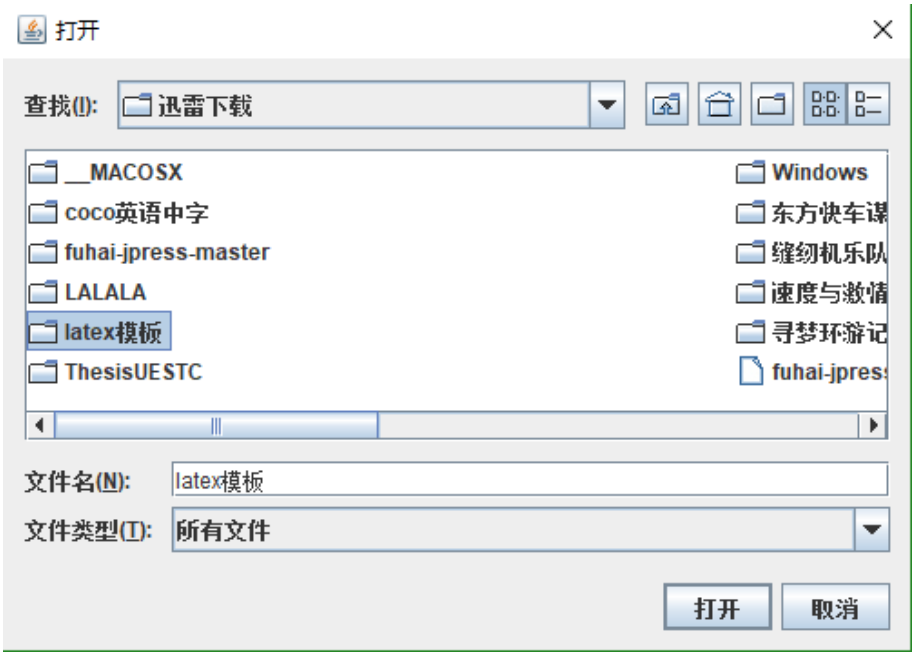

图 5-3 断点续传软件上传功能

如图 5-4 所示，展示的是软件从服务器获取已存在的完整文件的示意图，在这个下拉菜单中将显示所有已有的完整文件，当用户点击确定后将会出现图 5-3 所示类似的创建本地接收文件或选择本地已下载部分的文件，来完成文件的创建下载或恢复下载。

如图 5-5 所示，展示的是断点续传软件多线程完成用户任务的示意图，可以让用户分别操作不同线程控制的文件传输过程，而其他线程的任务运行不受影响。

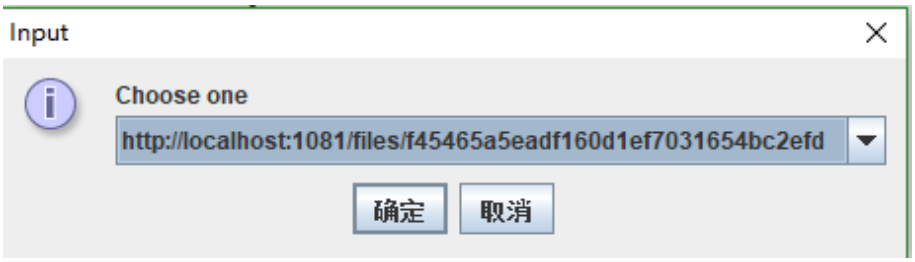

图 5-4 从服务器选择文件

虽然程序没有设定最大允许的活动线程数量，但是由于网络等硬件条件的限制，当多个线程同时进行文件传输时，传输速度将受到较大影响。得出这个结论的依据是当启用多个线程时，文件进度条进度明显变慢，但将其他传输线程暂停支流单个线程时，传输速度将会恢复到单独运行的水平。

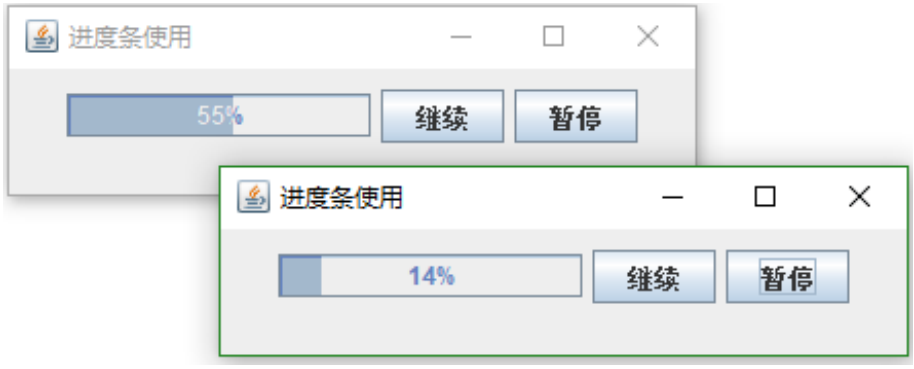

图 5-5 上传和下载并行运行

由于本软件采用的是分片传输原理，因此当特殊故障发生使得程序崩溃或停止运行时，不会影响到文件传输位置 offset 的记录。

5.4 本章小结

本章通过运行 tusd 服务器与本软件进行程序黑箱测试，包括文件的上传、下载以及多线程运行。并通过测试类函数的编写，利用 JUnit 工具完成了部分类功能函数的数据测试，及结果展示。

第六章 全文总结与展望

6.1 全文总结

本文以 tus 协议为基础，介绍了 tus 协议的基本内容，然后对软件的需求分析，模块设计进行了详细说明，最后是软件的运行测试。tus 断点续传软件以 http 协议为基础，遵循 tus 协议的规定标准，并采用分块循环传输的方式来记录文件传输过程中的传输位置，用配置文件的方式记录本地 offset 以完成断点续传的功能。

6.2 后续工作展望

在 tus 协议中，还有很多附加的传输功能，能够使服务器与客户端间更加便捷的通信。按照 tus 协议标准，在服务器端存储的文件名称是软件自行定义的，所以从服务器拉取文件时文件名称不容易被用户接受，可以添加用户自定义文件名称，与 tus 规定的文件名称一一对应。

由于此软件在进行文件传输时的运行传输函数和进度条窗口以及包含的暂停恢复按钮是在同一个线程中定义的，所以不能通过单纯的在此线程外部停止和恢复线程运行，这是本软件多线程管理中不方便的地方，但是可以通过把文件传输函数和文件传输过程的进度条显示分到两个线程的方式来改进，这就可以通过传参的方式实时更新进图条的值，并且可以在运行函数的线程外部轻松的调用 JAVA 线程机制，来暂停和恢复线程的运行。同时使用 java 多线程时可以添加 java 线程池的使用，从而更加方便的对正在运行的线程进行管理。

致 谢

在进行毕业设计期间，首先衷心感谢我的导师丘志杰老师对我的认真督促和帮助。在项目的实现过程中也遇到很多困难，同时也感谢给予我帮助的同学。

参考文献

[1] 汤碧玉. 嵌入式系统中基于 web 的远程监控设计与实现 [D]. 厦门: 厦门大学, 2004,

[2] R. Fielding. Http/1.1[EB/OL]. https://tools.ietf.org/html/rfc7230, June 2014

[3] M. Belshe. Http/2[EB/OL]. https://tools.ietf.org/html/rfc7540, May 2015

[4] F. Geisendörfer. Protocol[EB/OL]. https://tus.io/protocols/resumable-upload.html, 2016-03-25

[5] wikipedia. Base64[EB/OL]. https://zh.wikipedia.org/wiki/Base64, 2018 年 5 月 4 日 (星期五) 17:02

[6] wikipedia. Java[EB/OL]. https://zh.wikipedia.org/zh-hans/Java, 2018 年 5 月 9 日 (星期三) 13:25

[7] wikipedia. Jdk[EB/OL]. https://zh.wikipedia.org/zh-hans/JDK, 2018 年 3 月 24 日 (星期六) 09:53

[8] wikipedia. Jre[EB/OL]. https://zh.wikipedia.org/zh-hans/JRE, 2017 年 11 月 5 日 (星期日) 04:17

[9] oracle. Jdk,jre[EB/OL]. http://www.oracle.com/technetwork/java/javase/downloads/index.html,

[10] oracle. Jdk,jre[EB/OL]. https://docs.oracle.com/javase/10/install/installation-jdk-and-jre-microsoft-windows-platforms.htm#JSJIG-GUID-A7E27B90-A28D-4237-9383-A58B416071CA,

[11] wikipedia. demand analysis[EB/OL]. https://en.wikipedia.org/wiki/Requirements_analysis, 2018 年 5 月 7 日 (星期一) 11:12

外文资料原文

1 A protocol for resumable file uploads

With mobile devices becoming the dominant source of user generated media files, reliable file uploading through unreliable mobile networks has become an important issue for anybody interested in content acquisition.

Reliability here means the ability to detect network errors, and resuming an upload without having to start from the beginning. In many scenarios this can mean the difference between a file reaching your application, or the user giving up in frustration.

Ideally, this should be a trivial feature to add. In reality however, there is quite a lack of solutions in this space. Sure, there are a few JavaScript libraries that claim to support resumable uploading, but in reality you will end up spending a lot of time coming up with your own API for it, or implementing a poorly specified one specific to a library. This is incredibly frustrating, especially if you are planning to support this feature on multiple platforms such as HTML5, iOS and Android.

Now, if you’re a big company like Google, you may sit down and create such a protocol for your needs. And in fact, Google has been working on a such a protocol since 2010, for the now defunct Google Gears. The latest incarnation of this are two incompatible protocols for Google Drive and YouTube. But unfortunately both of these protocols rely on a non-standard http status code (308 Resume Incomplete), and are far from being generic enough for general adoption.

This means that smaller companies are currently doomed to invent, implement and maintain their own incompatible protocols and solutions for something that should be a trivial component of a modern application.

We find this unacceptable, so the tus project is a community project that was born in order to level the playing field and make resumable file uploading easy for anybody to implement.

As time progresses, we share ever larger media files from our phones and desktops. More than often, however, complications arise during this process. Whether it is through servers misbehaving or mobile users switching to a WiFi connection, the outcome is the same: ‘upload interrupted’.

This is by itself a negative user experience, but it becomes even worse when it happens in the middle of a 2GB upload on a slow connection. And of course, the longer an upload takes, the more exposed it is to poor connections. A failed upload will then have to be retried from the start, if the user even bothers with it at all.

With media files growing larger and networks remaining fragile, it is clear that we need a better solution to handle uploading.


A protocol for resumable file uploads

Published on April 15, 2013 by Felix Geisendörfer

tl;dr: We are happy to announce version 0.1 of the tus resumable upload protocol and are interested in your feedback!

Figure 1 blog on tus.io

外文资料译文

1 用于可恢复文件上传的协议

随着移动设备成为用户生成媒体文件的主要来源，通过不可靠的移动网络进行可靠的文件上传已成为任何对内容获取感兴趣的人的重要问题。

这里的可靠性意味着能够检测网络中出现的错误，并且无需从头开始即可恢复上传的特性。在许多情况下，这可能意味着文件到达您的应用程序或用户放弃挫败之间的差异。

理想情况下，这应该是一个微不足道的功能。但事实上，在这个领域还缺乏解决方案。当然，有一些 JavaScript 库声称支持可恢复上传，但事实上，你最终会花费大量时间为自己的 API 提供自己的 API，或者实现一个针对库的错误指定的 API。这非常令人沮丧，特别是如果您打算在多个平台（如 HTML5，iOS 和 Android）上支持此功能。

现在，如果你是像 Google 这样的大公司，你可以坐下来为你的需求创建一个这样的协议。事实上，谷歌自 2010 年以来一直在制定这样的协议，目前已停止使用 Google Gears。这是 Google Drive 和 YouTube 的两个不兼容的协议。但不幸的是，这些协议都依赖于一个非标准的 http 状态代码（308 Resume Incomplete），并且远远不够通用。

这意味着小公司现在注定要发明，实施和维护他们自己的不兼容的协议和解决方案，以应对现代应用中微不足道的组件。

我们发现这是不可接受的，所以 tus 项目是一个社区项目，它的诞生是为了平衡竞争环境，并使任何人都可以轻松实现可恢复的文件上传。

随着时间的推移，我们会从手机和台式机共享更大的媒体文件。然而，在这个过程中往往会出现复杂情况。无论是通过服务器行为异常还是移动用户切换到 WiFi 连接，结果都是一样的：“上传中断”。

这本身就是一种负面的用户体验，但当它发生在缓慢连接的 2GB 上传中间时，它变得更糟。当然，上传时间越长，连接不良就越暴露。如果用户甚至无法使用，上传失败将不得不从一开始就重新尝试。

随着媒体文件越来越大，网络保持脆弱，很显然我们需要更好的解决方案来处理上传。