计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2020,29(3):234−239 [doi: 10.15888/j.cnki.csa.007352]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 Web 的大文件高效上传方法①

阮晓龙[1], 李朋楠[2]

[1](河南中医药大学 信息技术学院, 郑州 450046)
[2](郑州祺石信息技术有限公司, 郑州 450008)
通讯作者: 李朋楠, E-mail: lipengnan@yeework.cn

摘　要: 随着互联网、HTML5 等技术的发展, 文件上传在 Web 端的应用越来越普遍, 同时对文件上传, 特别是大文件上传的效率、稳定性、安全性、普适性等要求也越来越高, 然而目前多数文件上传方式均不能很好的满足这些要求. 因此, 本文提出文件分片和多并发传输方法, 提高大文件传输效率, 从而解决 Web 前端多并发传输控制以及同一文件重复多次上传等问题.

关键词: 断点续传; 文件切片; 并发上传; 文件秒传

引用格式: 阮晓龙,李朋楠.基于 Web 的大文件高效上传方法.计算机系统应用,2020,29(3):234–239. http://www.c-s-a.org.cn/1003-3254/7352.html

High Efficient Upload Method of Large Files Based on Web

RUAN Xiao-Long[1], LI Peng-Nan[2]

[1](School of Information Technology, Henan University of Chinese Medicine, Zhengzhou 450046, China)
[2](QISHI Corporation, Zhengzhou 450008, China)

Abstract: With the development of Internet and HTML5 technology, the application of file upload on the Web is more and more popular. At the same time, the requirement of efficiency, stability, security and universality of file upload, especially large file upload, is higher and higher. However, most of the current file upload methods can not meet these requirements very well. Therefore, this study proposes to use file fragmentation and multi-concurrent transmission method to improve the efficiency of large file transmission, so as to solve the related problems of multi-concurrent transmission control in the front end of the Web and repeated upload of the same file.

Key words: discontinuous upload; file slicing; concurrent upload; file seconds transfer

1　现状分析

1.1　应用价值

大文件上传是 Web 应用系统中常见的问题[1], 尽管 HTTP 是 TCP 上层的协议, 但是 HTTP 协议本身并不适合处理超大的请求体, 文件上传的稳定性存在着很大的问题. 如果传输过程中因某种异常而中断, 将前功尽弃, 同时, 若没有断点续传功能, 那文件只能重新上传, 这样不仅造成带宽资源的浪费, 而且不能保证再次上传文件就能成功[2].

目前基于 HTTP 协议的断点续传方案虽然保证大文件上传过程中的稳定性, 但是在传输效率和安全性方面仍有不足, 满足不了当前互联网 Web 应用系统的需求, 因此基于 Web 的大文件高效上传方法的研究和实现是 Web 应用中的重要基础[3,4].

1.2　常用技术

目前互联网中, 已经存在许多基于 Web 上传文件的方法, 文献[5]中介绍了常用 3 种方法如下所示:

(1) HTML Form

该方法主要利用 HTML 中 Form 表单实现简单的文件上传. 其中表单的提交方式为 POST, 编码类型

① 收稿时间: 2019-08-10; 修改时间: 2019-09-06; 采用时间: 2019-10-21; csa 在线出版时间: 2020-02-28

enctype 为“multipart/form-data”. 同时 Form 表单里还需要包含一个文件框 Input, 文件框类型 (type) 应为 file. 通过结合 Javascript、Ajax 等脚本技术, 从而实现文件异步上传、交互等操作.

(2) RIA 技术

RIA 技术主要是包括 Flex、Silverlight、JavaFX 等技术, 该技术中最常用的文件上传插件为 SWFUpload.

RIA 技术的倡导者为 Adobe, 其提供了 Flex 技术来使程序员可以用编程的方式生成 Flash 内容, 因此一般常用 Flex 技术开发文件上传的客户端程序.

同时, 与 Silverlight、JavaFX 等技术在运行库方面比较, Flex 也最是轻量级的.

(3) 插件技术

在浏览器中也可以使用插件实现文件上传. 虽然可能由于客户浏览器的安全性设置, 导致插件无法运行, 但是在学校内网、企业内网等内部环境还是可以考虑使用的. 插件技术主要包括 Activex、Applet 等. 例如 ActiveX 控件, 该控件利用 Winsock 技术建立与 Web 服务之间的通信, 并读取上传文件数据, 再通过 Socket 技术把数据以 HTTP POST 方式发送给服务器.

此外, 文献[6]中提出了一种文件分块上传的方法, 使用单线程传输, 通过固定大小分片可计算得到上传文件的偏移量, 进而较好的实现了大文件的断点续传. 文献[2,7]中指出了一种双线程传输方法, 即一个线程传输文件内容, 另一个线程记录文件内容的偏移量, 同样可实现断点续传. 这两种方法虽然可实现断点续传, 但是其传输效率还有待提高. 文献[8]中通过多线程方式, 按线程数对文件进行分块, 传输效率虽然有所提高, 但是其每个线程同样是通过记录分块文件的偏移量, 来实现断点续传. 以上方法均为时序传输, 一旦中断再传, 只是从断点位置继续上传, 不能保证已上传文件内容的完整性与正确性.

1.3 存在问题

目前大文件上传常用方法中主要存在以下 4 个方面问题.

(1) 单线程上传

常用的大文件上传方法一般采用单一线程进行文件的传输, 对带宽的利用率较低、稳定性较差, 不能较好实现文件传输过程中的交互操作.

(2) 断点不续传

当使用简单上传功能来进行大文件上传, 如果上传的过程中出现了异常造成中断, 那么此次上传失败, 重试必须从文件起始位置上传.

(3) 安全性不足

常用的大文件上传方法在文件上传前后不做完整性和正确性验证, 或只是简单的进行文件大小方面的完整性验证, 无法保证文件上传后还是原来的文件.

(4) 文件重复上传

当已成功上传的文件再次被上传时, 如不能够有效识别该文件, 会使文件再次进行上传操作, 从而造成带宽、时间资源的浪费.

2 关键技术

本文提出基于 Web 的大文件高效上传方法的研究与实现, 其中包含 7 个关键技术, 具体如下.

2.1 无客户端值守

用户在上传一个几百兆、甚至是上千兆的大文件到服务器上时, 通常采用的是 FTP 协议和某些客户端软件, 从而能较好地支持大文件的上传以及实现文件断点续传功能. 由于 HTTP 协议的超大范围使用和 Web 技术的本身特点, 也涌现了一批 Web 上传大文件的插件[9,10].

本文提出的方法是基于 HTML5 技术可以采用化整为零的文件上传方法, 可将单个大文件转化为多个小分片文件进行上传, 有效地解决了 HTTP 协议本身并不适合处理超大请求文件问题.

2.2 多并发上传

在实际工作中, 单一的 Web 请求往往无法让网络传输速率达到饱和状态[8], 因此采用多并发的方式进行文件上传, 可以充分利于网络带宽、提高文件传输速度[7,11].

本文采用构建线程池的方法, 使并发线程保持在一个合理值, 实现了多并发可控上传, 避免了并发线程过多且无序控制造成的线程排队长时等待现象, 造成资源空耗的浪费.

2.3 文件传输校验

传统方式下, 文件识别校验方法主要是对文件传输前后的 MD5 值进行比对[12,13], 该方法在针对小文件时可行, 但对大文件或超大文件来说, 采用该方法在计算 MD5 值时就会非常耗费时间, 且有可能造成服务器计算处理失败而崩溃.

针对此问题, 本文提出同样可采用化整为零的

MD5 计算方法, 将单个大文件的 MD5 计算, 转化为多个小分片文件的 MD5 计算, 这样只要确保每个分片的 MD5 在上传前后一致, 即可确定在分片有序合并后的文件和原来保持一致[14,15].

2.4 断点续传

断点续传是指文件在上传过程中, 如果碰到网络故障或其他一些因素, 导致文件传输中断. 待故障恢复后, 可以接着未上传部分继续上传, 从而节省上传文件时间[16].

本文断点续传不同于其他时序传输的断点续传, 在某一分片首次上传时, 遇到网络抖动、高延时等短时网络故障, 即使该分片传输失败, 传输并不会立即中断, 而是会继续上传下一个分片, 直至网络恢复或最后一个分片上传结束. 在续传过程中, 并不是简单的从中断的文件偏移位置继续上传, 而会逐个校验分片的上传结果, 找出传输失败和 MD5 不一致的分片进行补充上传.

2.5 极速秒传

极速秒传是指当服务器上已经存在某个文件, 而用户又要上传此文件, 此时, 服务器则会检测此文件的特征码已被收录, 就可直接提示用户文件已上传完毕, 而免去用户再次上传的过程[17,18].

2.6 文件切片

要想实现断点续传、多并发上传、文件传输校验等技术, 关键在于文件分片, 而且是在文件上传前已经完成分片. 将大文件进行分片传输, 可以有效地避免文件上传过程中可能出现的失败问题[19,20].

在 HTML5 标准制定前, 如果实现文件分片, 则必须采用某些前端控件来实现; 从 HTML5 标准制定之后, 直接可以使用 HTML5 相应的 File API 功能函数, 解决文件在 Web 前端无插件分片的问题.

2.7 文件合并

目前有两种方法实现对上传分片文件进行合并, 一种是待所有分片传完之后统一合并, 这就需要服务端将接收到的所有分片进行临时存储, 最终合并也会消耗较长时间; 另一种方法是“边传边合并”[19], 服务端可以直接将上传过程中接收到文件分片, 按顺序追加保存至第一个分片文件中, 这就不需要存储临时文件, 且最终所有分片文件上传完毕同时也完成文件合并操作, 减少文件合并消耗时间.

3 方案设计

基于上述大文件上传方法与关键技术, 综合其优缺点, 本文提出一种基于 Web 的大文件安全高效上传方案. 该方案是基于 HTTP 协议, 利用 HTML5 的 XMLHttpRequest 2 特性[21], 采用 File API 提供的功能函数, 实现无客户端方式下文件分块传输. 并在首传和续传过程中对每一个分片进行 MD5 校验, 提高传输安全性. 同时利用多线程技术, 在前端构建线程池[22], 实现多线程的创建与管理, 提高文件传输的带宽利用率, 进而提升了文件的上传效率. 此外在处理重复文件上传时, 通过提取文件特征码, 匹配已上传的文件信息库, 如果匹配成功, 则可免去重复文件上传过程, 节约带宽资源及时间成本. 其基于 Web 的大文件高效上传方法的逻辑设计过程如图 1 所示.

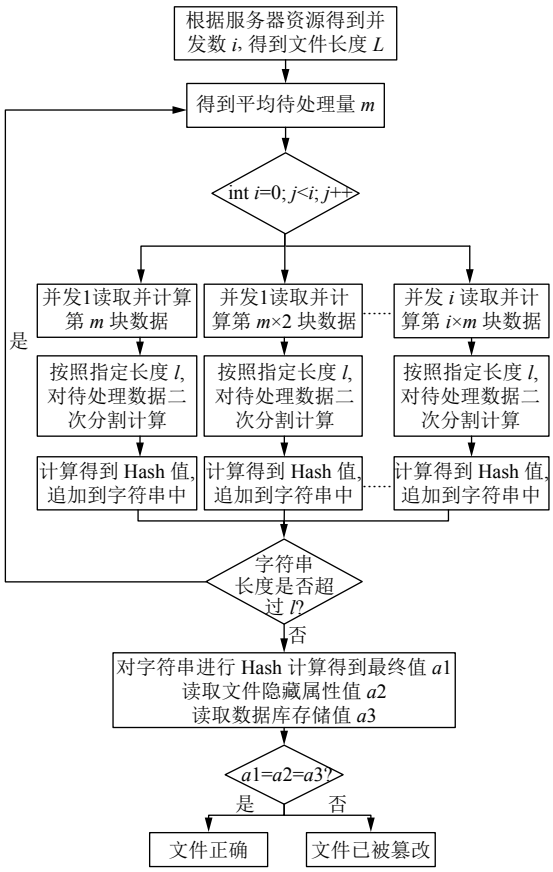

图 1 并发文件校验算法流程示意图

4 具体实现

针对文件上传流程示意图, 基于 Javascript 语言对流程中关键步骤进行代码实现, 具体如下所示.

4.1 文件分片

在前端分片传输过程中, 分片大小是一定的, 分片越小, 请求越多, 开销越大;分片越大, 灵活度越小,

分片上传的优势就会相对越不明显[6], 因此, 分片大小可以根据实际情况设置一个合适的值[23].

本文采取的 shardSize = 2×1024×1024. 初始化 start = currentChunk×shardSize;end = start + shardSize >= file.size? file.size : start + shardSize, 变量 start 表示的是分片的开始节点, 变量 end 表示的是分片的终止节点. 条件表达式 start + shardSize >= file.size? file.size : start + shardSize, 功能是获取下一个分片文件的终止节点. 其表达式如式 (1) 所示. 其中 x 代表分片序号 currentChunk, 取值从 0 开始, 递增量为 1; y 代表分片大小 shardSize, 是个固定值; z 代表整个文件的大小 file.size.

$$f(x,y,z)=\begin{cases} z, xy+y-z\geq 0 \\ xy+y, xy+y-z<0 \end{cases} \tag{1}$$

该表达式通过计算起始节点加分片大小获取文件上传传输的终止结点. 参数 arrayObj 为所有分片文件的 MD5 码存放数组, 文件的 MD5 码可以通过 spark.end() 函数生成.

大文件上传的前端分片及 MD5 计算关键代码摘录如下所示:

```
//每块文件读取完毕之后的处理
fileReader.onload = function (e) {
//每块交由 sparkMD5 进行计算
spark.append(e.target.result);
currentChunk++;
    var sparkShard = new SparkMD5.ArrayBuffer();
sparkShard.append(e.target.result);
arrayObj[currentChunk - 1] = sparkShard.end();
//如果文件处理完成则计算 MD5, 如果还有分片则继续处理
if (currentChunk < shardCount) {
var start = currentChunk * shardSize, end = start + shardSize >= file.size? file.size : start + shardSize;
    fileReader.readAsArrayBuffer(file.slice(start, end));
} else {
    fileMD5 = spark.end();
}
```

4.2 多并发上传

在并发上传过程中, 浏览器支持的最大并发数 t 是一定的, 因此设置的并发数如果超过该值, 则会造成浏览器在执行过程中并发请求排队等待. 所以, 并发数的设置越接近 t, 传输效率越大.

本文采取不同并发数来观察它的效率变化情况, 以控制并发数在一个合理值, poolSize 是最大并发请求数, poolCount 是活动并发数. 通过定时函数 setInterval() 来执行并发请求, 当 poolCount<poolSize 时, 就发起一个并发请求进行分片传输, poolCount 增加 1, 传输完成后 poolCount 减少 1, 以此来保证并发数在设定值.

多并发上传的关键代码摘录如下所示:

```
var idInt = setInterval(function () {
    console.log("ajax 对象总数:" + ajaxTotal + "计数器:" + intTotal);
    intTotal++;
    if (ajaxTotal > shardCount || cancel == 1) {
        clearInterval(idInt);
    }
    else {
        if (poolCount < poolSize) {
            poolCount++;
            console.log("ajax 对象池:" + poolCount + "/" + poolSize);
            //计算每一片的起始与结束位置
            var start = (parseInt(ajaxTotal) - 1) * shardSize,
                end = Math.min(size, start + shardSize);
            //构造一个表单, FormData 是 HTML5 新增的
            var form = new FormData();
            form.append("data", file.slice(start, end)); //slice 方法用于切出文件的一部分
            form.append("name", name);
            form.append("total", shardCount); //总片数
            form.append("index", ajaxTotal); //当前是第几片
```

```
                form.append("fileMD5", fileMD5); //所属文件 MD5
                form.append("md5", arrayObj[parseInt(ajaxTotal) - 1]); //分片 md5
                //Ajax 提交
                $.ajax({
                    url: "/Home/Upload",
                    type: "POST",
                    data: form,
                    async: true, //异步
                    processData: false, //很重要, 告诉 jquery 不要对 form 进行处理
                    contentType: false, //很重要, 指定为 false 才能形成正确的 Content-Type
                    success: function (jsonString) {
                        //var jsonObj = JSON.parse(jsonString);
                        if (jsonString.Status=="2"){
    $("#mergetime").text(jsonString.TimeTotal);
                        }
                        ++succeed;
                        $("#output").text(succeed + " / " + shardCount);
                        if (succeed==shardCount){
                        //计算上传耗时 (毫秒)
                        var time2=new Date();
    $("#uptime").text(daysBetween(time1, time2));
                        }
                    },
                    complete: function () {
                        poolCount--;
                    }
                });
            ajaxTotal++;
        }
    }
}, 1);
```

5 实验结果与分析

5.1 实验环境

本次实验服务器资源配置如表 1 所示.

表 1 服务器配置信息

类型	详细描述
处理器	型号 AMD Opteron(tm) Processor 6376, 主频 2.30 GHz, 单处理器 2 核心
内存	4 GB
网卡	1000 Mbps
操作系统	Windows Server 2016 Standard

为排除其它因素对本次实验的影响, 此次实验仅保留原生的操作系统, 不安装任何软件/服务/组件 (测试程序除外), 最大程度的减少实验影响因素.

5.2 实验结果

本文提出基于 Web 的大文件安全高效上传方法主要设置参数为浏览器请求并发数, 为验证其灵活性及通用性, 本次实验在保持分片大小不变的基础上, 采用不同并发数上传同一文件进行验证, 重复进行 3 次, 实现结果如图 2 所示.

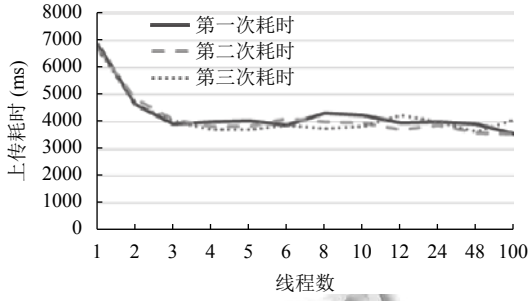

图 2 不同线程数文件上传耗时图

通过图 2 实验结果可得出文件传输耗时是 L 型变化的, 请求并发数设置为 3 时产生拐点. 请求并发数与耗费时间在拐点之前呈反比关系, 耗费时间随着并发数的增大而线性减少, 由于网络带宽以及浏览器软件等实验环境因素的影响, 在拐点之后耗费时间趋于平稳.

5.3 实验分析

对于文件上传来说其优劣主要由传输的正确性、时间成本、适应性、可移植性、鲁棒性这五个方面决定, 通过这些判断条件能够很好的评估大文件安全高效上传方案是否准确高效.

本文提出的大文件安全高效上传方案基于 Web 方式进行优化, 利用 HTML5 文件接口对大文件进行前端切片, 后端合并, 其本质相当于对待上传文件进行了一次完整的数据传输, 最终得到结果文件与上传文件完全相符. 在保持文件内容不变的前提下, 使用并发上传能够大大节约时间成本. 其中基于 Web 和

HTML5 切片的传输方式, 几乎能适应所有规模的文件 Web 上传环境; 并通过断点续传确保了 Web 传输的可靠性.

6 总结

本文提出基于 Web 的大文件高效上传方法, 在传统单线程大文件上传基础上, 采用了多并发上传, 充分利用带宽资源提高传输效率, 对于存储的资源文件进行特征库收录, 实现已收录文件的极速秒传, 节约了时间成本. 同时对于大文件的 MD5 计算, 利用循环切片计算的方式极大的降低磁盘 I/O 和内存的占用. 该方法普适性强、能够很好地在文件传输过程中提高上传效率、提升稳定性、增强安全性, 从而更好地为用户服务.

参考文献

1 刘苡, 吴刚. 基于. Net 的 Web 应用系统中大文件传输方案的研究. 微型电脑应用, 2012, 28(7): 27–30. [doi: 10.3969/j.issn.1007-757X.2012.07.008]

2 黎苑文, 程明智, 徐秀花, 等. 断点续传及多线程机制在远程传版中的应用研究. 北京印刷学院学报, 2012, 20(6): 53–56. [doi: 10.3969/j.issn.1004-8626.2012.06.019]

3 欧阳侃夫, 呼和. 广域网大数据传输方案的实现. 计算机应用, 2014, 34(S1): 35–37.

4 李文杰, 高鹏翔. 基于 Web Service 的大文件传输设计与实现. 计算机时代, 2014, (1): 27–28, 31.

5 王建斌, 赵靓. Web 上传文件的三种解决方案. 计算机与信息技术, 2011, 19(S1): 65–68.

6 王莉敏, 梁正和, 段全锋. 基于 HTML5 大文件断点续传的实现方案. 计算机与现代化, 2016, (3): 91–95. [doi: 10.3969/j.issn.1006-2475.2016.03.018]

7 秦绪彬. 广播节目传输中的断点续传和多线程技术运用. 黑龙江科技信息, 2016, (19): 18.

8 夏雪刚. 基于多线程文件传输关键技术研究与实现. 电脑知识与技术, 2016, 12(21): 48–50.

9 陈涛, 黄艳峰. Java Web 开发中文件上传方法研究与实现. 电脑知识与技术, 2016, 12(11): 48–49, 52.

10 刘杨. JSP 项目开发常用文件上传组件比较及举例. 电脑编程技巧与维护, 2015, (8): 15–16, 35. [doi: 10.3969/j.issn.1006-4052.2015.08.006]

11 眭俊华, 刘慧娜, 王建鑫, 等. 多核多线程技术综述. 计算机应用, 2013, 33(S1): 239–242, 261.

12 赵晓雨. 基于文件内容特征的取证方法研究. 工业控制计算机, 2016, 29(8): 26–27. [doi: 10.3969/j.issn.1001-182X.2016.08.011]

13 方燕飞, 王俊, 何王全. 基于多层 MD5 消息摘要的文件完整性实时检测技术. 计算机应用与软件, 2015, 32(1): 20–23. [doi: 10.3969/j.issn.1000-386x.2015.01.006]

14 毛熠, 陈娜. MD5 算法的研究与改进. 计算机工程, 2012, 38(24): 111–114, 118. [doi: 10.3969/j.issn.1000-3428.2012.24.027]

15 段国云, 陈浩, 黄文, 等. 一种 Web 程序防篡改系统的设计与实现. 计算机工程, 2014, 40(5): 149–153. [doi: 10.3969/j.issn.1000-3428.2014.05.031]

16 路石坚. 一种基于 HTTP 的断点续传客户端. 电脑编程技巧与维护, 2017, (9): 71–72. [doi: 10.3969/j.issn.1006-4052.2017.09.027]

17 胡渝苹. 文件秒传系统在云存储环境下的设计与实现. 计算机应用与软件, 2016, 33(4): 329–333. [doi: 10.3969/j.issn.1000-386x.2016.04.076]

18 赵大军. 基于 Silverlight 的大文件上传技术研究. 数字技术与应用, 2014, (7): 107–108.

19 邹鹤敏, 黄海于. 大文件分块上传和下载软件的设计与实现. 电子技术应用, 2013, 39(8): 137–139. [doi: 10.3969/j.issn.0258-7998.2013.08.040]

20 叶文全. 基于 HTML5、AJAX 的文件分割上传与加密存储研究. 三明学院学报, 2018, 35(4): 60–66.

21 刘耀钦. 利用 HTML5 拖放技术实现多文件异步上传. 四川理工学院学报 (自然科学版), 2015, 28(1): 17–20, 30.

22 任双君, 周旭, 任勇毛, 等. 基于 HTML5 的浏览器端多线程下载技术. 计算机系统应用, 2017, 26(11): 11–18. [doi: 10.15888/j.cnki.csa.006091]

23 孟欣. 基于 FTP 的文件高效上传方法的研究与实现[硕士学位论文]. 广州: 华南理工大学, 2014.