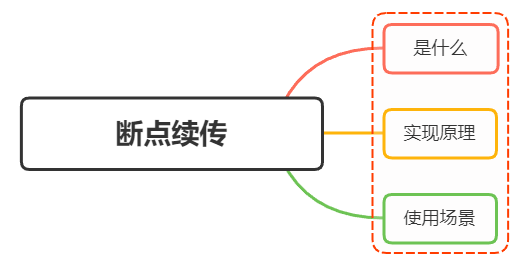
# 面试官：大文件上传如何做断点续传？



## 一、是什么

不管怎样简单的需求，在量级达到一定层次时，都会变得异常复杂

文件上传简单，文件变大就复杂

上传大文件时，以下几个变量会影响我们的用户体验

* 服务器处理数据的能力
* 请求超时
* 网络波动

上传时间会变长，高频次文件上传失败，失败后又需要重新上传等等

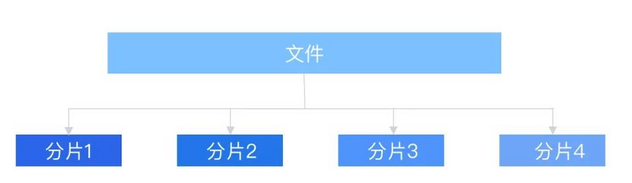
为了解决上述问题，我们需要对大文件上传单独处理

这里涉及到分片上传及断点续传两个概念

#### 分片上传

分片上传，就是将所要上传的文件，按照一定的大小，将整个文件分隔成多个数据块（Part）来进行分片上传

如下图



上传完之后再由服务端对所有上传的文件进行汇总整合成原始的文件

大致流程如下： 1. 将需要上传的文件按照一定的分割规则，分割成相同大小的数据块； 2. 初始化一个分片上传任务，返回本次分片上传唯一标识； 3. 按照一定的策略（串行或并行）发送各个分片数据块； 4. 发送完成后，服务端根据判断数据上传是否完整，如果完整，则进行数据块合成得到原始文件

#### 断点续传

断点续传指的是在下载或上传时，将下载或上传任务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或下载，如果碰到网络故障，可以从已经上传或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。用户可以节省时间，提高速度

一般实现方式有两种：

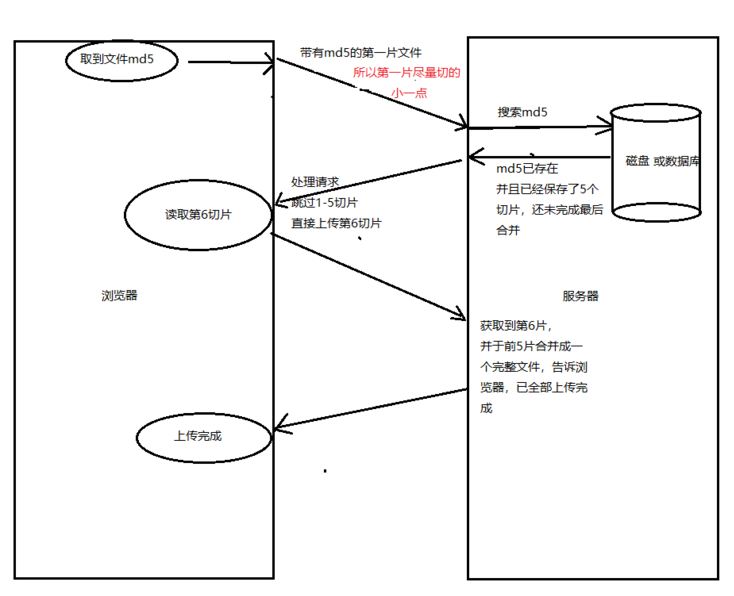
* 服务器端返回，告知从哪开始
* 浏览器端自行处理

上传过程中将文件在服务器写为临时文件，等全部写完了（文件上传完），将此临时文件重命名为正式文件即可

如果中途上传中断过，下次上传的时候根据当前临时文件大小，作为在客户端读取文件的偏移量，从此位置继续读取文件数据块，上传到服务器从此偏移量继续写入文件即可

## 二、实现思路

整体思路比较简单，拿到文件，保存文件唯一性标识，切割文件，分段上传，每次上传一段，根据唯一性标识判断文件上传进度，直到文件的全部片段上传完毕



下面的内容都是伪代码

读取文件内容：

const input = document.querySelector('input');  
input.addEventListener('change', function() {  
 var file = this.files[0];  
});

可以使用md5实现文件的唯一性

const md5code = md5(file);

然后开始对文件进行分割

var reader = new FileReader();  
reader.readAsArrayBuffer(file);  
reader.addEventListener("load", function(e) {  
 //每10M切割一段,这里只做一个切割演示，实际切割需要循环切割，  
 var slice = e.target.result.slice(0, 10\*1024\*1024);  
});

h5上传一个（一片）

const formdata = new FormData();  
formdata.append('0', slice);  
//这里是有一个坑的，部分设备无法获取文件名称，和文件类型，这个在最后给出解决方案  
formdata.append('filename', file.filename);  
var xhr = new XMLHttpRequest();  
xhr.addEventListener('load', function() {  
 //xhr.responseText  
});  
xhr.open('POST', '');  
xhr.send(formdata);  
xhr.addEventListener('progress', updateProgress);  
xhr.upload.addEventListener('progress', updateProgress);  
  
function updateProgress(event) {  
 if (event.lengthComputable) {  
 //进度条  
 }  
}

这里给出常见的图片和视频的文件类型判断

function checkFileType(type, file, back) {  
/\*\*  
\* type png jpg mp4 ...  
\* file input.change=> this.files[0]  
\* back callback(boolean)  
\*/  
 var args = arguments;  
 if (args.length != 3) {  
 back(0);  
 }  
 var type = args[0]; // type = '(png|jpg)' , 'png'  
 var file = args[1];  
 var back = typeof args[2] == 'function' ? args[2] : function() {};  
 if (file.type == '') {  
 // 如果系统无法获取文件类型，则读取二进制流，对二进制进行解析文件类型  
 var imgType = [  
 'ff d8 ff', //jpg  
 '89 50 4e', //png  
  
 '0 0 0 14 66 74 79 70 69 73 6F 6D', //mp4  
 '0 0 0 18 66 74 79 70 33 67 70 35', //mp4  
 '0 0 0 0 66 74 79 70 33 67 70 35', //mp4  
 '0 0 0 0 66 74 79 70 4D 53 4E 56', //mp4  
 '0 0 0 0 66 74 79 70 69 73 6F 6D', //mp4  
  
 '0 0 0 18 66 74 79 70 6D 70 34 32', //m4v  
 '0 0 0 0 66 74 79 70 6D 70 34 32', //m4v  
  
 '0 0 0 14 66 74 79 70 71 74 20 20', //mov  
 '0 0 0 0 66 74 79 70 71 74 20 20', //mov  
 '0 0 0 0 6D 6F 6F 76', //mov  
  
 '4F 67 67 53 0 02', //ogg  
 '1A 45 DF A3', //ogg  
  
 '52 49 46 46 x x x x 41 56 49 20', //avi (RIFF fileSize fileType LIST)(52 49 46 46,DC 6C 57 09,41 56 49 20,4C 49 53 54)  
 ];  
 var typeName = [  
 'jpg',  
 'png',  
 'mp4',  
 'mp4',  
 'mp4',  
 'mp4',  
 'mp4',  
 'm4v',  
 'm4v',  
 'mov',  
 'mov',  
 'mov',  
 'ogg',  
 'ogg',  
 'avi',  
 ];  
 var sliceSize = /png|jpg|jpeg/.test(type) ? 3 : 12;  
 var reader = new FileReader();  
 reader.readAsArrayBuffer(file);  
 reader.addEventListener("load", function(e) {  
 var slice = e.target.result.slice(0, sliceSize);  
 reader = null;  
 if (slice && slice.byteLength == sliceSize) {  
 var view = new Uint8Array(slice);  
 var arr = [];  
 view.forEach(function(v) {  
 arr.push(v.toString(16));  
 });  
 view = null;  
 var idx = arr.join(' ').indexOf(imgType);  
 if (idx > -1) {  
 back(typeName[idx]);  
 } else {  
 arr = arr.map(function(v) {  
 if (i > 3 && i < 8) {  
 return 'x';  
 }  
 return v;  
 });  
 var idx = arr.join(' ').indexOf(imgType);  
 if (idx > -1) {  
 back(typeName[idx]);  
 } else {  
 back(false);  
 }  
  
 }  
 } else {  
 back(false);  
 }  
  
 });  
 } else {  
 var type = file.name.match(/\.(\w+)$/)[1];  
 back(type);  
 }  
}

调用方法如下

checkFileType('(mov|mp4|avi)',file,function(fileType){  
 // fileType = mp4,  
 // 如果file的类型不在枚举之列，则返回false  
});

上面上传文件的一步，可以改成：

formdata.append('filename', md5code+'.'+fileType);

有了切割上传后，也就有了文件唯一标识信息，断点续传变成了后台的一个小小的逻辑判断

后端主要做的内容为：根据前端传给后台的md5值，到服务器磁盘查找是否有之前未完成的文件合并信息（也就是未完成的半成品文件切片），取到之后根据上传切片的数量，返回数据告诉前端开始从第几节上传

如果想要暂停切片的上传，可以使用XMLHttpRequest的 abort方法

## 三、使用场景

* 大文件加速上传：当文件大小超过预期大小时，使用分片上传可实现并行上传多个 Part， 以加快上传速度
* 网络环境较差：建议使用分片上传。当出现上传失败的时候，仅需重传失败的Part
* 流式上传：可以在需要上传的文件大小还不确定的情况下开始上传。这种场景在视频监控等行业应用中比较常见

## 小结

当前的伪代码，只是提供一个简单的思路，想要把事情做到极致，我们还需要考虑到更多场景，比如

* 切片上传失败怎么办
* 上传过程中刷新页面怎么办
* 如何进行并行上传
* 切片什么时候按数量切，什么时候按大小切
* 如何结合 Web Worker 处理大文件上传
* 如何实现秒传

人生又何尝不是如此，极致的人生体验有无限可能，越是后面才发现越是精彩 ~\_~

## 参考文献

* https://segmentfault.com/a/1190000009448892
* https://baike.baidu.com/