Node.js如何解析Form上传？

author : 黄俊涛

原文地址： Node.js如何解析Form上传？

NPM 和 GitHub 里的开源组件帮我们解决了很多繁琐的工作，但是也让我们失去了很多深入技术细节的机会。在现有组件无法满足我们需求的时候，就需要我们来自己动手丰衣足食了。

作者前段时间遇到了一个需要手动解析 Form 表单上传的机会，也借此为各位解析一下 Node.js 解析 Form 上传的实现细节。

背景

半年前微信小程序的推出，掀起了一股小程序开发的热潮，作者一样收到了来自产品妹子的小程序开发需求。

需求中涉及到照片 / 视频上传的功能，而我们本身是全国最大的照片上传服务产品 --QQ 空间相册，我们各主要上传渠道的上传都采用了 socket 分片上传技术，包括客户端和 PC 浏览器端。微信小程序提供的上传接口只能是 Form 表单上传，且又没有提供读取本地文件内容的接口，打消了我们通过客户端分片上传的念头。

所以现在的问题是从小程序发出的是 Form 表单的上传请求，后端提供的是基于 socket 的分片上传服务，我们需要把两者对接起来。

我们的解决方案就是在 Node.js 层加一个适配接入，接收来自小程序的 Form 上传请求，然后解析图片内容，并分片上传到后端。

wx.uploadFile(OBJECT)

将本地资源上传到开发者服务器。如页面通过 wx.chooseImage 等接口获取到一个本地资源的临时文件路径后，将此路径传给 wx.uploadFile，将本地资源上传到开发者服务器。客户端发起一个 HTTPS POST 请求，其中 `content-type` 为 `multipart/form-data` 。

OBJECT参数说明：

参数	类型	必填	说明
url	String	是	开发者服务器 url
filePath	String	是	要上传文件资源的路径
name	String	是	文件对应的 key，开发者在服务器端通过这个 key 可以
header	Object	否	HTTP 请求 Header，header 中不能设置 Referer
formData	Object	否	HTTP 请求中其他额外的 form data
success	Function	否	接口调用成功的回调函数
fail	Function	否	接口调用失败的回调函数
complete	Function	否	接口调用结束的回调函数（调用成功、失败都会执行）

方案

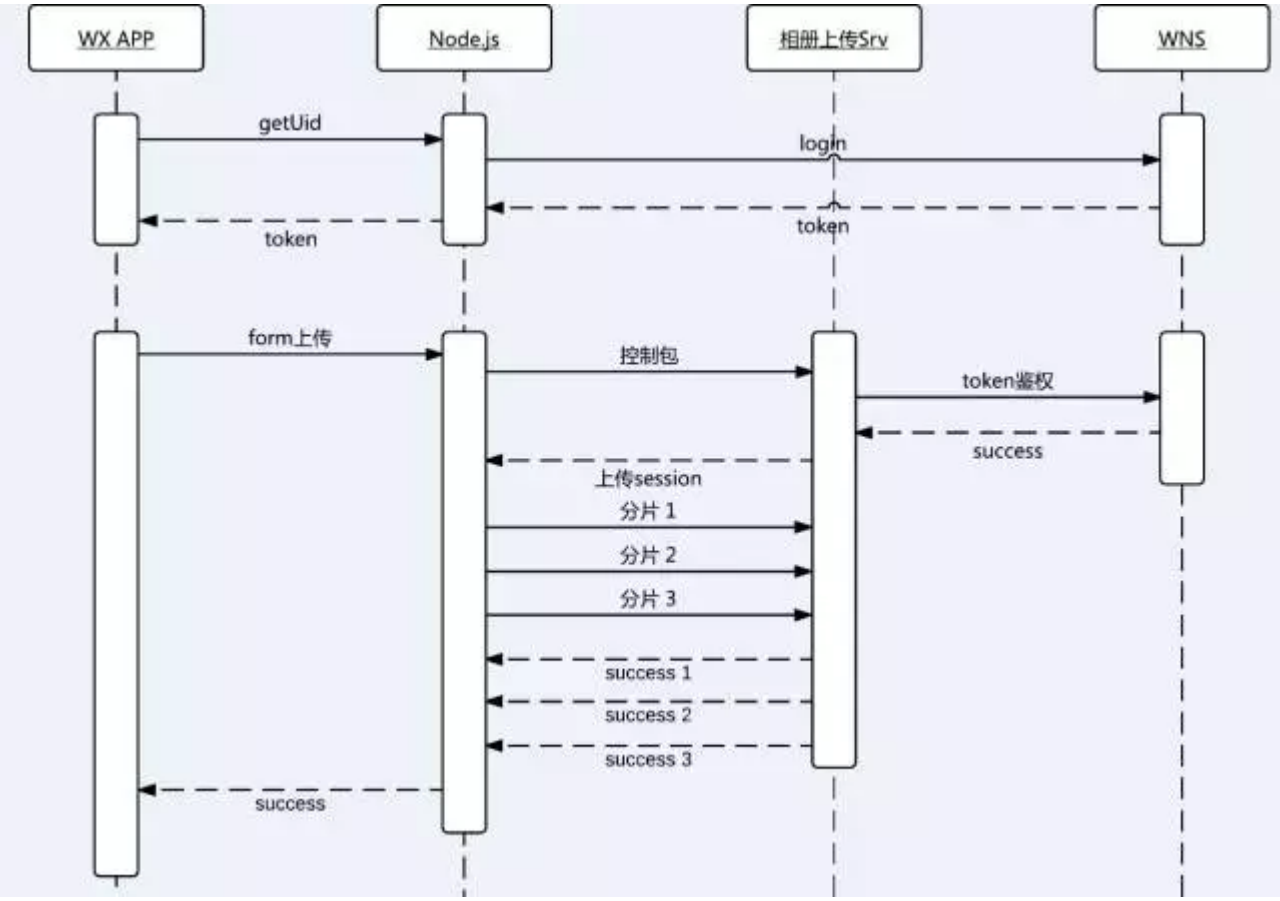
现在的任务就分为两部分：

1. 解析 Form 上传表单，将图片 / 视频内容解析出来；
2. 将图片 / 视频内容拆分成固定大小的小片数据包，通过后端提供的私有协议接口上传；

那在这里我们又面临了一个选择，是等整个 Form 表单接收完成再进行分片上传，还是收到 Form 表单就开始上传呢？

前者我们可以直接引入现成的 multipart 的 Node.js 解析组件，临时存储到本地或者内存，然后进行上传，但是缺点是整个上传过程变成了两个串行过程，上传延时会增加，另外我们同时支持图片和视频的上传，会导致 Node.js 侧的内存占用比较高。

而后者则是尽量同步，接收到图片 / 视频 part 我们就可以开始上传，只是没有刚好满足我们需求的组件来支持。基于以上对比我们选择后面的方案，自己来解析 Form 上传表单，整个上传流程可以看下面的时序图。



Form 表单

Form 表单我们平时并不少接触，但是需要自己去解析的场景并不多，在解析 Form 表单前，我们先简单看一下 Form 表单的结构是怎么样的。

```
--${bound}
Content-Disposition: form-data; name="Filename"

IMG-0719.jpg
--${bound}
Content-Disposition: form-data; name="filedata"; filename="IMG-0719.jpg"
Content-Type: application/octet-stream

file content
--${bound}
Content-Disposition: form-data; name="Upload"

Submit Query
--${bound}--
```

可以看到每一个表单 field 都是以一个 `--{bound}` 开头的，然后是一些属性信息，属性和内容之间都有一个空行，整个 Form 表单则以 `--{bound}--` 结束。那 bound 是一个什么存在呢？答案就在 Http 请求头里，因为 Form 请求的 Content-type 一般是这样的：

```
Content-Type: multipart/form-data; boundary=--4934898438743465560214 6
```

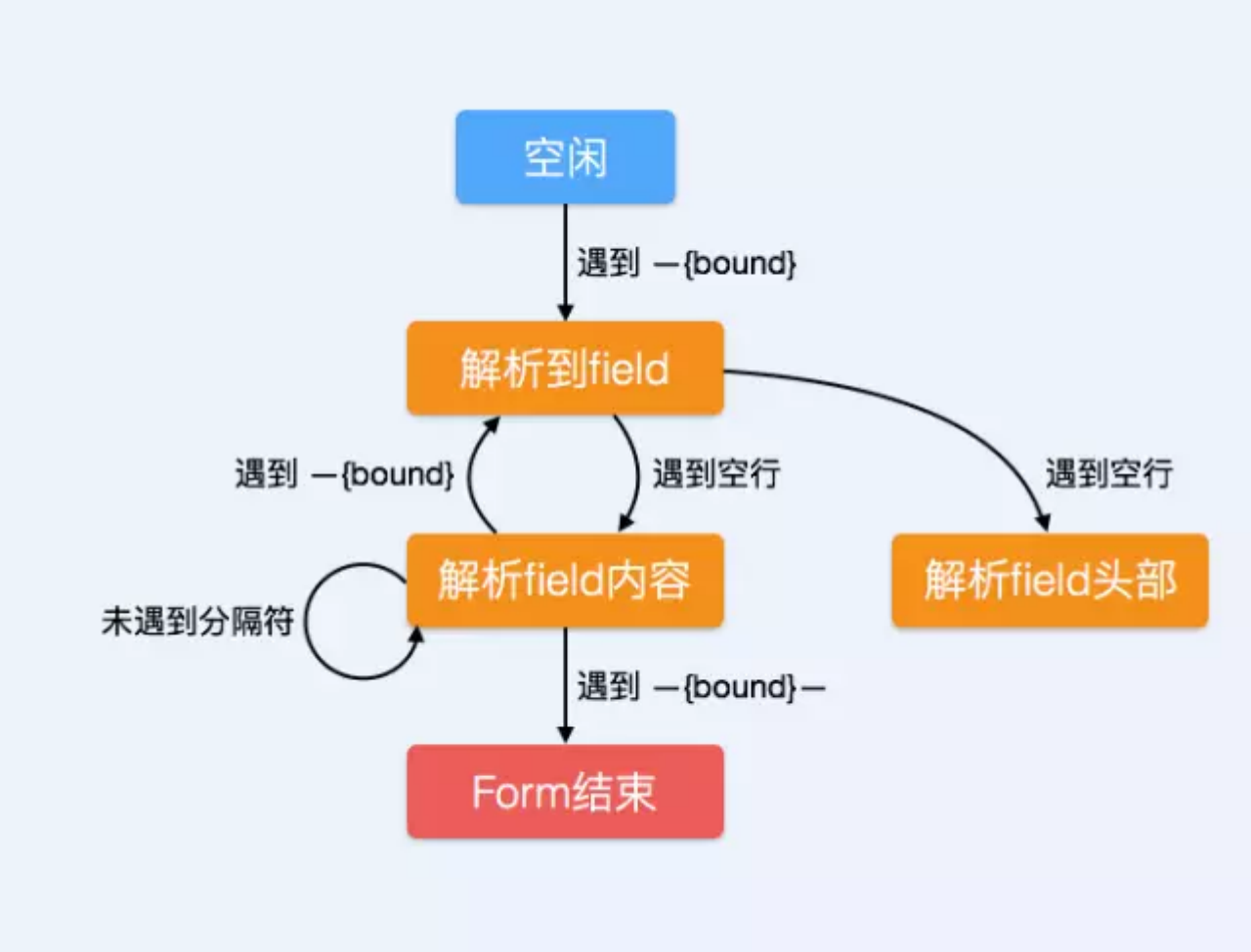
那 `--493489843874346556021 46` 就是我们要找的 bound，作用就是将 Form 表单里的 field 分隔开来，所以这个串一般比较复杂，并且有足够的长度。

再整理一下我们的思路：

1. 先从 Http 请求头里找到 Content-type，并渠道 boundary 的内容，该内容为表单分隔符；
2. 对表单内容从前往后查找 `--{bound}` 和 `--{bound}--`，`--{bound}` 与 `--{bound}` 或者 `--{bound}--` 之间的内容就是单个表单 field 内容；
3. 单个表单内容第一个空行之前的部分为表单头部，头部以换行符分隔，空行后面的内容则为 field 内容；

解析

Form 表单是流式的，表单 field 的查找其实是线性的，且是有状态的，整个解析过程我们可以看成是一个有限状态机，而 Form 请求的 data 事件则是这个状态机的驱动器。



可以看到解析器可能存在 5 个状态，分别为：

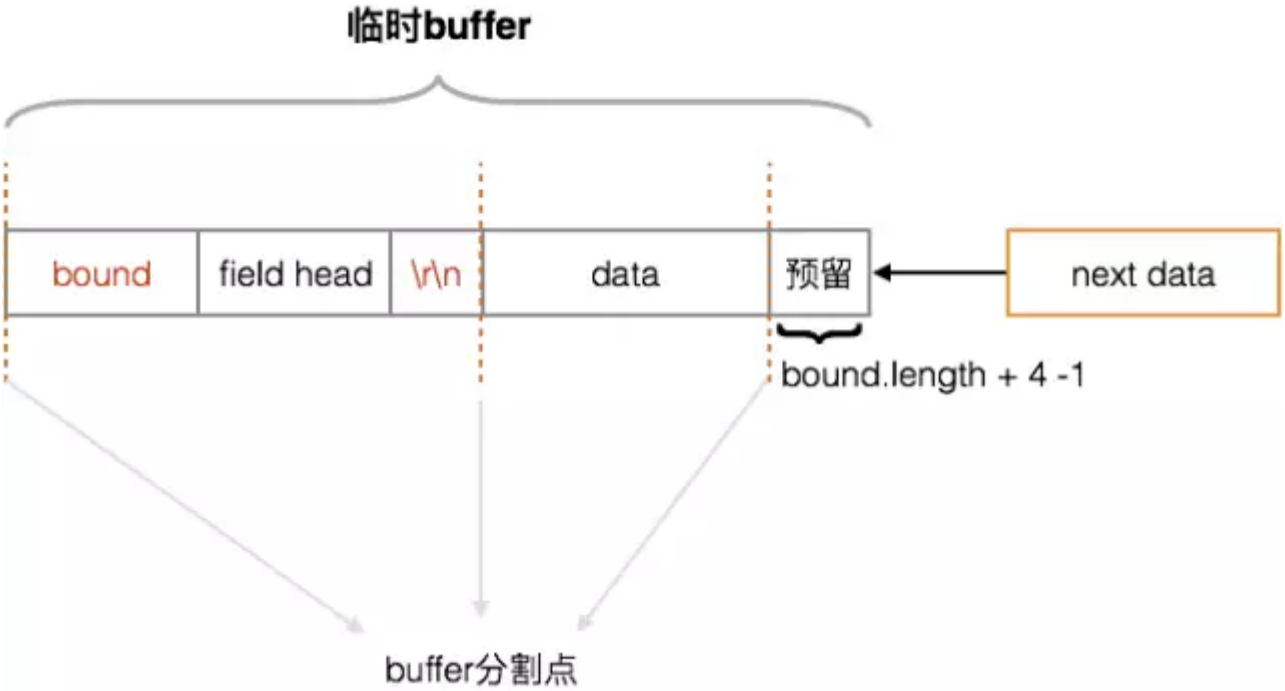
1. 空闲，即初始状态；
2. 解析到 field，即进入到待解析 field 内容的状态，后面的内容都存入临时 buffer，每次 data 事件都会更新这个 buffer，并触发一次检查是否有一个空行存在，即一组 `\r\n\r\n`；
3. 解析 field 头部，在遇到 `\r\n\r\n` 之后，临时 buffer 空行前面的内容即为 field 头部，根据换行拆分出单个头部，基于字符串解析就能获取到头部信息，这里就不详解了；
4. 解析 field 内容，再遇到 `\r\n\r\n` 之后，临时 buffer 空行后面的内容就包含了 field 的内容，但是要注意，**空行后面的内容并不是全都是当前 field 的内容**，我们需要对临时 buffer 空行后面的内容中寻找 `--{bound}` 或者 `--{bound}--`，如果查找到 `--{bound}`，则分隔符前面的内容为当前 field 内容，分隔符后面的内容为下一

个 field，那解析器则进入到状态 2 《**解析到 field**》；如果查找到 `--{bound}--`，则意味着当前 field 为 Form 表单最后一个 field，分隔符前面的内容为 field 内容，同时进入到状态 5 《**Form 结束**》；如果临时 buffer 里 `--{bound}` 和 `--{bound}--` 都没有找到，意味着当前 field 内容还没有接受完成，那么空行后面的内容均为 field 内容，每次 data 事件则会触发上述状态转换查询逻辑进行一遍；

5. Form 结束，即整个 Form 表单解析完成了。

再回到我们的上传场景中，我们需要将收到的图片 / 视频内容同步打包传给后端，意味着在状态 4 的处理中，如果 field 是图片 / 视频内容，就需要将收到的文件部分打包传输。在查找到 `--{bound}` 和 `--{bound}--` 的情况下比较好处理，分隔符前的内容都是文件内容，直接拆分打包传输就可以了，但是在 `--{bound}` 和 `--{bound}--` 都没有找到的时候，我们将哪部分内容进行打包呢？

虽然没有查找到这两种分隔符，但是有可能当前 buffer 的结尾已经包含了部分分隔符的内容，所以我们需要预留出一部分 buffer，这段 buffer 前面的部分认为是文件内容是安全的，这段 buffer 只能等与后续收到的 data 拼接在一起才知道是不是文件内容。因为预留的这段 buffer 的目的是为了防止将分隔符当做文件内容传输了，所以预留的这段 buffer 长度应该不短于 buffer 长度减一，即 buffer 的长度为 `bound.length + 4 - 1`，4 代表 4 个 `-` 长度。



总结

本文只是根据实际需求和应用场景介绍了 Form 表单的解析细节，并附带介绍了在 Form 表单流式接收的过程中流式代理上传的方案。

目前前端技术栈越来越丰富，前端社区越来越活跃，我们有海量的开源组件，唾手可得的框架库以及全链路支持的工具集，但是我们仍然需要掌握能探究技术细节的能力，在关键技术节点上往往还是要回归到技术细节本身。