课程

工作日常　　面试重点

表单提交方式

在表单提交数据的时候，浏览器内部的实现，其实是利用了FormData这样的一个类型，这个类型里面是key/value ,当它真正被传输的时候，就是变成http的body的时候，它会变成用Boundary分隔的数据。观察用表单提交数据时POST到服务器端的数据body是怎样的?观察用表单提交文件时body是怎样的？Express如何接收表单数据提交的数据?

服务端代码

```
//简单的接收文件的demo(开启一个3000端口的服务))
const express = require('express')
const path =  require('path')
const fileUpload = require('express-fileupload')

const app = express()
//访问localhost:3000/submit 返回formdata.html
app.get('/submit',(req,res)=>{
    // __dirname 当前目录的绝对目录
      res.sendFile(path.resolve(__dirname,'formdata.html'))
})
 //服务端接收文件，这里用了一个中间件去解析body
app.post('/file',fileUpload(),(req,res)=>{
  req.files.file.mv(path.resolve(__dirname,'upload/aaa.txt'))
  res.status(201).send('ok')
})

app.listen(3000,()=>{
    console.log('进入服务')
})
```

客户端的formHTML

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
     <h2>HTML-Forms</h2>
     <!-- 默认body的传输格式就是 application/x-www-form-urlencoded    使用x-www-form-urlencoded 只能带上文件名 文件数据带不上 文件数据太大-->
     <form action="/file" method="post"  enctype="multipart/form-data">
         <label for="fname">First Name</label>
         <input type="text" id="fname" name="fname" value="张">
         <br>
         <label for="lname">Last Name</label>
         <input type="text" id="lname" name="lname" value="三">
         <br>
         <br>
         <label for="file">file:</label>
         <br>
         <input type="file" id="file" name="file">
         <br>
          <br>
          <input type="submit" value="Submit">
     </form>
</body>
</html>
```

formData展示(Boundary 分割数据)

```
------WebKitFormBoundaryxjgS9lXiAeu0QNsj
Content-Disposition: form-data; name="fname"

张
------WebKitFormBoundaryxjgS9lXiAeu0QNsj
Content-Disposition: form-data; name="lname"

三
------WebKitFormBoundaryxjgS9lXiAeu0QNsj
Content-Type: text/plain

 1.客户详情 -- 业务信息 ： 选择业务类型、选择审批状态和业务信息列表的对接，对接的相关后台
高龙
                新增业务新增和列表的详情 -- 对接后台人员：欧阳高龙
               列表导出 -- 对接后台人员 宋富豪

需要注意的地方：接口的传参以及获取参数的时候可能会出现异步的问题。


2.客户详情 -- 结算单：选择业务类型、选择审批状态和业务信息列表的对接，对接的相关后台人
              列表的详情 -- 对接后台人员：欧阳高龙
              列表导出 -- 对接后台人员 宋富豪

需要注意的地方：接口的传参以及获取参数的时候可能会出现异步的问题。


3.PC端合同--新增合同：对接后台人员：郑丁源

需要注意的地方：1.可能会被要求更改合同模板样式问题
                  2.合同字段相关权限以及校验问题


4.PC端合同--编辑合同：对接后台人员：郑丁源

需要注意的地方：1.组件的传值问题（暂时定为传 合同详情对象）
                  2.合同字段相关权限以及校验问题

5.Admin端合同--合同模板：增删改查，对接后台人员：郑丁源

需要注意的地方：1.可能会被要求更改合同模板样式问题
                  2.相关权限以及校验问题
```

HTML5方式

- **base64方式**

base64是用64个打印字符(A-Za-z0-9) A-Z a-z 0-9来编码二进制 26 + 26 +10 + (加号和逗号)

1. 0-255的ascii码中有许多不能打印的字符
2. 不能打印的字符就不需要在中间过程中转义，如空格变成%2

下面就有个实战 HTML5 + Base64 上传文件

1. 观察HTML5的文件api使用

2. 观察base64方法(四个可见字母，对于大文件来说base64完全是不可以接收的，大文件压缩成64需要消耗非常久的时间，更何况压缩成base64使得文件变得更大)

3. 观察的直接用流上传

客户端:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <label for="file">file:</label>
        <input type="file" id="file" name="file">
        <input type="submit" onclick="submit()" value="Submit">
    </div>
</body>
<script>
 //这只是一个demo ， 才用一个全局变量去缓存,正式场景不能这样
 let   upload = {}
 document.getElementById('file').addEventListener('change',(e)=>{
            const files = e.target.files
            // file类型就是File
            // file的父类就是Blob  file的_proto_指向的是Blob
            // Blob本身就是用来描述文件的，有人说为什么不是Buffer,Buffer是缓冲区，
            // Buffer不能代表数据,只是用来缓数据的,但是Steam流可以代表数据，但是web端是没有这个Steam这个类型
            for(let file of files){
                //通过 new FileReader 把Blob的base64字符串 通过readAsDataURL 读取出来 Base64对于大文件是不可接收的
                const fr = new FileReader()
                fr.readAsDataURL(file)
                //fr初始化加载
                fr.onload =function(){
                    console.log(fr.result)
                    upload.data = fr.result.substr(22)
```

```
                    upload.name = file.name
                }
            }
            // console.log(files)
        })
    function submit(){
        fetch('/fileb64',{
            method:'POST',
            body:JSON.stringify(upload),
            headers:{
            'Content-type':'application/json'
             }
            }
            )
    }
</script>
</html>
```

服务端:

```
const express = require('express')
const path =  require('path')
const fs = require('fs')
const bodyParser = require('body-parser')


const app = express()
app.get('/submit',(req,res)=>{
    // __dirname 当前目录的绝对目录
      console.log(req)
      res.sendFile(path.resolve(__dirname,'html5_base64.html'))
})
 //服务端接收文件，这里用了一个中间件去解析body
app.post('/fileb64',bodyParser.json(),(req,res)=>{
  console.log(req.body)
  const  buffer = new Buffer(req.body.data,'base64')
  fs.writeFileSync(path.resolve(__dirname,`upload/${req.body.name}`),buf
fer)
  res.status(201).send('created')
})

app.listen(3000,()=>{
```

```
    console.log('进入服务')
})
```

- **Blob方式(主流方式)**

客户端:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <label for="file">file:</label>
        <input type="file" id="file" name="file">
        <input type="submit" onclick="submit()" value="Submit">
    </div>
</body>
<script>
 //这只是一个demo ， 才用一个全局变量去缓存,正式场景不能这样
 let   upload = {}
 document.getElementById('file').addEventListener('change',(e)=>{
            const files = e.target.files
            // file类型就是File
            // file的父类就是Blob  file的_proto_指向的是Blob
            // Blob本身就是用来描述文件的，有人说为什么不是Buffer,Buffer是缓冲区，
            // Buffer不能代表数据,只是用来缓数据的,但是Steam流可以代表数据，但是web端是没有这个Steam这个类型
            for(let file of files){
                //Blob方式上传就是直接上传Blob数据
                upload.data = file
                upload.name = file.name
            }
            // console.log(files)
        })
    function submit(){
       //创建一个formData对象
```

```
        const formData = new FormData()
        formData.append('file',upload.data)
        formData.append('name',upload.name)
        fetch('/fileBlob',{
             method:'POST',
             body:formData,
             headers:{
              //这里有个bug   当数据格式是multipart/form-data fetch不需要手动提案加header 'Content-type':'multipart/form-data' 其他的库是需要的
             // 'Content-type':'multipart/form-data'
              }
             }
             )
    }
</script>
</html>
```

服务端:

```
const express = require('express')
const path =  require('path')
const fs = require('fs')
const bodyParser = require('body-parser')
const fileUpload = require('express-fileupload')


const app = express()
app.get('/submit',(req,res)=>{
    // __dirname 当前目录的绝对目录
     console.log(req)
     res.sendFile(path.resolve(__dirname,'blob.html'))
})
 //服务端接收文件，这里用了一个中间件去解析body
app.post('/fileBlob',fileUpload(),(req,res)=>{
    req.files.file.mv(path.resolve(__dirname,`upload/${req.body.name}`))
    res.status(201).send('ok')
})

app.listen(3000,()=>{
    console.log('进入服务')
})
```

base64和Blob方式优劣势

Blob比bese64更加传输更快

base64对cpu压力更小

本章小结

- 上传文件和提交输入框中的字段没有本质的区别
- Base64方法速度慢，没有特殊原因通常用blob上传