构建文件上传服务器

by calidion

处理POST请求

处理最基本的POST请求

```
if (req.method === "POST") {
  var data = [];
  var length = 0;
  req.on('data', function (chunk) {
      length += chunk.length;
      data.push(chunk);
  });
  req.on('end', function () {
    data = Buffer.concat(data, length);
    // process(data)
  });
}
```

测试POST处理：

```
curl -X POST -d "a=1010&b=100&c=中文" http://127.0.0.1:8080
```

处理文件的提交

1. 获取multi-part属性
   处理文件时需要确定提交的内容属性是
   `multipart/form-data`

```
var contentType =  req.headers['content-type'];

if (contentType.indexOf('multipart/form-data;') !== -1) {
// 有上传文件内容
}
```

获取分割符号(boundary)

*分割符是放在**content-type**里面的。*

所以我们获取`boundary`时需要按下面的步骤来实现。

1. 取出content-type的内容，并由"; "(注意;后有空格)进行切分。

```
var splitors = contentType.split("; ");
```

获取分割符号(boundary)

2. 遍历分割的字符数组，找出包含`boundary=`的一个字符串，并取出对应的值。

```
function getBoundary(contentType) {
  const boundaries = contentType.split("; ");
  for (let item of boundaries) {
    if (!item) {
      continue;
    }
    if (item.indexOf("boundary=") !== -1) {
      return item.split("=")[1];
    }
  }
}
```

分割分段内容

由于上传的内容是通过`--` + `boundary`实现的，所以我们可以这样切割分段内容。其中data是所有的POST数据。

```
// 注意前面多两个--
var parts = String(data).split("--" + splitor);
```

提取分段内容

1. 获取分段

```
function getParts(boundary, data) {
  const parts = String(data).split("--" + boundary);
  return parts.filter(item => {
    if (!item) {
      return false;
    }
    return true;
  });
}
```

2. 分解分段数据

```
function separateParts(parts, req) {
  for (var i = 0; i < parts.length; i++) {
    var contents = parts[i].split("\r\n\r\n");
    for (var j = 0; j < contents.length; j++) {
      contents[j] = contents[j].replace(/\r\n$/, "");
      contents[j] = contents[j].replace(/^\r\n/, "");
    }
    // processPart(contents[0], contents[1], req);
  }
}
```

分解分段内容

提取分段头

分段头的内容形式：

1. 普通表单字段

```
content-disposition: form-data; name="field1"
```

2. 文件字段

```
Content-disposition: attachment; filename="file1.txt"
```

提取分段头

```
function getHeaderName(header) {
  var headers = header.split("\r\n");
  for (var i = 0; i < headers.length; i++) {
    var subHead = headers[i];
    var subHeaders = subHead.split(": ");
    var names = subHeaders[1].split("; ");
    for (var j = 0; j < names.length; j++) {
      if (names[j].indexOf("name") !== -1) {
        var values = names[j].split("=");
        return values[1];
      }
    }
  }
  return "";
}
```

分解分段内容

2. 提取分段体

```
function processFilePart(header, body, req) {
  const name = getHeaderName(header);
  if (!name) {
    return;
  }
  if (!req.files) {
    req.files = {};
  }
  req.files[name] = body;
}
```

express的文件上传服务器

★ by calidion

搭建express的服务器

```
const express = require("express");

const app = express();

app.use((req, res) => {
  res.send("Hello World!\n");
});

const port = 8081;

app.listen(port, () => {
  console.log("Express Server started at " + port);
});
```

选择文件处理包

multer是express里面最常用的。

```
const multer  = require('multer')
const dirname = "uploads/";
const upload = multer({ dest: dirname });
```

multer的API比较初级，应该说设计的并不是很理想。但是因为是官方推荐，所以我们暂时以multer为蓝本介绍。

multer的设计问题

1. 需要事先指定文件字段名，而不是在使用时指定
2. 事先指定文件数量，并使用不同的函数处理
3. 为一个功能做多个接口
4. 容错性差

所以这里也推荐尝试skipper，相对来讲接口比较友好一些。

接收一个文件提交

```
app.post("/", upload.single("one"), (req, res) => {
  fs.renameSync(req.file.path, dirname +
    req.file.originalname);
  res.end();
});
```

接收多个文件提交

```
app.post("/array", upload.array("array"), (req, res) => {
  req.files.forEach(item => {
    fs.renameSync(item.path, dirname + item.originalname);
  });
  res.end();
});
```

接收多名称，多文件提交

```
app.post("/array", upload.fields([{ name: 'avatar',
  maxCount: 1 }, { name: 'gallery', maxCount: 8 }]),
  (req, res) => {
  req.files.forEach(item => {
    fs.renameSync(item.path, dirname + item.originalname);
  });
  res.end();
});
```