目录

Introduction

访问gitbook地址https://findwisdom.github.io/koa2-note

koa2 快速开始

环境准备

- 因为node.js v7.6.0开始完全支持async/await，不需要加flag，所以node.js环境都要7.6.0以上
- node.js环境 版本v7.6以上
  - 直接安装node.js 7.6：node.js官网地址https://nodejs.org
  - nvm管理多版本node.js：可以用nvm 进行node版本进行管理
    - Mac系统安装nvm https://github.com/creationix/nvm#manual-install
    - windows系统安装nvm https://github.com/coreybutler/nvm-windows
    - Ubuntu系统安装nvm https://github.com/creationix/nvm
- npm 版本3.x以上

快速开始

安装koa2

```
# 初始化package.json
npm init

# 安装koa2
npm install koa
```

hello world 代码

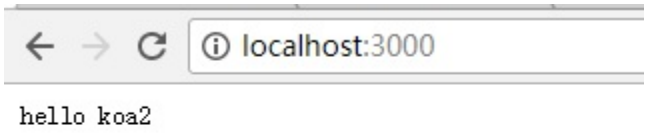
```
const Koa = require('koa')
const app = new Koa()

app.use( async ( ctx ) => {
    ctx.body = 'hello koa2'
})

app.listen(3000)
console.log('[demo] start-quick is starting at port 3000')
```

启动demo

由于koa2是基于async/await操作中间件，目前node.js 7.x的harmony模式下才能使用，所以启动的时的脚本如下：

```
node index.js
```

访问http:localhost:3000，效果如下



async/await使用

快速上手理解

先复制以下这段代码，在粘贴在chrome的控制台console中，按回车键执行

```
function getSyncTime() {
    return new Promise((resolve, reject) => {
        try {
            let startTime = new Date().getTime()
            setTimeout(() => {
        let endTime = new Date().getTime()
        let data = endTime - startTime
        resolve( data )
    }, 500)
} catch ( err ) {
        reject( err )
    }
})
}

async function getSyncData() {
    let time = await getSyncTime()
    let data = `endTime - startTime = ${time}`
    return data
}

async function getData() {
    let data = await getSyncData()
    console.log( data )
}

getData()
```

在chrome的console中执行结果如下

```
> function getSyncTime() {
    return new Promise((resolve, reject) => {
      try {
        let startTime = new Date().getTime()
        setTimeout(() => {
          let endTime = new Date().getTime()
          let data = endTime - startTime
          resolve( data )
        }, 500)
      } catch ( err ) {
        reject( err )
      }
    })
  }

  async function getSyncData() {
    let time = await getSyncTime()
    let data = `endTime - startTime = ${time}`
    return data
  }

  async function getData() {
    let data = await getSyncData()
    console.log( data )
  }

  getData()
< ▼Promise {<pending>}
    ▶__proto__: Promise
      [[PromiseStatus]]: "resolved"
      [[PromiseValue]]: undefined
  endTime - startTime = 500
>
```

从上述例子可以看出 **async/await** 的特点：

- 可以让异步逻辑用同步写法实现
- 最底层的await返回需要是Promise对象
- 可以通过多层 async function 的同步写法代替传统的callback嵌套

koa2简析结构

源码文件

```
├─ lib
│   ├─ application.js
│   ├─ context.js
│   ├─ request.js
│   └─ response.js
└─ package.json
```

这个就是 `GitHub` https://github.com/koajs/koa上开源的koa2源码的源文件结构，核心代码就是lib目录下的四个文件

- application.js 是整个koa2 的入口文件，封装了context，request，response，以及最核心的中间件处理流程。
- context.js 处理应用上下文，里面直接封装部分request.js和response.js的方法
- request.js 处理http请求
- response.js 处理http响应

koa2特性

- 只提供封装好http上下文、请求、响应，以及基于 `async/await` 的中间件容器。
- 利用ES7的 `async/await` 的来处理传统回调嵌套问题和代替koa@1的generator，但是需要在node.js 7.x的harmony模式下才能支持 `async/await` 。
- 中间件只支持 `async/await` 封装的，如果要使用koa@1基于generator中间件，需要通过中间件koa-convert封装一下才能使用。

koa中间件开发和使用

- koa v1和v2中使用到的中间件的开发和使用
- generator 中间件开发在koa v1和v2中使用
- async await 中间件开发和只能在koa v2中使用

---

generator中间件开发

> generator中间件返回的应该是function * () 函数

```
/* ./middleware/logger-generator.js */
function log( ctx ) {
    console.log( ctx.method, ctx.header.host + ctx.url )
}

module.exports = function () {
    return function * ( next ) {

        // 执行中间件的操作
        log( this )

        if ( next ) {
            yield next
        }
    }
}
```

generator中间件在koa@1中的使用

> generator 中间件在koa v1中可以直接use使用

```
const koa = require('koa')  // koa v1
const loggerGenerator  = require('./middleware/logger-generator')
const app = koa()

app.use(loggerGenerator())

app.use(function *( ) {
    this.body = 'hello world!'
```

```
})

app.listen(3000)
console.log('the server is starting at port 3000')
```

generator中间件在koa@2中的使用

> generator 中间件在koa v2中需要用koa-convert封装一下才能使用

```
const Koa = require('koa') // koa v2
const convert = require('koa-convert')
const loggerGenerator  = require('./middleware/logger-generator')
const app = new Koa()

app.use(convert(loggerGenerator()))

app.use(( ctx ) => {
    ctx.body = 'hello world!'
})

app.listen(3000)
console.log('the server is starting at port 3000')
```

---

async中间件开发

```
/* ./middleware/logger-async.js */

function log( ctx ) {
    console.log( ctx.method, ctx.header.host + ctx.url )
}

module.exports = function () {
  return async function ( ctx, next ) {
    log(ctx);
    await next()
  }
}
```

async 中间件在koa@2中使用

> async 中间件只能在 koa v2中使用

```
const Koa = require('koa') // koa v2
const loggerAsync  = require('./middleware/logger-async')
const app = new Koa()

app.use(loggerAsync())

app.use(( ctx ) => {
    ctx.body = 'hello world!'
})

app.listen(3000)
console.log('the server is starting at port 3000')
```

koa2 原生路由实现

简单例子

```
const Koa = require('koa')
const app = new Koa()

app.use( async ( ctx ) => {
    let url = ctx.request.url
    ctx.body = url
})
app.listen(3000)
```

访问 http://localhost:3000/hello/world 页面会输出 /hello/world，也就是说上下文的请求request对象中url之就是当前访问的路径名称，可以根据ctx.request.url 通过一定的判断或者正则匹配就可以定制出所需要的路由。

定制化的路由

demo源码

https://github.com/findwisdom/koa2-note/tree/master/demo/route-simple

源码文件目录

```
.
├── index.js
├── package.json
└── view
    ├── 404.html
    ├── index.html
    └── todo.html
```

demo源码

```
const Koa = require('koa')
const fs = require('fs')
const app = new Koa()
```

```
/**
 * 用Promise封装异步读取文件方法
 * @param  {string} page html文件名称
 * @return {promise}
 */
function render(page) {
    return new Promise((resolve, reject) => {
        let viewUrl = `./view/${page}`
        fs.readFile(viewUrl, "binary", (err, data) => {
            if(err) {
                reject(err)
            } else {
                resolve(data)
            }
        })
    })
}

/**
 * 根据URL获取HTML内容
 * @param  {string} url koa2上下文的url，ctx.url
 * @return {string}     获取HTML文件内容
 */
async function route (url) {
    let view = '404.html'
    switch (url) {
        case '/':
            view = 'index.html'
            break
        case '/index':
            view = 'index.html'
            break
        case '/todo':
            view = 'todo.html'
            break
        case '/404':
            view = '404.html'
            break
        default:
            break
    }
    let html = await render(view)
    return html
}
```

```
app.use(async(ctx) => {
    let url = ctx.request.url
    let html = await route(url)
    ctx.body = html
})

app.listen(3000, () => {
    console.log('[demo] route-simple is starting at port 3000')
})
c'd
```

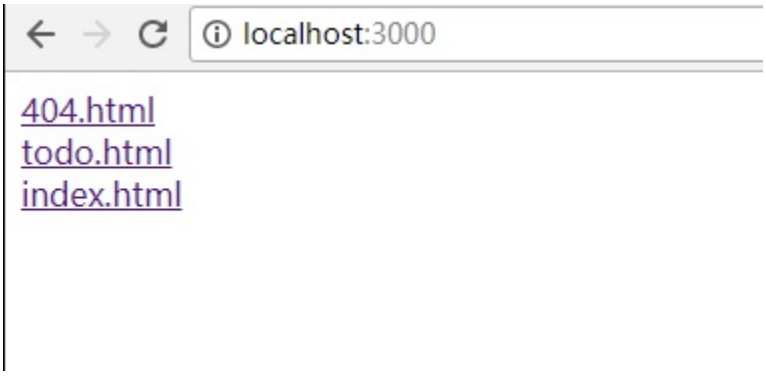
运行**demo**

执行运行脚本

```
node -harmony index.js
```

运行效果如下

访问http://localhost:3000/index



koa-router中间件

如果依靠ctx.request.url去手动处理路由，将会写很多处理代码，这时候就需要对应的路由的中间件对路由进行控制，这里介绍一个比较好用的路由中间件koa-router

安装koa-router中间件

```
# koa2 对应的版本是 7.x
npm install --save koa-router@7
```

快速使用koa-router

demo源码

https://github.com/findwisdom/koa2-note/tree/master/demo/route-use-middleware

```
const Koa = require('koa')
const fs = require('fs')
const app = new Koa()

const Router = require('koa-router')

let home = new Router()

// 子路由1
home.get('/', async( ctx ) => {
    let html = `
        <ul>
          <li><a href="/page/helloworld">/page/helloworld</a></li>
          <li><a href="/page/404">/page/404</a></li>
          <li><a href="/page">/page</a></li>
        </ul>
      `
    ctx.body = html
})

// 子路由2
let page = new Router()
page
    .get('/', async ( ctx ) => {
        ctx.body = 'page!'
```

```
    })
    .get('/404', async ( ctx ) => {
        ctx.body = '404 page!'
    })
    .get('/helloworld', async ( ctx ) => {
        ctx.body = 'helloworld page!'
    })

// 装载所有子路由
let router = new Router()
router.use('/', home.routes(), home.allowedMethods())
router.use('/page', page.routes(), page.allowedMethods())

// 加载路由中间件
app.use(router.routes()).use(router.allowedMethods())

app.listen(3000, () => {
    console.log('[demo] route-use-middleware is starting at port 3000')
})
```

GET请求数据获取

使用方法

在koa中，获取GET请求数据源头是koa中request对象中的query方法或querystring方法，query返回是格式化好的参数对象，querystring返回的是请求字符串，由于ctx对request的API有直接引用的方式，所以获取GET请求数据有两个途径。

- 1.是从上下文中直接获取
  - 请求对象ctx.query，返回如 { a:1, b:2 }
  - 请求字符串 ctx.querystring，返回如 a=1&b=2
- 2.是从上下文的request对象中获取
  - 请求对象ctx.request.query，返回如 { a:1, b:2 }
  - 请求字符串 ctx.request.querystring，返回如 a=1&b=2

举个例子

demo源码

https://github.com/findwisdom/koa2-note/blob/master/demo/request/get.js

例子代码

```
const Koa = require('koa')
const app = new Koa()

app.use( async ( ctx ) => {
    let url = ctx.url

    // 从上下文的request对象中获取
    let request = ctx.request
    let req_query = request.query
    let req_querystring = request.querystring

    // 从上下文中直接获取
    let ctx_query = ctx.query
    let ctx_querystring = ctx.querystring

    ctx.body = {
        url,
```

```
            req_query,
            req_querystring,
            ctx_query,
            ctx_querystring
        }
})

app.listen(3000, () => {
    console.log('[demo] request get is starting at port 3000')
})
```

执行程序

```
node get.js
```

执行后程序后，用chrome访问 http://localhost:3000/page/user?a=1&b=2 会出现以下情况

> 注意：我是用了chrome的json格式化插件才会显示json的格式化

POST请求参数获取

原理

对于POST请求的处理，koa2没有封装获取参数的方法，需要通过解析上下文context中的原生node.js请求对象req，将POST表单数据解析成query string（例如：`a=1&b=2&c=3`），再将query string 解析成JSON格式（例如：`{"a":"1", "b":"2", "c":"3"}`）

> 注意：ctx.request是context经过封装的请求对象，ctx.req是context提供的node.js原生HTTP请求对象，同理ctx.response是context经过封装的响应对象，ctx.res是context提供的node.js原生HTTP请求对象。
>
> 具体koa2 API文档可见
> https://github.com/koajs/koa/blob/master/docs/api/context.md#ctxreq

解析出POST请求上下文中的表单数据

demo源码

https://github.com/findwisdom/koa2-note/blob/master/demo/request/post.js

```
// 解析上下文里node原生请求的POST参数
function parsePostData(ctx) {
    return new Promise((resolve, reject) => {
        try {
            let postdata = "";
        ctx.req.addListener('data', (data) => {
            postdata += data
        })
        ctx.req.addListener("end", function () {
            let parseData = parseQueryStr(postdata)
            resolve(parseData)
        })
    } catch
        (err) {
            reject(err)
        }
    })
}

// 将POST请求参数字符串解析成JSON
function parseQueryStr(queryStr) {
```

```
    let queryData = {}
    let queryStrList = queryStr.split('&')
    console.log(queryStrList)
    for (let [index, queryStr] of queryStrList.entries()) {
        let itemList = queryStr.split('=')
        queryData[itemList[0]] = decodeURIComponent(itemList[1])
    }
    return queryData
}

app.listen(3000, () => {
    console.log('[demo] request post is starting at port 3000')
})
```

举个例子

源码在 /demos/request/post.js中

例子代码

```
const Koa = require('koa')
const app = new Koa()
app.use(async(ctx) => {
    if(ctx.url === '/' && ctx.method === 'GET'){
        // 当GET请求时候返回表单页面
        let html = `
          <h1>koa2 request post demo</h1>
          <form method="POST" action="/">
            <p>userName</p>
            <input name="userName" /><br/>
            <p>nickName</p>
            <input name="nickName" /><br/>
            <p>email</p>
            <input name="email" /><br/>
            <button type="submit">submit</button>
          </form>
        `
    ctx.body = html
} else if (ctx.url === '/' && ctx.method === 'POST') {
    // 当POST请求的时候，解析POST表单里的数据，并显示出来
    let postData = await parsePostData(ctx)
    ctx.body = postData
} else {
```

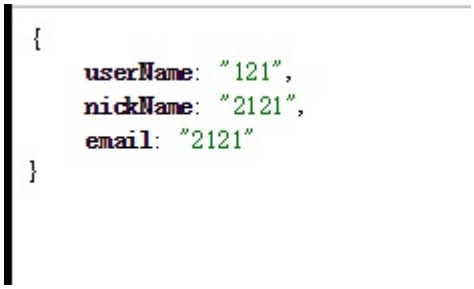
```
    // 其他请求显示404
    ctx.body = '<h1>404! ! !  o(╯□╰)o</h1>'
    }
})

// 解析上下文里node原生请求的POST参数
function parsePostData(ctx) {
    return new Promise((resolve, reject) => {
        try {
            let postdata = "";
        ctx.req.addListener('data', (data) => {
            postdata += data
        })
        ctx.req.addListener("end", function () {
            let parseData = parseQueryStr(postdata)
            resolve(parseData)
        })
    } catch
        (err) {
            reject(err)
        }
    })
}

// 将POST请求参数字符串解析成JSON
function parseQueryStr(queryStr) {
    let queryData = {}
    let queryStrList = queryStr.split('&')
    console.log(queryStrList)
    for (let [index, queryStr] of queryStrList.entries()) {
        let itemList = queryStr.split('=')
        queryData[itemList[0]] = decodeURIComponent(itemList[1])
    }
    return queryData
}

app.listen(3000, () => {
    console.log('[demo] request post is starting at port 3000')
})
```

启动例子

```
node post.js
```

访问页面

提交表单发起**POST**请求结果显示



koa-bodyparser中间件

原理

对于POST请求的处理，koa-bodyparser中间件可以把koa2上下文的formData数据解析到ctx.request.body中

安装koa2版本的koa-bodyparser@3中间件

```
npm install --save koa-bodyparser@3
```

举个例子

例子代码

demo源码

https://github.com/findwisdom/koa2-note/blob/master/demo/request/post-middleware.js

```
const Koa = require('koa')
const app = new Koa()
const bodyParser = require('koa-bodyparser')

// 使用ctx.body解析中间件
app.use(bodyParser())

app.use( async ( ctx ) => {

    if ( ctx.url === '/' && ctx.method === 'GET' ) {
    // 当GET请求时候返回表单页面
    let html = `
      <h1>koa2 request post demo</h1>
      <form method="POST" action="/">
        <p>userName</p>
        <input name="userName" /><br/>
        <p>nickName</p>
        <input name="nickName" /><br/>
        <p>email</p>
        <input name="email" /><br/>
        <button type="submit">submit</button>
```

```
        </form>
    `
    ctx.body = html
} else if ( ctx.url === '/' && ctx.method === 'POST' ) {
    // 当POST请求的时候，中间件koa-bodyparser解析POST表单里的数据，并显示出来
    let postData = ctx.request.body
    ctx.body = postData
} else {
    // 其他请求显示404
    ctx.body = '<h1>404! ! !  o(╯□╰)o</h1>'
}
})

app.listen(3000, () => {
    console.log('[demo] request post is starting at port 3000')
})
```

启动例子

```
node post-middleware.js
```

访问页面

提交表单发起**POST**请求结果显示

```
{
    userName: "121",
    nickName: "2121",
    email: "2121"
}
```

原生**koa2**实现静态资源服务器

前言

一个http请求访问web服务静态资源，一般响应结果有三种情况

- 访问文本，例如js，css，png，jpg，gif
- 访问静态目录
- 找不到资源，抛出404错误

原生**koa2** 静态资源服务器例子

demo源码

https://github.com/findwisdom/koa2-note/tree/master/demo/static-server

代码目录

```
├── static # 静态资源目录
│   ├── css/
│   ├── image/
│   ├── js/
│   └── index.html
├── util # 工具代码
│   ├── content.js # 读取请求内容
│   ├── dir.js # 读取目录内容
│   ├── file.js # 读取文件内容
│   ├── mimes.js # 文件类型列表
│   └── walk.js # 遍历目录内容
└── index.js # 启动入口文件
```

代码解析

index.js

```
const Koa = require('koa')
const path = require('path')
const content = require('./util/content')
const mimes = require('./util/mimes')
```

```
const app = new Koa()

// 静态资源目录对于相对入口文件index.js的路径
const staticPath = './static'

// 解析资源类型
function parseMime( url ) {
    let extName = path.extname( url )
    extName = extName ?  extName.slice(1) : 'unknown'
    return  mimes[ extName ]
}

app.use( async ( ctx ) => {
    // 静态资源目录在本地的绝对路径
    let fullStaticPath = path.join(__dirname, staticPath)

    // 获取静态资源内容，有可能是文件内容，目录，或404
    let _content = await content( ctx, fullStaticPath )

    // 解析请求内容的类型
    let _mime = parseMime( ctx.url )

    // 如果有对应的文件类型，就配置上下文的类型
    if ( _mime ) {
        ctx.type = _mime
    }

    // 输出静态资源内容
    if ( _mime && _mime.indexOf('image/') >= 0 ) {
    // 如果是图片，则用node原生res，输出二进制数据
    ctx.res.writeHead(200)
    ctx.res.write(_content, 'binary')
    ctx.res.end()
} else {
    // 其他则输出文本
    ctx.body = _content
}


})

app.listen(3000, () => {
    console.log('[demo] static-server is starting at port 3000')
})
```

util/content.js

```
const path = require('path')
const fs = require('fs')

// 封装读取目录内容方法
const dir = require('./dir')

// 封装读取文件内容方法
const file = require('./file')


/**
 * 获取静态资源内容
 * @param  {object} ctx koa上下文
 * @param  {string} 静态资源目录在本地的绝对路径
 * @return  {string} 请求获取到的本地内容
 */
async function content( ctx, fullStaticPath ) {

    // 封装请求资源的完绝对径
    let reqPath = path.join(fullStaticPath, ctx.url)

    // 判断请求路径是否为存在目录或者文件
    let exist = fs.existsSync( reqPath )

    // 返回请求内容， 默认为空
    let content = ''

    if( !exist ) {
        //如果请求路径不存在，返回404
        content = '404 Not Found! o(╯□╰)o! '
    } else {
        //判断访问地址是文件夹还是文件
        let stat = fs.statSync( reqPath )

        if( stat.isDirectory() ) {
            //如果为目录，则渲读取目录内容
            content = dir( ctx.url, reqPath )

        } else {
            // 如果请求为文件，则读取文件内容
            content = file( reqPath )
        }
    }
```

```
    return content
}

module.exports = content
```

util/dir.js

```
const url = require('url')
const fs = require('fs')
const path = require('path')

// 遍历读取目录内容方法
const walk = require('./walk')

/**
 * 封装目录内容
 * @param  {string} url 当前请求的上下文中的url，即ctx.url
 * @param  {string} reqPath 请求静态资源的完整本地路径
 * @return {string} 返回目录内容，封装成HTML
 */
function dir ( url, reqPath ) {

    // 遍历读取当前目录下的文件、子目录
    let contentList = walk( reqPath )

    let html = `<ul>`
    for ( let [ index, item ] of contentList.entries() ) {
        html = `${html}<li><a href="${url === '/' ? '' : url}/${item}">${item}</
a></li>`
    }
    html = `${html}</ul>`

    return html
}

module.exports = dir
```

util/file.js

```
const fs = require('fs')

/**
 * 读取文件方法
 * @param  {string} 文件本地的绝对路径
```

```
 * @return {string|binary}
 */
function file ( filePath ) {

    let content = fs.readFileSync(filePath, 'binary' )
    return content
}

module.exports = file
```

util/walk.js

```
const fs = require('fs')
const mimes = require('./mimes')

/**
 * 遍历读取目录内容（子目录，文件名）
 * @param  {string} reqPath 请求资源的绝对路径
 * @return {array} 目录内容列表
 */
function walk( reqPath ){

    let files = fs.readdirSync( reqPath );

    let dirList = [], fileList = [];
    for( let i=0, len=files.length; i<len; i++ ) {
        let item = files[i];
        let itemArr = item.split("\.");
        let itemMime = ( itemArr.length > 1 ) ? itemArr[ itemArr.length - 1 ] :
"undefined";

        if( typeof mimes[ itemMime ] === "undefined" ) {
            dirList.push( files[i] );
        } else {
            fileList.push( files[i] );
        }
    }


    let result = dirList.concat( fileList );

    return result;
};

module.exports = walk;
```

util/mime.js

```
let mimes = {
    'css': 'text/css',
    'less': 'text/css',
    'gif': 'image/gif',
    'html': 'text/html',
    'ico': 'image/x-icon',
    'jpeg': 'image/jpeg',
    'jpg': 'image/jpeg',
    'js': 'text/javascript',
    'json': 'application/json',
    'pdf': 'application/pdf',
    'png': 'image/png',
    'svg': 'image/svg+xml',
    'swf': 'application/x-shockwave-flash',
    'tiff': 'image/tiff',
    'txt': 'text/plain',
    'wav': 'audio/x-wav',
    'wma': 'audio/x-ms-wma',
    'wmv': 'video/x-ms-wmv',
    'xml': 'text/xml'
}

module.exports = mimes
```

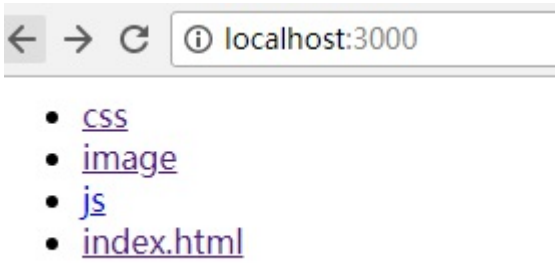
运行效果

启动服务

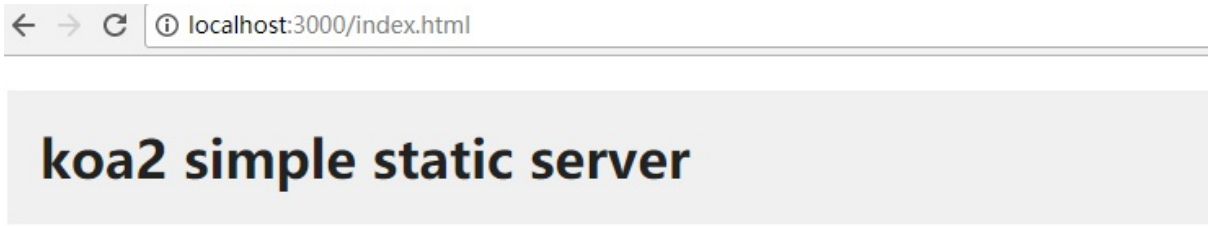
```
node index.js
```

效果

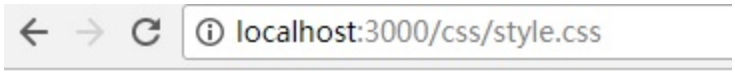
访问**http://localhost:3000**



访问**http://localhost:3000/index.html**



访问**http://localhost:3000/css/style.css**




```
.h1 {
    padding: 20px;
    color: #222;
    background: #f0f0f0;
}
```

koa-static中间件使用

使用例子

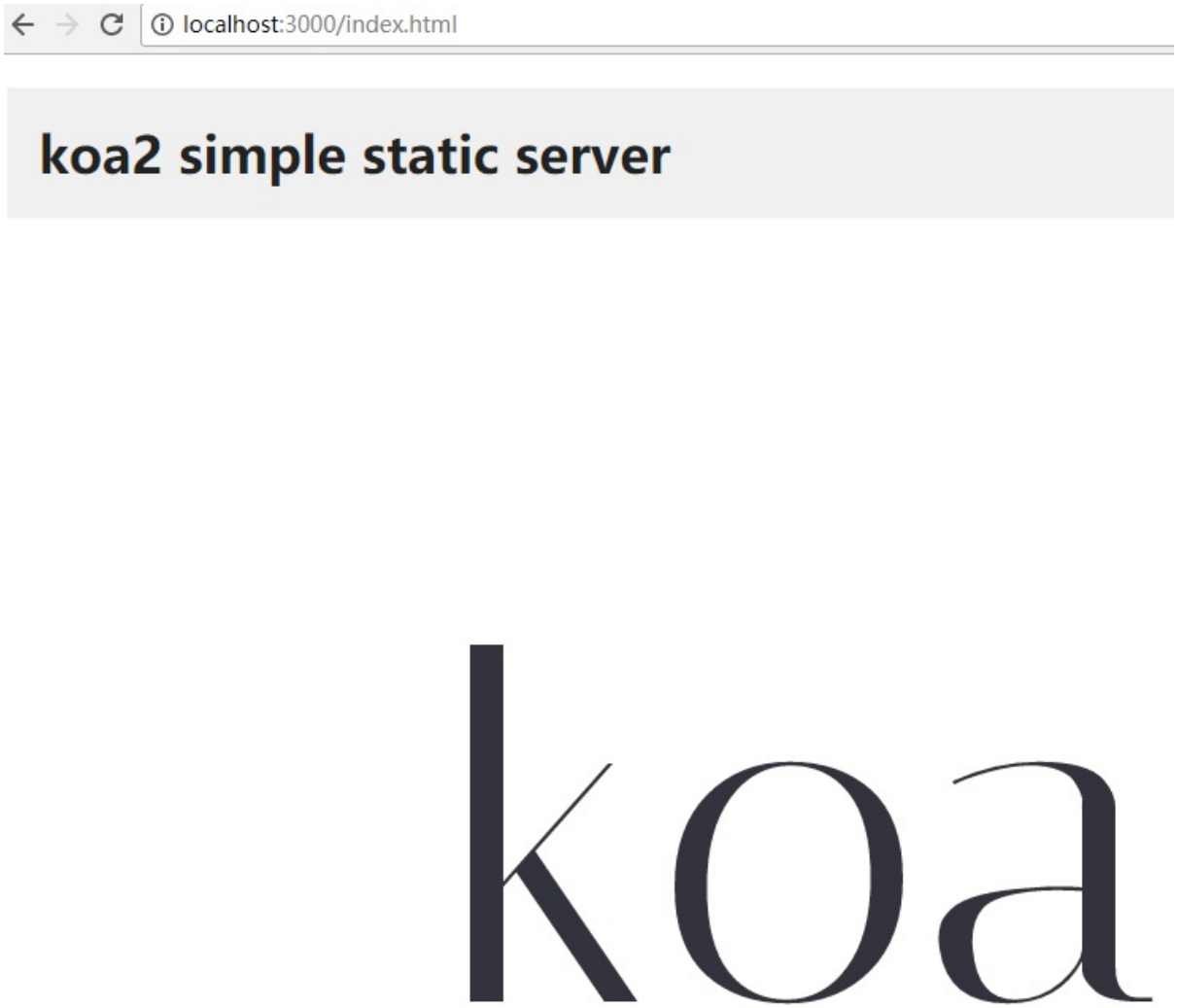
demo源码

https://github.com/findwisdom/koa2-note/tree/master/demo/static-use-middleware

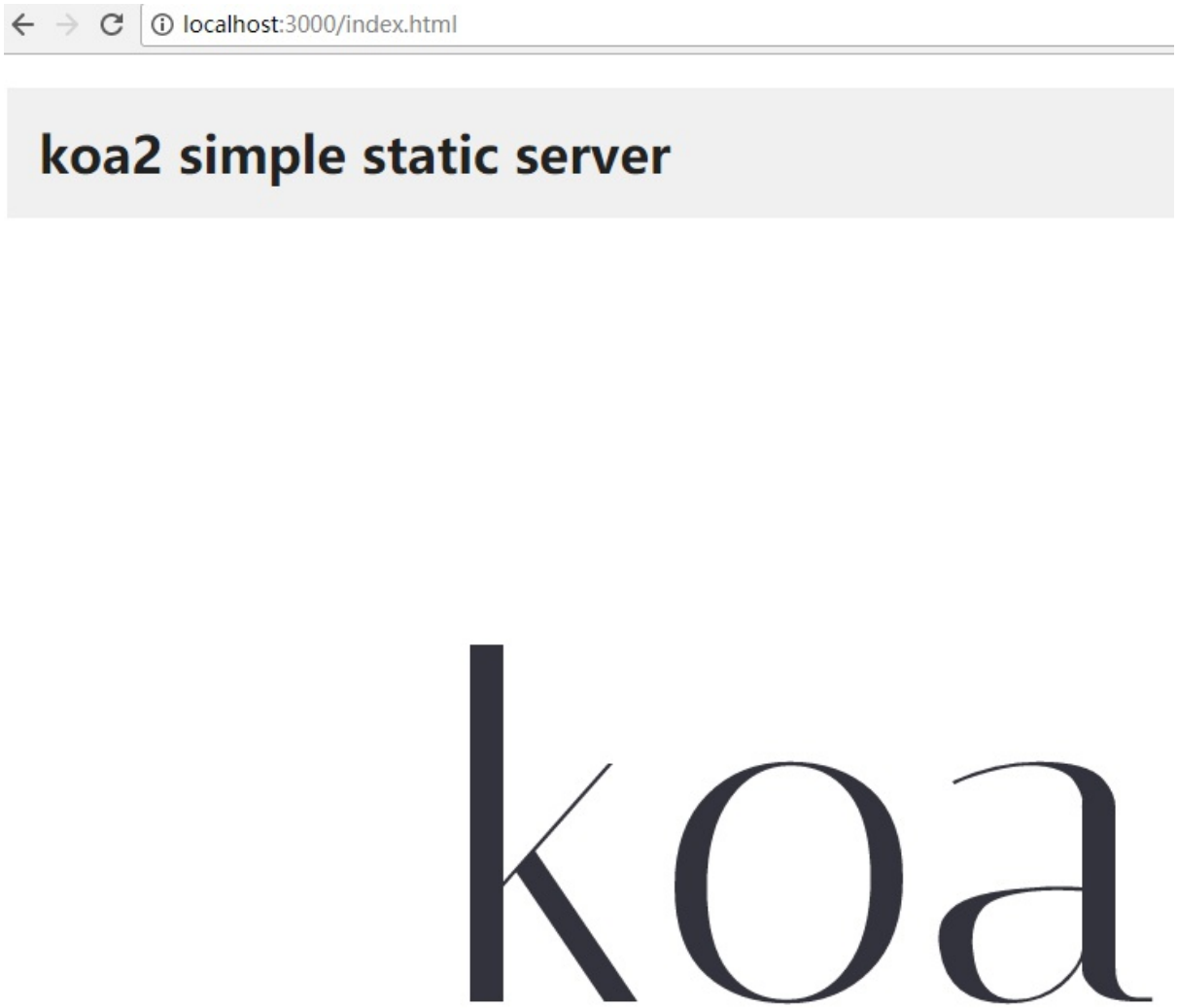
```
const Koa = require('koa')
const path = require('path')
const static = require('koa-static')

const app = new Koa()

// 静态资源目录对于相对入口文件index.js的路径
const staticPath = './static'

app.use(static(
    path.join( __dirname,  staticPath)
))


app.use( async ( ctx ) => {
  ctx.body = 'hello world'
})

app.listen(3000, () => {
    console.log('[demo] static-use-middleware is starting at port 3000')
})
```

效果

访问**http://localhost:3000**



访问**http://localhost:3000/index.html**



访问**http://localhost:3000/css/style.css**


```
.h1 {
    padding: 20px;
    color: #222;
    background: #f0f0f0;
}
```

koa2使用cookie

使用方法

koa提供了从上下文直接读取、写入cookie的方法

- ctx.cookies.get(name, [options]) 读取上下文请求中的cookie
- ctx.cookies.set(name, value, [options]) 在上下文中写入cookie

koa2 中操作的cookies是使用了npm的cookies模块，源码在https://github.com/pillarjs/cookies，所以在读写cookie的使用参数与该模块的使用一致。

例子代码

```
const Koa = require('koa')
const app = new Koa()

app.use( async ( ctx ) => {

    if ( ctx.url === '/index' ) {
    // 设置cookie
        ctx.cookies.set(
            'cid',
            'hello world',
            {
                domain: 'localhost',  // 写cookie所在的域名
                path: '/index',       // 写cookie所在的路径
                maxAge: 10 * 60 * 1000, // cookie有效时长
                expires: new Date('2017-02-15'),  // cookie失效时间
                httpOnly: false,  // 是否只用于http请求中获取
                overwrite: false  // 是否允许重写
            }
        )
            ctx.body = 'cookie is ok'
        } else {
            ctx.body = 'hello world'
    }

})

app.listen(3000, () => {
    console.log('[demo] cookie is starting at port 3000')
```

```
})
```

运行例子

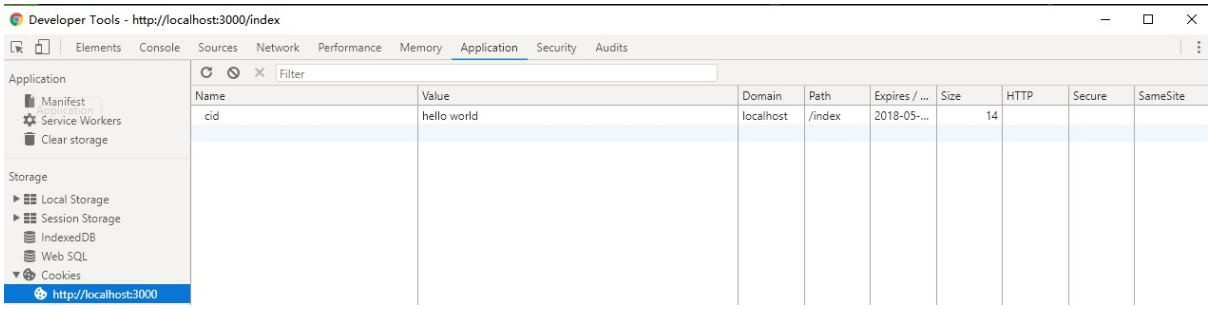
执行脚本

```
node index.js
```

运行结果

访问http://localhost:3000/index

- 可以在控制台的cookie列表中中看到写在页面上的cookie
- 在控制台的console中使用document.cookie可以打印出在页面的所有cookie（需要是httpOnly设置false才能显示）



busboy模块

快速开始

安装

```
npm install --save busboy
```

模块简介

busboy 模块是用来解析POST请求，node原生req中的文件流。

开始使用

```
const inspect = require('util').inspect
const path = require('path')
const fs = require('fs')
const Busboy = require('busboy')

// req 为node原生请求
const busboy = new Busboy({ headers: req.headers })

// ...

// 监听文件解析事件
busboy.on('file', function(fieldname, file, filename, encoding, mimetype) {
    console.log(`File [${fieldname}]: filename: ${filename}`)


    // 文件保存到特定路径
    file.pipe(fs.createWriteStream('./upload'))

    // 开始解析文件流
    file.on('data', function(data) {
        console.log(`File [${fieldname}] got ${data.length} bytes`)
    })

    // 解析文件结束
    file.on('end', function() {
        console.log(`File [${fieldname}] Finished`)
```

```
    })
})

// 监听请求中的字段
busboy.on('field', function(fieldname, val, fieldnameTruncated, valTruncated) {
    console.log(`Field [${fieldname}]: value: ${inspect(val)}`)
})

// 监听结束事件
busboy.on('finish', function() {
    console.log('Done parsing form!')
    res.writeHead(303, { Connection: 'close', Location: '/' })
    res.end()
})
req.pipe(busboy)
```

更多模块信息

更多详细API可以访问npm官方文档 https://www.npmjs.com/package/busboy

上传文件简单实现

依赖模块

安装依赖

```
npm install --save busboy
```

- busboy 是用来解析出请求中文件流

例子源码

demo源码

https://github.com/findwisdom/koa2-note/blob/master/demo/upload/

封装上传文件到写入服务的方法

```
const inspect = require('util').inspect
const path = require('path')
const fs = require('fs')
const Busboy = require('busboy')

/**
 * 同步创建文件目录
 * @param  {string} dirname 目录绝对地址
 * @return {boolean}        创建目录结果
 */
function mkdirsSync( dirname ) {
    if (fs.existsSync( dirname )) {
        return true
    } else {
        if (mkdirsSync( path.dirname(dirname)) ) {
            fs.mkdirSync( dirname )
            return true
        }
    }
}
```

```
/**
 * 获取上传文件的后缀名
 * @param  {string} fileName 获取上传文件的后缀名
 * @return {string}          文件后缀名
 */
function getSuffixName( fileName ) {
    let nameList = fileName.split('.')
    return nameList[nameList.length - 1]
}

/**
 * 上传文件
 * @param  {object} ctx     koa上下文
 * @param  {object} options 文件上传参数 fileType文件类型， path文件存放路径
 * @return {promise}
 */
function uploadFile( ctx, options) {
    let req = ctx.req
    let res = ctx.res
    let busboy = new Busboy({headers: req.headers})

    // 获取类型
    let fileType = options.fileType || 'common'
    let filePath = path.join( options.path,  fileType)
    let mkdirResult = mkdirsSync( filePath )

    return new Promise((resolve, reject) => {
        console.log('文件上传中...')
    let result = {
        success: false,
        formData: {},
    }

    // 解析请求文件事件
    busboy.on('file', function(fieldname, file, filename, encoding, mimetype) {
        let fileName = Math.random().toString(16).substr(2) + '.' + getSuffixNam
e(filename)
        let _uploadFilePath = path.join( filePath, fileName )
        let saveTo = path.join(_uploadFilePath)

        // 文件保存到制定路径
        file.pipe(fs.createWriteStream(saveTo))
        // 文件写入事件结束
        file.on('data', function(data) {
            console.log(`File [${fieldname}] got ${data.length} bytes`)
        })
```

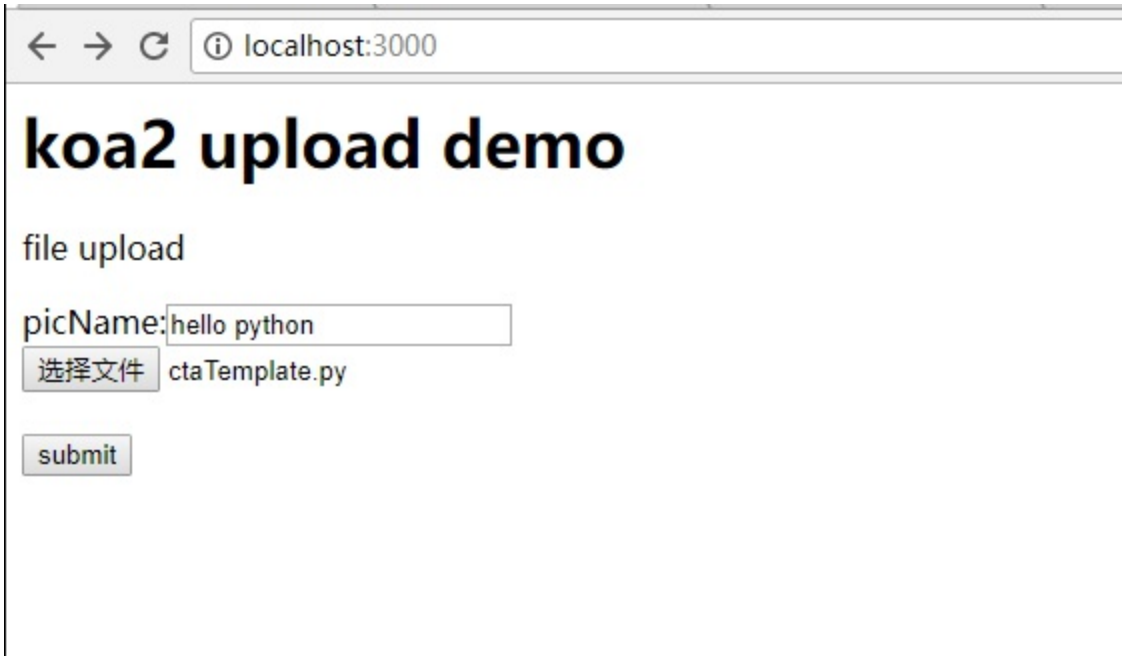
```
        // 文件写入事件结束
        file.on('end', function() {
            result.success = true
            result.message = '文件上传成功'
            console.log('文件上传成功! ')
        })
    })

    // 解析表单中其他字段信息
    busboy.on('field', function(fieldname, val, fieldnameTruncated, valTruncated
, encoding, mimetype) {
        console.log('表单字段数据 [' + fieldname + ']: value: ' + inspect(val));
        result.formData[fieldname] = inspect(val);
    });

    // 解析结束事件
    busboy.on('finish', function( ) {
        console.log('文件上结束')
        resolve(result)
    })

    // 解析错误事件
    busboy.on('error', function(err) {
        console.log('文件上出错')
        reject(result)
    })

    req.pipe(busboy)
})

}


module.exports =  {
    uploadFile
}
```

入口文件

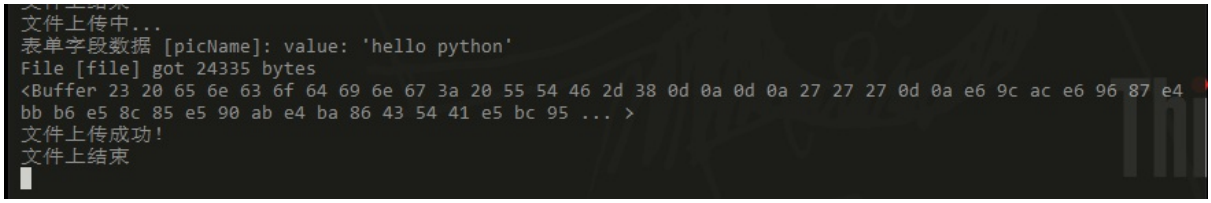
```
const Koa = require('koa')
const path = require('path')
const app = new Koa()
```

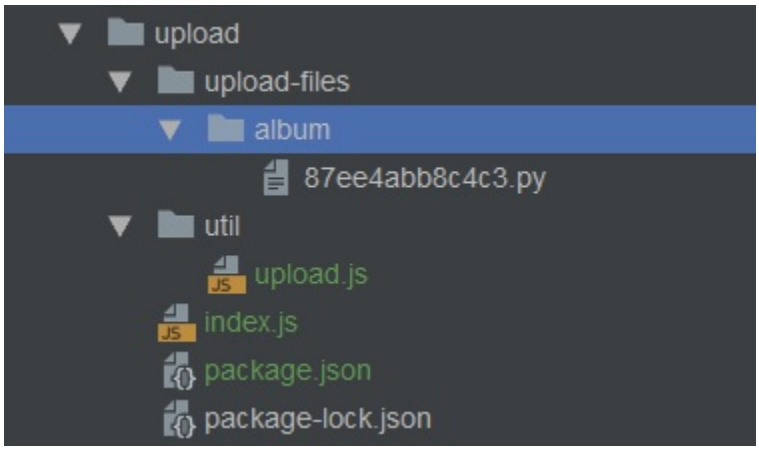
```
const { uploadFile } = require('./util/upload')

app.use( async ( ctx ) => {

    if ( ctx.url === '/' && ctx.method === 'GET' ) {
    // 当GET请求时候返回表单页面
    let html = `
      <h1>koa2 upload demo</h1>
      <form method="POST" action="/upload.json" enctype="multipart/form-data">
        <p>file upload</p>
        <span>picName:</span><input name="picName" type="text" /><br/>
        <input name="file" type="file" /><br/><br/>
        <button type="submit">submit</button>
      </form>
    `
    ctx.body = html

} else if ( ctx.url === '/upload.json' && ctx.method === 'POST' ) {
    // 上传文件请求处理
    let result = { success: false }
    let serverFilePath = path.join( __dirname, 'upload-files' )

    // 上传文件事件
    result = await uploadFile( ctx, {
        fileType: 'album',
        path: serverFilePath
    })

    ctx.body = result
} else {
    // 其他请求显示404
    ctx.body = '<h1>404! ! !  o(╯□╰)o</h1>'
}
})

app.listen(3000, () => {
    console.log('[demo] upload-simple is starting at port 3000')
})
```

运行结果


localhost:3000
koa2 upload demo
file upload
picName:hello python
选择文件
ctaTemplate.py
submit


localhost:3000/upload.json
{
success: true,
formData: {
picName: "'hello python'"
},
message: "文件上传成功"
}


文件上传中...
表单字段数据 [picName]: value: 'hello python'
File [file] got 24335 bytes
<Buffer 23 20 65 6e 63 6f 64 69 6e 67 3a 20 55 54 46 2d 38 0d 0a 0d 0a 27 27 27 0d 0a e6 9c ac e6 96 87 e4
bb b6 e5 8c 85 e5 90 ab e4 ba 86 43 54 41 e5 bc 95 ... >
文件上传成功!
文件上结束


upload
upload-files
album
87ee4abb8c4c3.py
util
upload.js
index.js
package.json
package-lock.json

异步上传图片实现

快速上手

demo 地址

https://github.com/findwisdom/koa2-note/tree/master/demo/upload-async

源码理解

demo源码目录

```
.
├── index.js # 后端启动文件
├── node_modules
├── package.json
├── static # 静态资源目录
│   ├── image # 异步上传图片存储目录
│   └── js
│       └── index.js # 上传图片前端js操作
├── util
│   └── upload.js # 后端处理图片流操作
└── view
    └── index.ejs # ejs后端渲染模板
```

后端代码

入口文件 demo/upload-async/index.js

```
const Koa = require('koa')
const views = require('koa-views')
const path = require('path')
const static = require('koa-static')
const { uploadFile } = require('./util/upload')

const app = new Koa()


/**
```

```
 * 使用第三方中间件 start
 */
app.use(views(path.join(__dirname, './view'), {
    extension: 'ejs'
}))

// 静态资源目录对于相对入口文件index.js的路径
const staticPath = './static'

app.use(static(
    path.join( __dirname,  staticPath)
))
/**
 * 使用第三方中间件 end
 */


app.use( async ( ctx ) => {
    if ( ctx.method === 'GET' ) {
    let title = 'upload pic async'
    await ctx.render('index', {
        title,
    })
} else if ( ctx.url === '/api/picture/upload.json' && ctx.method === 'POST' ) {
    // 上传文件请求处理
    let result = { success: false }
    let serverFilePath = path.join( __dirname, 'static/image' )

    // 上传文件事件
    result = await uploadFile( ctx, {
        fileType: 'album',
        path: serverFilePath
    })
    ctx.body = result
} else {
    // 其他请求显示404
    ctx.body = '<h1>404! ! !  o(╯□╰)o</h1>'
}

})


app.listen(3000, () => {
    console.log('[demo] upload-pic-async is starting at port 3000')
})
```

后端上传图片流写操作 入口文件 demo/upload-async/util/upload.js

```
const inspect = require('util').inspect
const path = require('path')
const os = require('os')
const fs = require('fs')
const Busboy = require('busboy')

/**
 * 同步创建文件目录
 * @param  {string} dirname 目录绝对地址
 * @return {boolean}        创建目录结果
 */
function mkdirsSync( dirname ) {
    if (fs.existsSync( dirname )) {
        return true
    } else {
        if (mkdirsSync( path.dirname(dirname)) ) {
            fs.mkdirSync( dirname )
            return true
        }
    }
}

/**
 * 获取上传文件的后缀名
 * @param  {string} fileName 获取上传文件的后缀名
 * @return {string}          文件后缀名
 */
function getSuffixName( fileName ) {
    let nameList = fileName.split('.')
    return nameList[nameList.length - 1]
}

/**
 * 上传文件
 * @param  {object} ctx     koa上下文
 * @param  {object} options 文件上传参数 fileType文件类型， path文件存放路径
 * @return {promise}
 */
function uploadFile( ctx, options) {
    let req = ctx.req
    let res = ctx.res
    let busboy = new Busboy({headers: req.headers})
```

```
    // 获取类型
    let fileType = options.fileType || 'common'
    let filePath = path.join( options.path,  fileType)
    let mkdirResult = mkdirsSync( filePath )

    return new Promise((resolve, reject) => {
        console.log('文件上传中...')
    let result = {
        success: false,
        message: '',
        data: null
    }

    // 解析请求文件事件
    busboy.on('file', function(fieldname, file, filename, encoding, mimetype) {
        let fileName = Math.random().toString(16).substr(2) + '.' + getSuffixNam
e(filename)
        let _uploadFilePath = path.join( filePath, fileName )
        let saveTo = path.join(_uploadFilePath)

        // 文件保存到制定路径
        file.pipe(fs.createWriteStream(saveTo))

        // 文件写入事件结束
        file.on('end', function() {
            result.success = true
            result.message = '文件上传成功'
            result.data = {
                pictureUrl: `//${ctx.host}/image/${fileType}/${fileName}`
            }
            console.log('文件上传成功！')
            resolve(result)
        })
    })

    // 解析结束事件
    busboy.on('finish', function( ) {
        console.log('文件上结束')
        resolve(result)
    })

    // 解析错误事件
    busboy.on('error', function(err) {
        console.log('文件上出错')
        reject(result)
    })
```

```
    req.pipe(busboy)
})

}


module.exports =  {
    uploadFile
}
```

前端代码

```
<!DOCTYPE html>
<html>
    <head>
        <title><%= title %></title>
        <style type="text/css">
            .btn {
                width: 100px;
                height: 40px;
            }

            .preview-picture {
                width: 300px;
                min-height: 300px;
                border: 1px #f0f0f0 solid
            }
        </style>
    </head>
    <body>
        <button class="btn" id="J_UploadPictureBtn">上传图片</button>
        <hr/>
        <p>上传进度<span id="J_UploadProgress">0</span>%</p>
        <p>上传结果图片</p>
        <div id="J_PicturePreview" class="preview-picture"></div>
        <script src="/js/index.js"></script>
    </body>
</html>
```

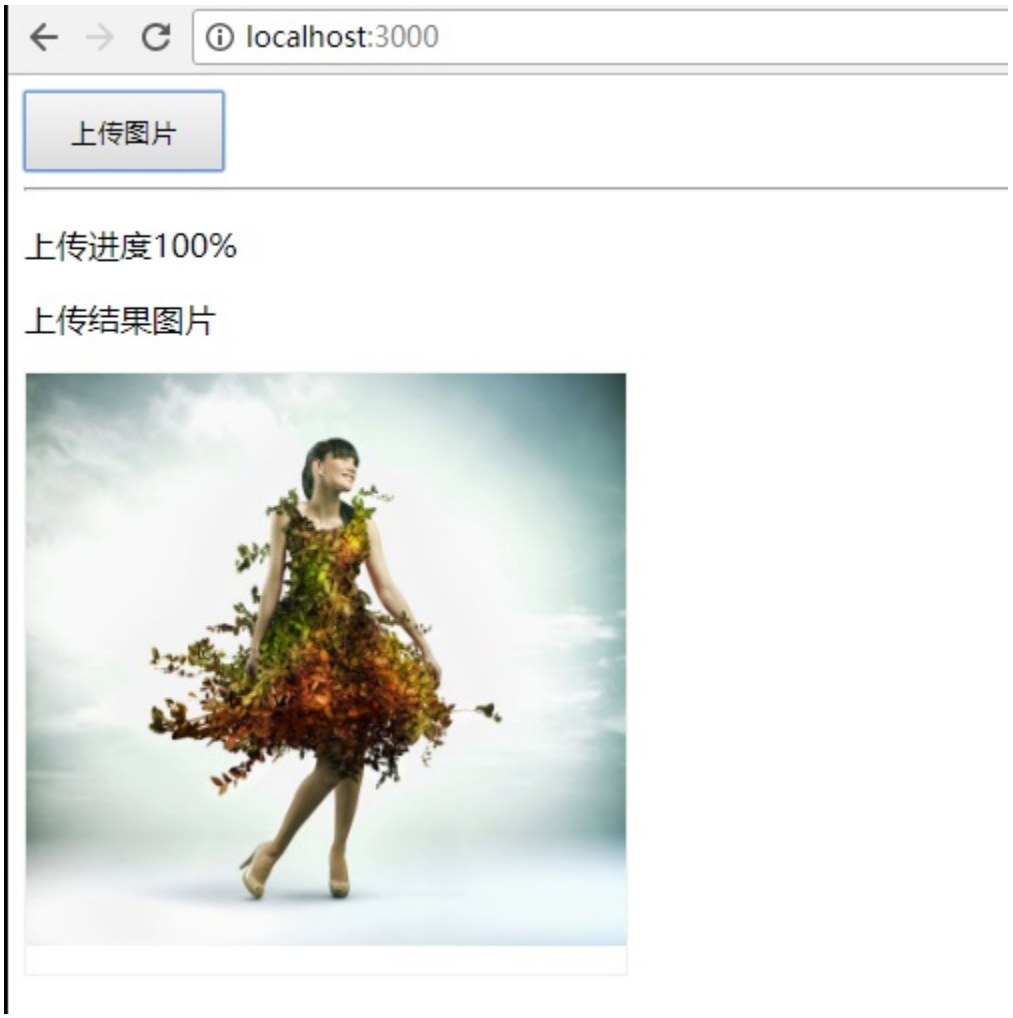
上传操作代码

```
(function(){

    let btn = document.getElementById('J_UploadPictureBtn')
    let progressElem = document.getElementById('J_UploadProgress')
    let previewElem = document.getElementById('J_PicturePreview')
    btn.addEventListener('click', function(){
        uploadAction({
            success: function( result ) {
                console.log( result )
                if ( result && result.success && result.data && result.data.pict
ureUrl ) {
                    previewElem.innerHTML = '<img src="'+ result.data.pictureUrl
 +'" style="max-width: 100%">'
                }
            },
            progress: function( data ) {
                if ( data && data * 1 > 0 ) {
                    progressElem.innerText = data
                }
            }
        })
    })


    /**
     * 类型判断
     * @type {Object}
     */
    let UtilType = {
        isPrototype: function( data ) {
            return Object.prototype.toString.call(data).toLowerCase();
        },

        isJSON: function( data ) {
            return this.isPrototype( data ) === '[object object]';
        },

        isFunction: function( data ) {
            return this.isPrototype( data ) === '[object function]';
        }
    }

    /**
     * form表单上传请求事件
```

```
 * @param  {object} options 请求参数
 */
function requestEvent( options ) {
    try {
        let formData = options.formData
        let xhr = new XMLHttpRequest()
        xhr.onreadystatechange = function() {

            if ( xhr.readyState === 4 && xhr.status === 200 ) {
                options.success(JSON.parse(xhr.responseText))
            }
        }

        xhr.upload.onprogress = function(evt) {
            let loaded = evt.loaded
            let tot = evt.total
            let per = Math.floor(100 * loaded / tot)
            options.progress(per)
        }
        xhr.open('post', '/api/picture/upload.json')
        xhr.send(formData)
    } catch ( err ) {
        options.fail(err)
    }
}

/**
 * 上传事件
 * @param  {object} options 上传参数
 */
function uploadEvent ( options ){
    let file
    let formData = new FormData()
    let input = document.createElement('input')
    input.setAttribute('type', 'file')
    input.setAttribute('name', 'files')

    input.click()
    input.onchange = function () {
        file = input.files[0]
        formData.append('files', file)

        requestEvent({
            formData,
            success: options.success,
            fail: options.fail,
```

```
                progress: options.progress
            })
        }

    }

    /**
     * 上传操作
     * @param  {object} options 上传参数
     */
    function uploadAction( options ) {
        if ( !UtilType.isJSON( options ) ) {
            console.log( 'upload options is null' )
            return
        }
        let _options = {}
        _options.success = UtilType.isFunction(options.success) ? options.succes
s : function() {}
        _options.fail = UtilType.isFunction(options.fail) ? options.fail : funct
ion() {}
        _options.progress = UtilType.isFunction(options.progress) ? options.prog
ress : function() {}

        uploadEvent(_options)
    }


})()
```

运行效果


localhost:3000
上传图片
上传进度100%
上传结果图片