目录

[app 4](#_Toc531120261)

[app.listen 4](#_Toc531120262)

[app.callback 4](#_Toc531120263)

[app.use(function) 4](#_Toc531120264)

[app.keys 4](#_Toc531120265)

[app.context 4](#_Toc531120266)

[context对象 5](#_Toc531120267)

[概览 5](#_Toc531120268)

[req/ request 6](#_Toc531120269)

[res/response 6](#_Toc531120270)

[state 6](#_Toc531120271)

[app 6](#_Toc531120272)

[.cookies.get/set 6](#_Toc531120273)

[throw([status],[mes],[properties]) 6](#_Toc531120274)

[assert() 6](#_Toc531120275)

[respond 6](#_Toc531120276)

[request别名 6](#_Toc531120277)

[response别名 7](#_Toc531120278)

[request 8](#_Toc531120279)

[头，主体相关 8](#_Toc531120280)

[header/header=/headers/headers= 8](#_Toc531120281)

[method/method= 9](#_Toc531120282)

[length 9](#_Toc531120283)

[type 9](#_Toc531120284)

[charset 9](#_Toc531120285)

[fresh/stale 9](#_Toc531120286)

[is(types 9](#_Toc531120287)

[accepts四个 9](#_Toc531120288)

[路径相关 9](#_Toc531120289)

[url/url=/originalUrl/origin 9](#_Toc531120290)

[href 10](#_Toc531120291)

[path/path= 10](#_Toc531120292)

[querystring/querystring= 10](#_Toc531120293)

[search/search= 10](#_Toc531120294)

[query/query= 10](#_Toc531120295)

[host/hostname 10](#_Toc531120296)

[其他 11](#_Toc531120297)

[URL 11](#_Toc531120298)

[ips 11](#_Toc531120299)

[idempotent 11](#_Toc531120300)

[socket 11](#_Toc531120301)

[get 11](#_Toc531120302)

[response 11](#_Toc531120303)

[头相关 11](#_Toc531120304)

[header/headers 11](#_Toc531120305)

[length/length= 11](#_Toc531120306)

[String 11](#_Toc531120307)

[Buffer 11](#_Toc531120308)

[Stream 12](#_Toc531120309)

[Object 12](#_Toc531120310)

[get(field)/set(field,value)/append(field,value)remove(field) 12](#_Toc531120311)

[type= 12](#_Toc531120312)

[is 12](#_Toc531120313)

[attachment() 12](#_Toc531120314)

[headerSent 12](#_Toc531120315)

[lastModified/lastModified= 12](#_Toc531120316)

[etag 12](#_Toc531120317)

[vary() 12](#_Toc531120318)

[flushHeaders() 12](#_Toc531120319)

[状态码/状态消息 12](#_Toc531120320)

[\*status/status= 13](#_Toc531120321)

[message/message= 13](#_Toc531120322)

[响应主体 13](#_Toc531120323)

[\*body 13](#_Toc531120324)

[重定向 13](#_Toc531120325)

[redirect(url,[alt]) 13](#_Toc531120326)

[其他 13](#_Toc531120327)

[socket 13](#_Toc531120328)

[插件 13](#_Toc531120329)

[koa-router 13](#_Toc531120330)

[new Router 13](#_Toc531120331)

[router.get|put|post|patch|delete|del|all 14](#_Toc531120332)

[ctx.\_matchedRoute 14](#_Toc531120333)

[ctx.\_matchedRouteName 14](#_Toc531120334)

[ctx.params 14](#_Toc531120335)

[.url(name, params [, options]) ⇒ String | Error 14](#_Toc531120336)

[router.routes 14](#_Toc531120337)

[router.use([path], middleware) 15](#_Toc531120338)

[router.prefix(prefix) 15](#_Toc531120339)

[router.allowedMethods() 15](#_Toc531120340)

[.redirect(source, destination, code) ⇒ Router 15](#_Toc531120341)

[.route(name) ⇒ Layer | false 15](#_Toc531120342)

[.param(param, middleware) ⇒ Router 16](#_Toc531120343)

[Router.url (path, params) ⇒ String 16](#_Toc531120344)

[技巧 16](#_Toc531120345)

[链式调用 16](#_Toc531120346)

[路由嵌套 17](#_Toc531120347)

[koa-passport 17](#_Toc531120348)

[确定策略 17](#_Toc531120349)

[绑定session 18](#_Toc531120350)

[初始化 18](#_Toc531120351)

[使用 18](#_Toc531120352)

[其他插件 18](#_Toc531120353)

[koa-static 18](#_Toc531120354)

[koa-logger 19](#_Toc531120355)

[koa-bodyparser 19](#_Toc531120356)

[koa-multer 19](#_Toc531120357)

[kcors 19](#_Toc531120358)

[busboy 20](#_Toc531120359)

[koa-jsonp 20](#_Toc531120360)

# app

## app.listen

app.listen(3000);

一个语法糖

**const** http = require('http');

**const** Koa = require('koa');

**const** app = **new** Koa();

http.createServer(app.callback()).listen(3000);

## app.callback

## app.use(function)

## app.keys

## app.context

# context对象

## 概览

{ request:

{ method: 'POST',

url: '/path1',

header:

{ host: 'localhost:3333',

connection: 'keep-alive',

'content-length': '141',

'user-agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_12\_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.94 Safari/537.36',

'cache-control': 'no-cache',

origin: 'chrome-extension://fhbjgbiflinjbdggehcddcbncdddomop',

'postman-token': '87f77571-fdec-6a8f-d2d9-71afb8d25b98',

'content-type': 'multipart/form-data; boundary=----WebKitFormBoundaryohgjlzNe3d81cKMx',

accept: '\*/\*',

'accept-encoding': 'gzip, deflate, br',

'accept-language': 'zh-CN,zh;q=0.9,en;q=0.8' } },

response: { status: 404, message: 'Not Found', header: {} },

app: { subdomainOffset: 2, proxy: false, env: 'development' },

originalUrl: '/path1',

req: '<original node req>',

res: '<original node res>',

socket: '<original node socket>' }

request和response就是node里的request和response。

## req/ request

req表示原生node的requset，request表示koa的request

## res/response

## state

## app

## .cookies.get/set

## throw([status],[mes],[properties])

## assert()

## respond

## request别名

* ctx.header
* ctx.headers
* ctx.method
* ctx.method=
* ctx.url
* ctx.url=
* ctx.originalUrl
* ctx.origin
* ctx.href
* ctx.path
* ctx.path=
* ctx.query
* ctx.query=
* ctx.querystring
* ctx.querystring=
* ctx.host
* ctx.hostname
* ctx.fresh
* ctx.stale
* ctx.socket
* ctx.protocol
* ctx.secure
* ctx.ip
* ctx.ips
* ctx.subdomains
* ctx.is()
* ctx.accepts()
* ctx.acceptsEncodings()
* ctx.acceptsCharsets()
* ctx.acceptsLanguages()
* ctx.get()

## response别名

* ctx.body
* ctx.body=
* ctx.status
* ctx.status=
* ctx.message
* ctx.message=
* ctx.length=
* ctx.length
* ctx.type=
* ctx.type
* ctx.headerSent
* ctx.redirect()
* ctx.attachment()
* ctx.set()
* ctx.append()
* ctx.remove()
* ctx.lastModified=
* ctx.etag=

# request

## 头，主体相关

### header/header=/headers/headers=

获取和设置请求头对象。

后面两个是前面两个的别名。

### method/method=

### length

### type

### charset

### fresh/stale

### is(types

### accepts四个

* request.accepts(types)
* request.acceptsEncodings(types)
* request.acceptsCharsets(charsets)
* request.acceptsLanguages(langs)

## 路径相关

### url/url=/originalUrl/origin

url: /?key2=value2

originalUrl 类似url

origin类似host http://localhost:3333

### href

### path/path=

'/'或者'/path1'

### querystring/querystring=

只比search少个？

### search/search=

?key2=value2,只是链接?及后面的部分。

### query/query=

{key2:'value2'}

### host/hostname

没有协议名。

protocol

secure

ip

subdomains

## 其他

### URL

### ips

### idempotent

### socket

### get

# response

## 头相关

### header/headers

### length/length=

### String

### Buffer

### Stream

### Object

### get(field)/set(field,value)/append(field,value)remove(field)

### type=

### is

### attachment()

### headerSent

### lastModified/lastModified=

### etag

### vary()

### flushHeaders()

## 状态码/状态消息

### \*status/status=

### message/message=

## 响应主体

### \*body

## 重定向

### redirect(url,[alt])

## 其他

### socket

# 插件

# koa-router

## new Router

[opts]/[opts.prefix]

## router.get|put|post|patch|delete|del|all

返回一个路由，所以可以使用函数式调用。

回调可以用箭头函数

.post('/users/:id', function (ctx, next) {

// ...

})

### ctx.\_matchedRoute

路径名

### ctx.\_matchedRouteName

路由名

### ctx.params

router.get('/:category/:title', function (ctx, next) {

console.log(ctx.params);

// => { category: 'programming', title: 'how-to-node' }

});

## .url(name, params [, options]) ⇒ String | Error

构造url路径，返回一个string，用于测试吧。

## router.routes

router.use('/', home.routes(), home.allowedMethods())

app.use(router.routes())

## router.use([path], middleware)

不能直接用use处理某个路径

router.use(['/users', '/admin'], userAuth());

## router.prefix(prefix)

router.prefix('/things/:thing\_id')

替换初始化的前缀。

## router.allowedMethods()

对不支持的请求方法的处理

app.use(router.routes());

app.use(router.allowedMethods({

throw: true,

notImplemented: () => new Boom.notImplemented(),

methodNotAllowed: () => new Boom.methodNotAllowed()

}));

## .redirect(source, destination, code) ⇒ Router

router.redirect('/login', 'sign-in');

一个语法糖。

router.all('/login', function (ctx) {

ctx.redirect('/sign-in');

ctx.status = 301;

});

.route(name) ⇒ Layer | false

## .param(param, middleware) ⇒ Router

为路由引入一些属性，对象。

router

.param('user', function (id, ctx, next) {

ctx.user = users[id];

if (!ctx.user) return ctx.status = 404;

return next();

})

.get('/users/:user', function (ctx) {

ctx.body = ctx.user;

})

## Router.url (path, params) ⇒ String

## 技巧

### 链式调用

router.get(

'/users/:id',

function (ctx, next) {

return User.findOne(ctx.params.id).then(function(user) {

ctx.user = user;

return next();

});

},

function (ctx) {

console.log(ctx.user);

// => { id: 17, name: "Alex" }

}

);

### 路由嵌套

const forums = new Router();

const posts = new Router();

posts.get('/', function (ctx, next) {...});

posts.get('/:pid', function (ctx, next) {...});

forums.use('/forums/:fid/posts', posts.routes(), posts.allowedMethods());

// responds to "/forums/123/posts" and "/forums/123/posts/123"

app.use(forums.routes());

# koa-passport

只是提供passport对koa的支持，所以还是要安装passport

## 确定策略

可以有多个策略，用关键字区分。

export function registerLocal(*passport*) {

passport.use('local', **new** *LocalStrategy*({

usernameField: 'username',

passwordField: 'password'

}, async(*username*, *password*, *done*) => {

try {

} catch (error) {

done(error);

}

}));

}

## 绑定session

// serializeUser 在用户登录验证成功以后将会把用户的数据存储到 session 中

passport.serializeUser((user, done) => done(null, user.username));// deserializeUser 在每次请求的时候将从 session 中读取用户对象

passport.deserializeUser(async(username, done) => {

done(null, await User.findOne({username}));

});

## 初始化

session不是必须的。

const session = require("koa-session2")

app.use(session({key: "SESSIONID"}))

app.use(passport.initialize())

app.use(passport.session()

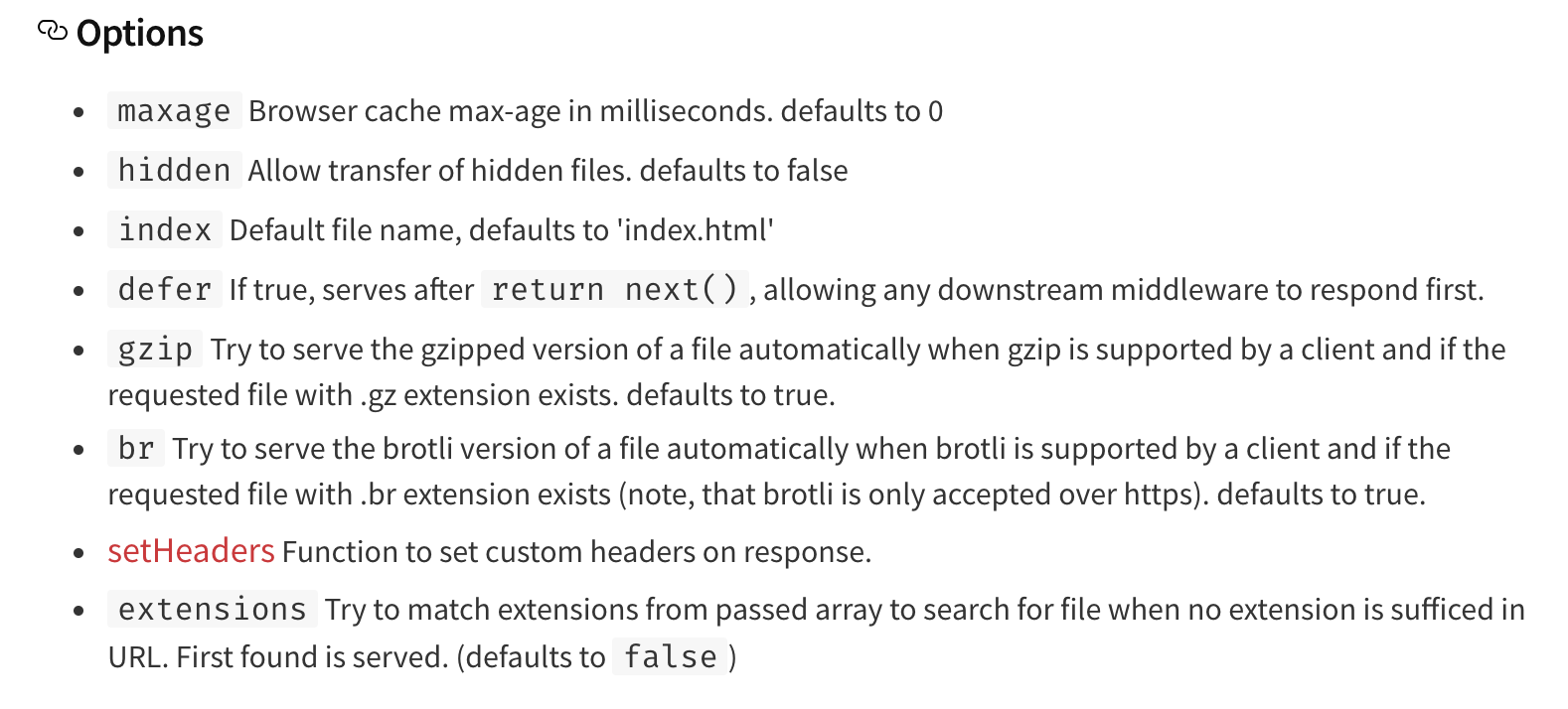
## 使用

passport.authenticate('local', {session: false});

# 其他插件

## koa-static

app.use(require('koa-static')(root, opts));



extensions

## koa-logger

## koa-bodyparser

把koa2上下文的formData数据解析到ctx.request.body中

支持Josn,form,text类型

如果想要解析multipart/form-data类型,可以用koa-multer

**注意：如果使用了router,这个要放在router前面。**

## koa-multer

const upload = multer({ dest: 'uploads/' });

app.use(route.post('/profile', upload.single('avatar')));

## kcors

app.use(kcors());

## busboy

busboy 模块是用来解析POST请求，node原生req中的文件流。

## koa-jsonp

app.use(jsonp())即可。