基于node的http库

node开发web服务

# 从一个callback开始

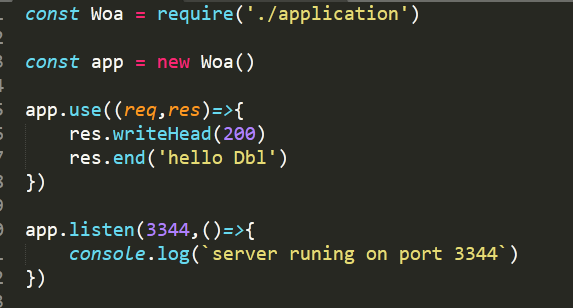


# 封装api

## application.js



## server.js



# 构建ctx

### application.js

const http = require('http')

let request = {

get url(){

return this.req.url

}

}

let response = {

get body(){

return this.\_body

},

set body(val){

this.\_body = val

}

}

let context = {

get url(){

return this.request.url

},

get body(){

return this.response.body

},

set body(val){

this.response.body = val

}

}

class Application{

constructor(){

//this.callback = ()=>{}//定义一个空函数

this.context = context

this.request = request

this.response = response

}

use(callback){

this.callback = callback //把本身的callback对应传进来的callback

}

listen(...args){

const server = http.createServer(async(req,res)=>{

let ctx = this.createCtx(req,res)

await this.callback(ctx)

ctx.res.end(ctx.body)

})

server.listen(...args)//将10行传进来的所有的参数直接传到这里来就行了

}

createCtx(req,res){

let ctx = Object.create(this.context)

ctx.request = Object.create(this.request)

ctx.response = Object.create(this.response)

ctx.req = ctx.request.req = req//将原生的req挂载到我们模拟的req上

ctx.res = ctx.response.res = res

return ctx

}

}

//对外暴露

//

module.exports = Application

### server.js

const Woa = require('./application')

const app = new Woa()

app.use(async (ctx)=>{

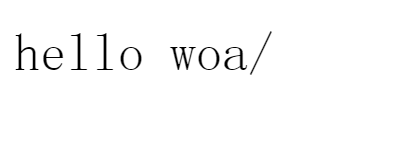
ctx.body = 'hello woa'+ctx.url

})

app.listen(3344,()=>{

console.log(`server runing on port 3344`)

})



# 实现中间件机制——同步compose

引出compose的概念

function add(x,y){

return x+y

}

function double(z){

return z\*2

}

//将两个函数栓起来

const res1 = add(1,2)

const res2 = double(res1)

console.log(res2)

//compose

const res3 = double(add(1,2))

问题来了：

我们不建议这么写。因为我们koa中间件的数目是不固定的（就只有1和2？就只有一个add?）

要有有一个良好的形式，能够处理任意数量的中间件！

function add(x,y){

return x+y

}

function double(z){

return z\*2

}

const middlewares = [add,double]

let len = middlewares.length

function compose(midds){

return (...args)=>{

//初始值

let res = midds[0](...args)

for(let i=1;i<len;i++){

res = midds[i](res)

}

return res

}

}

const fn = compose(middlewares)

const res = fn(1,2)

console.log(res)

# 实现中间件机制——异步compose

上面由于用了for，是不能完成异步有序执行的。

由于要支持async,所以对外暴露的next要是promise

//同步

async function fn1(next){

console.log('fn1')

await next()

console.log('end fn1')

}

//延迟的异步

async function fn2(next){

console.log('fn2')

await delay()

await next()

console.log('end fn2')

}

async function fn3(next){

console.log('fn3')

}

function delay(){

return new Promise((resolve,reject)=>{

setTimeout(()=>{

resolve()

},2000)

})

}

//要实现fn1->fn2->2000ms->fn3->end fn2->end fn1

//

function compose(middlewares){

return function(){

return dispatch(0)

function dispatch(i){

let fn = middlewares[i]

if(!fn){

return Promise.resolve()

}

return Promise.resolve(fn(function next(){

return dispatch(i+1)

//fn1(fn2(fn3))但是都是以promise对象返回的

}))

}

}

}

const middlewares = [fn1,fn2,fn3]

const finalFn = compose(middlewares)

finalFn()

# 过关斩将：打造属于自己的简单版koa