一.网络编程

1.http是单双工，websocket 是全双工

二.node基础

1.nodejs是一个异步的事件驱动的，javascript运行时，具有事件驱动，非阻塞等特点。

2.nodejs核心模块：fs http buffer event os

3.执行node程序：node app.js

4.想要每次修改重新启动，安装npm i -g nodemon,启动：nodemon app.js

5.Vscode 调试node 配置如下：

Configurations:[

Type”node”,

Request:’launch’,

Name:’Lanunch Program’,

“program”:”${workspaceFolder}/app.js”

]

6.内存占用率

Const os = require(“os”);

Const men = os.freemem()/os.totalmen() \*100

Console.log(“内存占用率${men}”)

7.cpu占用率

Const cupStat = require(‘cpu-stat’);

cpuStat.usagePercent((err,percent)=>{

Console.log(‘cpu占用${percent}%’)

})

8.文件系统

Const fs =require(‘fs’)

//同步读取数据

Const data =fs.readFileSync(‘package.json’)

Console.log(data.toString())//默认data.toStrng(‘utf-8’)

//异步的方式

Fs.readFile(‘./package.json’,()=>{

Console.log(data.toString(‘utf-8’))

})

//promise方式：

Const {promisify} = reqquire(‘util’)

Const readFile = promisify(fs.readFile)

readFile(“.package.json”).then(data=>{

Console.log(data.toString())

})

//Async await 方式

(async ()=>{

//await 后面跟

Const data =await readFile(‘./package.json’)

Console.log(data.toString())

})()

9.线程通讯

###main.js

let child = require('child\_process');

let os = require('os');

let time = 1;

createChild();

function createChild() {

let start = new Date().getTime();

let cpus = os.cpus();

for(let i=0;i<cpus.length;i++){

let child\_process = child.fork('./zhishu.js');

child\_process.on('message',function (m) {//当子线程send的时候触发此方法

console.log(child\_process.pid+'的时间：'+m.time+',质数个数为：'+m.total);

});

child\_process.send({time:i});//给子线程传递参数

}

let end = new Date().getTime();

console.log('start server ...'+(end-start));

}

##child.js

//判断一个数是不是质数

function zhishu\_js(num) {

if (num == 1) {

return false;

}

if (num == 2) {

return true;

}

for (var i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) {

return false;

}

}

return true;

}

let time = 1;

//测试一共需要多长时间

//test();

function test() {

let total = 0;

let start = new Date().getTime();

for(let i = 1;i<400000;i++){

if(zhishu\_js(i)){

total+=1;

}

}

let end = new Date().getTime();

console.log('total time ='+(end - start)+'total = '+total);

}

//接受主线程中线程传递进来的参数

process.on('message',function (message) {

let total = 0;

time = message.time\*100000;

console.log('message:'+time);

let start = new Date().getTime();

let i = time+100000;

for(time;time<i;time++){

if(zhishu\_js(time)){

total++;

}

}

let end = new Date().getTime();

process.send({time:(end-start),total:total});

});

10.bugger

//新建一个buffer数组

const buf01 = Buffer.alloc(10)

console.log(buf01)

//把数组转变为buffer

const buf02 = Buffer.from([1,2,3])

console.log(buf02)

buf01.write('hello')

//把字符串转换为buffer

const buf03 = Buffer.from('Buffer创建方法')

// buf03.write('hello')

// buf03.write('hello')

// buf03.write('hello')

// console.log(buf03)

// console.log(buf03.toString())

//合并buff01和buff03

const buf04 = Buffer.concat([buf01,buf03])

console.log(buf04.toString('utf-8'))//字符串输出

11.http，创建服务器

Const http = require(‘http’)

//所有的请求都会请求到这个地方

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

Console.log(‘request’,res)

Const {ur, method} = req;

Res.end(‘’res’)

})

Server.listen(3000)

12.nodejs最简单的服务器

const http = require('http')

const fs = require('fs')

const server = http.createServer((request, response) => {

const { url, method, headers } = request

console.log('this is a request', request.url,headers.accept)

if (url === '/' && method === 'GET') {

fs.readFile('index.html', (err, data) => {

if (err) {

response.writeHead(500, { 'Content-Type': 'text/pain;charset=utf-8' })

response.end('服务器错误')

}

response.statusCode = 200

response.setHeader('Content-Type', 'text/html')

response.end(data)

})

} else if (url == '/users' && method === 'GET') {

response.statusCode = 200

response.setHeader('Content-Type', 'text/html')

response.end(JSON.stringify({ a: 123 }))

//头部是image

} else if (method == 'GET' && headers.accept.indexOf('image/\*') !== -1) {

//把读取的文件流引到response里面

fs.createReadStream('.' + url).pipe(response)

}

else {

response.statusCode = 404

response.setHeader('Content-Type', 'text/plain;charset=utf-8')

response.end('No Page 页面不存在')

}

13.express = require(‘express’)

Const app = express()

App.get(‘/’,()=>{

Res.end(‘helloword..’)

})

App.get(“/user”,()=>{

Res.end(JSON.stringify({name:’abc’}))

})

App.listen(3000,()=>{

Console.log(‘app listen at 3000’)

})

14.简易express 的实现：

const http = require('http')

const url = require('url')

let router = []

class Application {

get(path, handler) {

router.push({

path,

method: 'get',

handler

})

}

listen() {

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

const { pathname } = url.parse(req.url, true)

for (const item of router) {

const { path, method, handler } = item

if (pathname == path && req.method.toLowerCase() == method) {

return handler(req, res)

}

}

})

server.listen(...arguments)

}

}

module.exports = function createApplication(){

return new Application()

}

14.tcp 协议，实现一个即时通讯im

Const net = require(“net”);

Const chatServer = net.createSever()

Const clientList =[];

chatServer.on(‘connection’,client=>{

Client.wrire(“hi!\n”)

clientList.push(client);

Client.on(‘data’,data=>{

Console.log(‘receive:’,data.toString())

clientList.forEac(v=>{

v.write(data)

})

})

}

15状态码：

1xx：指示信息--表示请求已接收，继续处理

2xx：成功--表示请求已被成功接收、理解、接受

3xx：重定向--要完成请求必须进行更进一步的操作

4xx：客户端错误--请求有语法错误或请求无法实现

5xx：服务器端错误--服务器未能实现合法的请求

16.解决跨域问题

1.jsonp：通过script标签的src 发get请求到后台，带上callback参数

服务器带上callbackname(json.stringify(data))这样的数据格式append到script标签内部从而现实跨域

2.webpack代理服务器模式

3.Cors，后端的方案

1.在头部设置关键字：

const http = require('http')

const fs = require('fs')

const server = http.createServer((request, response) => {

const { url, method, headers } = request

//允许跨域设置，这个是简单请求的情况下的配置（请求header里面没有自己加的字段）

Response.setHeader(“Access-Conrtrol-Allow-Origin”,\*)

if (url === '/' && method === 'GET') {

fs.readFile('index.html', (err, data) => {

if (err) {

response.writeHead(500, { 'Content-Type': 'text/pain;charset=utf-8' })

response.end('服务器错误')

}

response.statusCode = 200

response.setHeader('Content-Type', 'text/html')

response.end(data)

})

} else if (url == '/users' && method === 'GET') {

response.statusCode = 200

response.setHeader('Content-Type', 'text/html')

response.end(JSON.stringify({ a: 123 }))

}

3.1简单请求:get head post,解决跨域只需要加上，

//Response.setHeader(“Access-Conrtrol-Allow-Origin”,\*)

3.2需要预检的请求：put delete options patch 或者改动header字段或者新增

字段的,需要发预检请求，就是一个options的请求,解决跨域的问题需要配置多行

res.setHeader('Access-Control-Allow-Origin', 'http://127.0.0.1:8080')

res.setHeader('Access-Control-Allow-Headers', 'X-Token,Content-Type')

3.3如果想跨域可以取得到cookie 配置

res.setHeader('Access-Control-Allow-Credentials', 'true');//允许认证鉴权，跨域可以取到cookie

用插件方式解决：

Const cors =require(‘koa2-cors’);

App.use(cors())

17.网络爬虫：

const originRequest = require("request");

const cheerio = require("cheerio");

const iconv = require("iconv-lite");

function request(url, callback) {

const options = {

url: url,

encoding: null

};

originRequest(url, options,callback);

}

for (let i = 100553; i < 100563; i++) {

const url = `https://www.dy2018.com/i/${i}.html`;

request(url, function(err, res, body) {//请求html

const html = iconv.decode(body, "gb2312");//编码

const $ = cheerio.load(html);//cheerio 是类似jquery的一个库

console.log($(".title\_all h1").text());//通过jquery的形式取数值

});

}

18.聊天程序：

//Index.js:

var app = require('express')();

var http = require('http').Server(app);

var io = require('socket.io')(http);

app.get('/', function(req, res){

res.sendFile(\_\_dirname + '/index.html');

});

io.on('connection', function(socket){

console.log('a user connected');

//响应某用户发送消息

socket.on('chat message', function(msg){

console.log('chat message:' + msg);

// 广播给所有人

io.emit('chat message', msg);

// 广播给除了发送者外所有人

// socket.broadcast.emit('chat message', msg)

});

socket.on('disconnect', function(){

console.log('user disconnected');

});

});

http.listen(3000, function(){

console.log('listening on \*:3000');

});

//index.html

<!DOCTYPE html>

<html>

<head>

<title>Socket.IO chat</title>

<style>

</style>

</head>

<body>

<ul id="messages"></ul>

<form action="">

<input id="m" autocomplete="off" /><button>Send</button>

</form>

<script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/2.2.0/socket.io.js"></script>

<script src="http://libs.baidu.com/jquery/2.1.1/jquery.min.js"></script>

<script>

$(function() {

var socket = io();

$("form").submit(function(e) {

e.preventDefault(); // 避免表单提交行为

socket.emit("chat message", $("#m").val());

$("#m").val("");

return false;

});

socket.on("chat message", function(msg) {

$("#messages").append($("<li>").text(msg));

});

});

</script>

</body>

</html>

19.http2:

多路复用 - 雪碧图、多域名CDN、接口合并

官方演示 - https://http2.akamai.com/demo

多路复用允许同时通过单一的 HTTP/2 连接发起多重的请求-响应消息；而HTTP/1.1协议中，浏览器客户

端在同一时间，针对同一域名下的请求有一定数量限制。超过限制数目的请求会被阻塞\*\*

首部压缩

http/1.x 的 header 由于 cookie 和 user agent很容易膨胀，而且每次都要重复发送。http/2使用

encoder 来减少需要传输的 header 大小，通讯双方各自 cache一份 header fifields 表，既避免了重复

header 的传输，又减小了需要传输的大小。高效的压缩算法可以很大的压缩 header，减少发送包的数

量从而降低延迟

服务端推送

// }, 1000);

// websocket方式

const socket = io(host)

socket.on('chat', list => {

this.list = list

});

}

# 创建私钥

openssl genrsa -out privatekey.pem 1024

# 创建证书签名请求

openssl req -new -key privatekey.pem -out certrequest.csr

# 获取证书，线上证书需要经过证书授证中心签名的文件；下面只创建一个学习使用证书

openssl x509 -req -in certrequest.csr -signkey privatekey.pem -out certificate.pem

# 创建pfx文件

openssl pkcs12 -export -in certificate.pem -inkey privatekey.pem -out

certificate.pfx开课吧

web

全栈架构师

在 HTTP/2 中，服务器可以对客户端的一个请求发送多个响应。举个例子，如果一个请求请求的是

index.html，服务器很可能会同时响应index.html、logo.jpg 以及 css 和 js 文件，因为它知道客户端会

用到这些东西。这相当于在一个 HTML 文档内集合了所有的资源

1. fs文件管理程序

const fs = require("fs");

function get(key) {

fs.readFile("./db.json", (err, data) => {

const json = JSON.parse(data);

console.log(json[key]);

});

}

function set(key, value) {

fs.readFile("./db.json", (err, data) => {

// 可能是空文件，则设置为空对象

const json = data ? JSON.parse(data) : {};

json[key] = value; // 设置值

// 重新写入文件

fs.writeFile("./db.json", JSON.stringify(json), err => {

if (err) {

console.log(err);

}

console.log("写入成功！");

});

});

}

// 命令行接口部分

const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

rl.on("line", function(input) {

const [op, key, value] = input.split(" ");

if (op === 'get') {

get(key)

} else if (op === 'set') {

set(key, value)

} else if(op === 'quit'){

rl.close();

}else {

console.log('没有该操作');

}

});

rl.on("close", function() {

console.log("程序结束");

process.exit(0);

});

1. Nodejs---Mysql

安装mysql模块： npm i mysql --save

const mysql = require("mysql");

// 连接配置

const cfg = {

host: "localhost",

user: "root",

password: "admin", // 修改为你的密码

database: "kaikeba" // 请确保数据库存在

};

// 创建连接对象

const conn = mysql.createConnection(cfg);

// 连接

conn.connect(err => {

if (err) {

throw err;

} else {

console.log("连接成功！");

}

});

// 查询 conn.query()

// 创建表

const CREATE\_SQL = `CREATE TABLE IF NOT EXISTS test (

id INT NOT NULL AUTO\_INCREMENT,

message VARCHAR(45) NULL,

PRIMARY KEY (id))`;开课吧

const INSERT\_SQL = `INSERT INTO test(message) VALUES(?)`;

const SELECT\_SQL = `SELECT \* FROM test`;

conn.query(CREATE\_SQL, err => {

if (err) {

throw err;

}

// 插入数据

conn.query(INSERT\_SQL, "hello,world", (err, result) => {

if (err) {

throw err;

}

console.log(result);

conn.query(SELECT\_SQL, (err, results) => {

console.log(results);

conn.end(); // 若query语句有嵌套，则end需在此执行

})

});

});

1. Nodejs my-sql orm -- sequelize

基于promise的orm，支持多种数据库，事务，关联等等

安装 npm i sequelize myql2 -S

基本用法：

const Sequelize = require("sequelize");

const sequelize = new Sequelize("kaikeba", "root", "admin", {

host: "localhost",

dialect: "mysql",

operatorsAliases: false

});

// 定义模型

const Fruit = sequelize.define("Fruit", {

//name: { type: Sequelize.STRING(20), allowNull: false },

Name:{

type:Sequelize.STRING,

allowNull:false,

Get(){

Const fname = this.getDataValue(‘name’)

Const price = this.getDataValue(“price”)

Const stock = this.getDataValue(“stock”)

Return `${fname}()价格：${price} 库存：${stock}kg`

}

},

price: { type: Sequelize.FLOAT, allowNull: false,

validate: { //字段验证

isFloat: { msg: "价格字段请输入数字" },

min: { arg

},

stock: { type: Sequelize.INTEGER, defaultValue: 0 }

});

// 同步数据库，force: true则会删除已存在表

Fruit.sync()

.then(() => {

// 添加测试数据

return Fruit.create({

name: "香蕉",

price: 3.5

});

})

.then(() => {

// 查询

Fruit.findAll().then(fruits => {

console.log(JSON.stringify(fruits));

});

});

//强制同步：

Fruit.sync({force: true})

//不自动生成时间字段：

const Fruit = sequelize.define("Fruit", {}, {

{

timestamps: false,//不会自动生成时间字段

getterMethods:{

amount(){

return this.getDataValue("stock") + "kg";

}

},

setterMethods:{

amount(val){

const idx = val.indexOf('kg');

const v = val.slice(0, idx);

this.setDataValue('stock', v);

}

}

}

});

Fruit.classify = function (name) {

const tropicFruits = ['香蕉', '芒果', '椰子']; // 热带水果

return tropicFruits.includes(name) ? '热带水果' : '其他水果';

};

['香蕉', '草莓'].forEach(f => console.log(f + '是' + Fruit.classify(f)));

Fruit.prototype.totalPrice = function (count) {

return (this.price \* count).toFixed(2);

};

// 同步数据库，force: true则会删除已存在表

let ret = await Fruit.sync({ force: false })

// console.log('sync', ret)

ret = await Fruit.create({

name: "香蕉",

price: 3.5

})

1. nodejs---mogodb

(async () => {

const { MongoClient: MongoDB } = require('mongodb')

// 创建客户端

const client = new MongoDB(

'mongodb://localhost:27017',

{

userNewUrlParser: true //用新的user解释器皿

}

)

let ret

// 创建连接

ret = await client.connect()

console.log('ret:', ret)

const db = client.db('test')

const fruits = db.collection('fruits')

// 添加文档

ret = await fruits.insertOne({

name: '芒果',

price: 20.1

})

console.log('插入成功', JSON.stringify(ret))

// 查询文档

ret = await fruits.findOne()

console.log('查询文档:', ret)

// 更新文档

ret = await fruits.updateOne({ name: '芒果' },

{ $set: { name: '苹果' } })

console.log('更新文档', JSON.stringify(ret.result))

// 删除文档

ret = await fruits.deleteOne({name: '苹果'})//删除一条

await fruits.deleteMany()//全部删除

client.close()

})()

1. nodejs event

Const EventRmitter = require(‘events’).EventEmitter;

Const event = new EventEmitter();

Event.on(“some\_event”,num=>{

Console.log(num)

})

Event.emit(“some\_event”,num++)

1. 用vue + express+ mongo实现网站

//新建db.js

const conf = {

url: "mongodb://localhost:27017",

dbName: 'test',

}

const EventEmitter = require("events").EventEmitter;

// 客户端

const MongoClient = require("mongodb").MongoClient;

class Mongodb {

constructor(conf) {

// 保存conf

this.conf=conf;

this.emmiter = new EventEmitter();

// 连接

this.client = new MongoClient(conf.url, { useNewUrlParser: true });

this.client.connect(err => {

if (err) throw err;

console.log("连接成功");

this.emmiter.emit("connect");

});

}

col(colName, dbName = conf.dbName) {

return this.client.db(dbName).collection(colName);

}

once(event, cb) {

this.emmiter.once(event, cb);

}

}

// 2.导出db

module.exports = new Mongodb(conf);

//新建index.js

const express = require("express");

const app = express();

const path = require("path");

const mongo = require("./models/db");

const testdata = require("./models/testdata");

app.get("/", (req, res) => {

res.sendFile(path.resolve("./index.html"));

});

app.get("/api/list", async (req, res) => {

// 分页查询

const page = +req.query.page;

try {

const col = mongo.col("fruits");

const total = await col.find().count();

const fruits = await col

.find()

.skip((page - 1) \* 5)

.limit(5)

.toArray();

res.json({ ok: 1, data: { fruits, pagination: { total, page } } });

} catch (error) {

console.log(error);

}

});

app.listen(3000);

//新建index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<meta http-equiv="X-UA-Compatible" content="ie=edge" />

<script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>

<script src="https://unpkg.com/element-ui/lib/index.js"></script>

<script src="https://unpkg.com/axios/dist/axios.min.js"></script>

<link rel="stylesheet" href="https://unpkg.com/element-ui/lib/theme-chalk/index.css" />

<title>瓜果超市</title>

</head>

<body>

<div id="app">

<ul>

<li v-for="fruit in fruits" :key="fruit.\_id">

{{ fruit.name }} - {{ fruit.price }}

</li>

</ul>

<el-pagination layout="prev, pager, next" @current-change="currentChange" :total="total"></el-pagination>

</div>

<script>

var app = new Vue({

el: "#app",

data: {

page: 1,

total: 0,

fruits: []

},

created() {

this.getData();

},

methods: {

currentChange(page) {

this.page = page;

this.getData();

},

getData() {

axios

.get(`/api/list?page=${this.page}`)

.then(res => res.data)

.then(({

data

}) => {

this.fruits = data.fruits;

this.total = data.pagination.total;

});

}

}

});

</script>

</body>

</html>

运行node index.js

1. 当没有管理后台的时候可以用keystone

https://www.keystonejs.com/

1. eventLoop

Js首先在执行栈中执行同步代码，如果遇到异步代码，就会把异步代码挂起放到任务队列中，当执行完成所有的同步代码，eventloop把任务队列中的代码压进执行栈中执行，如此循环。

其中异步任务又分为宏任务，和微任务

微任务有：process.nextTick.promise,object.observe,mutationObserber 注意(process.nextTick > promise.then)

宏任务有：script,setTimout,setInterval,i/o,ui-rendering

setTimeout(() => {

setImmediate(() => {

console.log('setImmediate');

});

setTimeout(() => {

console.log('setTimeout');

}, 0);

}, 0);

被异步代码包围的话 setImmediate先于setTimeout，否则两者随机

1. koa框架

koa是express的下一代基于nodejs的web框架

Koa2完全使用Promise并配合async来实现异步

安装：npm i koa -S

1.简单的koa服务器

//Index.js

Const Koa = require(‘koa’);

Const app = new Koa();

Const mid1 = async (ctx,next)=>{

Ctx.body =”hello ”

Await next()

Ctx.body = ctx.body+”!!!!”

}

Const mid2 = acync ()=>{

Ctx.type = “text/html;charse=utf-8”

Await next();

}

Const mid3 = async (ctx,next)=>{

Ctx.body = ctx.body =”kaikeba”

Await next()

}

App.use(mid1)

App.use(mid2)

App.use(mid3)

App.listen(3000)

洋葱圈模型输出：hello kaikeba ！！！！

ctx对象：

Ctx.req --- nodejs 的request

Ctx.res --- nodejs的response

Ctx.response --- koa的response

Ctx.requst --- koa的request

Ctx.state推挤命名空间 用于中间件传递信息和前端视图

Ctx.app应用程序引用

1. 错误处理：

app.use(async (ctx, next) => {

try {

await next();

} catch (error) {

// 响应用户

ctx.status = error.statusCode || error.status || 500;

ctx.body = error.message;

// 触发应用层级错误事件，没有emit就不会报错了

ctx.app.emit("error", error, ctx);

console.log('捕获到错误：', error.message);

}

});

//顶层接受事件

app.on('error', (err, ctx) =>{

// console.error(err);//没有写代码默认打印这个

// console.log('没得事')

// throw err //如果向上抛的话程序就直接挂了

})

1. koa -- router

新建文件Routes.

新建文件index.js

const Router = require("koa-router");

const router = new Router();

router.get("/", async ctx => {

//若cookie中存在记录则不再播放，index.js

let showVideo;

if (ctx.cookies.get('isPlayed')) {

showVideo = false;

} else {

showVideo = true;

ctx.cookies.set('isPlayed', true, { maxAge: 7 \* 24 \* 3600000 });

}

// ctx.body = "index";

const list = [...ctx.state.vipCourses];

list.sort((a, b) => (a.sort - b.sort));

await ctx.render('index', { list, showVideo })

});

module.exports = router;

//主程序入口 index.js

const index = require('./routes/index');

app.use(index.routes());

1. 静态文件服务

const static = require("koa-static");

app.use(static(\_\_dirname + '/public'));

1. 模版引擎

const hbs = require('koa-hbs')

app.use(hbs.middleware({

viewPath: \_\_dirname + '/views', //视图根目录

defaultLayout: 'layout', //默认布局页面

partialsPath: \_\_dirname + '/views/partials', //注册partial目录，header，footer等小部件

disableCache: true //开发阶段不缓存

}));

1. Ssh 免密登录

Ssh root@47298.252.43

实现：

1. 输入 ssh-keygen -t rsa -P ‘’ //在根目录下面生成公私钥
2. 私钥不要给人，公钥可以给别人，把公钥上传到服务器authorized\_keys
3. 查看文件，日志文件 tail -f readme.md //查看尾部
4. Vi readme.md //编辑文件

4.鉴权

1. nodejs http 设置cookie/session

### //注意cookie一般是后端设置的

const http = require('http')

const session = {}

http.createServer((req, res) => {

const sessionKey = 'sid'

if (req.url === '/favicon.ico') {

return

} else {

//获取cookie

const cookie = req.headers.cookie

if (cookie && cookie.indexOf(sessionKey) > -1) {

res.end('Come Back')

console.log('cookie:', req.headers.cookie)

// 简略写法未必具有通用性

const pattern = new RegExp(`${sessionKey}=([^;]+);?\s\*`)

const sid = pattern.exec(cookie)[1]

console.log('session:', sid, session, session[sid])

} else {

const sid = (Math.random() \* 9999999).toFixed()

//设置cookie

res.setHeader('Set-Cookie', `${sessionKey}=${sid}`)

session[sid] = { name: 'laowang' }

res.end('hello cookie')

}

}

}).listen(3000)

1. koa中使用session，redis 使用npm install koa-session -S

const koa = require('koa')

const app = new koa()

const session = require('koa-session')

const redisStore = require('koa-redis')//用这个连接到服务器

const redis = require('redis')

const redisClient = redis.createClient(6379, 'localhost')

const wrapper = require('co-redis')

const client = wrapper(redisClient) //promise化一下redisClient

app.keys = ['some secret']

const SESS\_CONFIG = {

key: 'kkb:sess', // session名

// maxAge: 8640000, // 有效期

// httpOnly: true, //防止cookie修改

// signed: true // 签名

store: redisStore({ client }) //用redis记录session

}

//使用session

app.use(session(SESS\_CONFIG, app))

app.use(ctx => {

// 查看全部的redis

redisClient.keys('\*',(err,keys) => {

console.log('keys:',keys)

keys.forEach(key => {

redisClient.get(key, (err,val) => {//打印redis中的所有值

console.log(val)

})

})

})

if (ctx.path === '/favicon.ico') return

let n = ctx.session.count || 0

ctx.session.count = ++n

ctx.body = '第' + n + '次访问'

})

app.listen(3000)

1. Redis

一般的部署是用一台nginx和多台nodejs连接，当内存需要共享的时候就要用到redis

代码：

const redis = require('redis')

const client = redis.createClient(6379,'localhost')

client.set('hello','hahaha')

client.get('hello',function(err,v){

console.log('redis key:',v)

})

1. egg.js

Npm install egg-init -g

Egg-init egg-example --type = simple

Cd egg-example

dNpm install

1. koa 上传

Const upload =require(‘koa-multer’)({dest:’./pulic.images’})

Router.post(‘/upload’,upload.single(‘file’),ctx=>{

Console.log(“file”,ctx.req.flie);

Console.log(‘body’,ctx.req.body)

Ctx.body=”上传成功”

})

1. node 字段验证

Const bouncer = require(‘koa-bouncer’)

App.use(bouncer.middleware())

//用中间件验证

Const val = async(ctx,next)=>{

Ctx.validateBody(‘name’).isLength(6,16,’用户长度应该为6-16’)

Next()

}

const bouncer = require("koa-bouncer");

const val = async (ctx, next) => {

try {

// 校验开始

ctx

.validateBody("name")

.required("要求提供用户名")

.isString()

.trim()

.isLength(6, 16, "用户名长度为6~16位")

console.log('ctx.vals', ctx.vals);

next()

} catch (error) {

if (error instanceof bouncer.ValidationError) {

ctx.body = '校验失败：' + error.message;

return;

}

throw error

}

}

1. 验证码：

const captcha = require("trek-captcha");

router.get("/captcha", async ctx => {

const { token, buffer } = await captcha({ size: 4 });

ctx.session.captcha = token

ctx.body = buffer;

});

//index.js

const api = require('./routes/api');

app.use(api.routes());

//前端点击换验证码

<img src="/api/captcha" id="captcha" />

<script>

document.getElementById('captcha').onclick = function () {

captcha.src = "/api/captcha?r=" + Date.now();

};

</script>

1. 短信验证码

秒滴科技

配置短信模版

调用api

示例代码：

const moment = require('moment')

const md5 = require('md5')

const axios = require('axios')

const qs = require('querystring')

router.get("/sms", async function (ctx) {

// 生成6位随机数字验证码

let code = (Math.random() \* 999999).toFixed();

// 构造参数

const to = ctx.query.to; // 目标手机号码

const accountSid = "3324eab4c1cd456e8cc7246176def24f"; // 账号id

const authToken = "b1c4983e2d8e45b9806aeb0a634d79b1"; // 令牌

const templateid = "613227680"; // 短信内容模板id

const param = `${code},1`; // 短信参数

const timestamp = moment().format("YYYYMMDDHHmmss");

const sig = md5(accountSid + authToken + timestamp); // 签名

try {

// 发送post请求

const resp = await axios.post(

"https://api.miaodiyun.com/20150822/industrySMS/sendSMS",

qs.stringify({ to, accountSid, timestamp, sig, templateid, param }),

{ headers: { "Content-Type": "application/x-www-form-urlencoded" } }

);

if (resp.data.respCode === "00000") {

// 短信发送成功，存储验证码到session，过期时间1分钟

const expires = moment()

.add(1, "minutes")

.toDate();

ctx.session.smsCode = { to, code, expires };

ctx.body = { ok: 1 }

} else {

ctx.body = { ok: 0, message: resp.data.respDesc }

}

} catch (e) {

ctx.body = { ok: 0, message: e.message }

}

});

1. Nginx

网站配置文件一般在etc/nginx/sites-enabled 下面

server:{

listen 80;

Server\_name taro.com;

Location / {

Root: /root/source/taro-node/dist;

Index index.html index.htm

}

}