Nmp init

Npm install koa

Koa路由

Npm install koa-router

# 编写服务端

|  |
| --- |
| // 导包实例化  let Koa = require("koa");  let Router = require("koa-router");  let app = new Koa();  let router = new Router();  // 路由配置  router.get("/", async (ctx) => {  ctx.body = "hello world";  }).get("/content/:id", async (ctx) => {  ctx.body = `content = ${ctx.params.id} , ${ctx.query.name}`;  console.log(ctx.query); //get请求的链接处理  console.log(ctx.params); //动态路由  })  app.use(router.routes()).use(router.allowedMethods()); //启动路由、配置返回头  app.listen(3000); //监听3000端口 |

# 中间件

1. 应用级别，不管放在前后，都会被先执行

|  |
| --- |
| app.use(async (ctx, next) => {  console.log(new Date());  await next();  })  app.use(async (ctx, next) => {  next();  if (ctx.status == 404) {  ctx.status = 404;  ctx.body = "这是一个 404 页面"  }  else {  console.log(ctx.status)  }    }); |

App.use ，多个，具体看代码顺序，决定执行顺序。同时要看next的位置，next先执行，则是路由后，next后执行，则是路有前。

中间件1 中间件2 路由 中间件3 中间件4

1. 在路由前后执行看next
2. 同一侧的中间件，看代码顺序

Ctx会被覆盖，以最后一个为准。

问题：这是拦截所有的，如何拦截某个请求链接呢。

1. 路由中间件，看放置的位置

|  |
| --- |
| router.get('/', async (ctx, next) => {  console.log(1);  next();  }) |

必须放在路由代码之前，否则不生效，然后具体执行顺序看next位置。

# 模板引擎

使用步骤

1.依赖

npm install koa-views

npm install ejs

2.配置

const views = require('koa-views');

app.use(views('views', { map: {html: 'ejs' }})); //路径是views（相对），使用ejs。页面是html

或者使用：

app.use(views(“xxx”, { extension: 'ejs' })) //页面是ejs

1. 使用

await ctx.render(‘index',{ title }) index页面，title是一个参数

其他：

引入其他页面：<%- include header.html %>

页面显示变量值：<%=h%>

页面显示变量值（html）：<%-h%>

页面语法：（类似el表达式）

|  |
| --- |
| <% if(true){ %>  <div>true</div>  <%} else{ %>  <div>false</div>  <%} %>  <%for(var i=0;i<list.length;i++) { %><li><%=list[i] %></li>  <%}%> |

添加全局的变量，使得在不同路由都能有值

|  |
| --- |
| app.use(async (ctx,next)=>{  ctx.state.userinfo='张三';  await next();/\*继续向下匹配路由\*/  }) |

# 全局变量

Ctx.state.userinfo=”xx”

这是一个全局变量，其他模板引擎都可以使用

和global的区别

放在state，在前端页面显示：{{userinfo}}

# 接收post数据

Get数据使用ctx.query即可。

## 原生做法：

1. 在接收数据后，进行处理，然后做出响应。这整个过程必须是同步的。否则接收数据时异步，就无法进行处理和响应了。
2. 使用await和promise使得接收过程同步。
3. 处理，响应。

|  |
| --- |
| function parsePostData(ctx){  return new Promise((resolve,reject)=>{  try{  let postdata="";  ctx.req.on('data',(data)=>{  postdata += data  })  ctx.req.on("end",function(){  resolve(postdata);  })  }catch(error){  reject(error);  }  });  } |

## 使用koa-bodyparser /koa-body

1. 依赖

Npm install koa-bodyparser

1. 导包

var bodyParser = require('koa-bodyparser');

app.use(bodyParser());

1. 使用  
   ctx.body = ctx.request.body;

# 处理静态资源

1. 依赖

Npm install koa-static

1. 导包使用

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

App.use(static(xxx)) 相对路径或绝对路径。

例子：

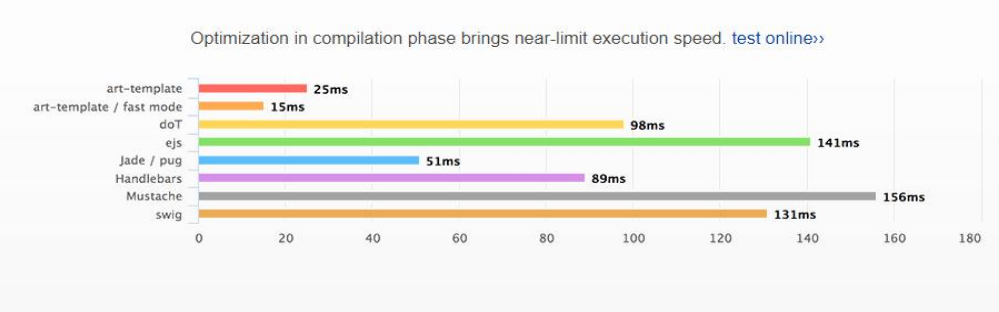
文件：{root}->static->css->index.css

则配置：app.use(static(“static”)) 也可以这样配置 app.use(static(“static/xx/../”))

那么页面的静态资源路径写 css/index.css ，因为这个路径会到配置的static文件夹下找css/index.css

注意：可以配置多个app.use(static(static))，在其中一个找不到，会去其他找，不会覆盖。

# 高速的模板引擎



1. 依赖

Npm install art-template

Npm install koa-art-template

1. 引入配置

Const path=require(“path”);

const Koa = require('koa');

const render = require('koa-art-template');

const app = new Koa();

render(app, {

root: path.join(\_\_dirname, '文件夹名'), //整体路径是 \_\_dirname（当前项目所在文件夹）+页面所在文件夹名

extname: '.html', //后缀名

debug: process.env.NODE\_ENV !== 'production' //是否debug，不改动

});

3.使用

await ctx.render('user',{});

可以使用可ejs的语法：参考：<http://aui.github.io/art-template/zh-cn/docs/syntax.html>

# 抽取公共模块，引入

使用{{include ‘xx/xx/html’}} 以view下面的文件夹开始



# Cookie设置

ctx.cookies.set(name, value, [options])

Options是设置cookie的其他值



maxAge:具体到毫秒

Expires:直到日期

Path:该cookie能被访问的路径。 如 /hjy 那么只有访问 /hjy路径的才会把cookie传递到服务端。

Domain : .baidu.com 这么设置则baike.baidu.com、xx.baidu.com都可以共享cookie

Httponly：只有服务端可以获取cookie，使用js获取不到。

Overwrite：覆盖，没啥用处。

Signed:true 防止cookie被前端篡改。

从cookie中取数据

ctx.cookies.get('name');

## 在koa的cookie中使用中文值，会报错。

解决： 存用base64，取也用base64

console.log(new Buffer('hello, world!').toString('base64'));// 转换成 base64 字符 串：aGVsbG8sIHdvcmxkIQ==

console.log(new Buffer('aGVsbG8sIHdvcmxkIQ==', 'base64').toString());// 还原 base 64 字符串：hello, world!

# Session

1. 依赖

npm install koa-session

1. 引入配置

const session = require('koa-session');

app.keys = ['some secret hurr'];

const CONFIG = {

key: 'koa:sess', //cookie key (default is koa:sess)

maxAge: 86400000, // cookie 的过期时间 maxAge in ms (default is 1 days) 是cookie还是session的过期事件

overwrite: true, //是否可以 overwrite (默认 default true)

httpOnly: true, //cookie 是否只有服务器端可以访问 httpOnly or not (default true)

signed: true, //签名默认 true

rolling: false, //在每次请求时强行设置 cookie，这将重置 cookie 过期时间（默认：false）

renew: false, //(boolean) renew session when session is nearly expired,

};

app.use(session(CONFIG, app));

1. 使用

设置值 ctx.session.username = "张三";

获取值 ctx.session.username

# Mongodb使用（略）

# mysql使用

Npm install mysql

|  |
| --- |
| const mysql = require('mysql');  const pool = mysql.createPool({  host: '127.0.0.1',  user: 'root',  password: 'woshige2b',  database: 'messageboard'  })  let query = function (sql, values) {  return new Promise((resolve, reject) => {  pool.getConnection(function (err, connection) {  if (err) {  reject(err)  } else {  connection.query(sql, values, (err, rows) => {  if (err) {  reject(err)  } else {  resolve(rows)  }  connection.release()  })  }  })  })  }  module.exports = { query } |

# Yield作用

# Koa生成器

1. 添加全局依赖

npm install koa-generator -g

1. 生成项目

koa koa\_demo

1. 生成项目的依赖

cd koa\_demo

npm install

1. 启动项目

不再是node xx.js 而是npm start

# Koa路由模块化

将路由，或者说接口放到其他文件中，而不是都放在一个js里

分别使用

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

Router.get(“”,async()=>{})

然后暴露出来

Module.exports=Router

再到app.js去启动这个路由

//主app建立新的router，装载所有子路由

let router = new Router();

router.use('/index,index.routes());

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

//加载路由中间件

app.use(router.routes());

app.use(router.allowedMethods());

# Koa视图模块化

使用koa-art-template，在app.js 引入，在其他js也能使用。