爱创课堂前端培训

# KOA

第1天课堂笔记（本课程共1天）

班级：北京前端训练营19期

讲师：李兰波

日期：2019年2月19日

爱创课堂官网 ：[www.icketang.com](http://www.icketang.com)

# 复习

Generator：它是一个新的函数类别

允许函数分段执行。

定义方式： function \* name() {} var name = function \* () {}

yield关键字： 该关键字只能够用于Generator函数内部

用于定义一段代码

function \* name() {

var a = yield code1;

yield code2;

yield code3;

}

当Generator函数执行之后，得到的是一个迭代器对象。注：不可以使用new调用Generator函数。

该迭代器对象：g

g.next(); 调用一次，表示内部的代码执行一段。从上一次的yield执行到下一个yield。

它的返回值是一个对象，拥有value属性和done属性。

value的值就是本次yield执行的返回值。

done的值是true或者false 表示是否可以继续迭代

g.next(); 该方法可以传递参数，只能够传递一个。该参数会被传递到上一次yield执行的返回结果。

Promise: 这是一个与Generator类似的异步编程的解决方案。

它的主要作用是将代码的异步编写方式变为同步编写。

初始化：var p = new Promise(handler);

handler: 一个函数，该函数有两个参数 resolve reject

通常我们会在该函数内发出异步请求。

p：是一个Promise实例。它表示本次请求的状态。它有三个状态：pending、resolved、rejected。

p.then: 是一个监听方法，用于监听p对象的状态。接收2个参数。

第一个参数是pending=>resolved时触发的回调函数

第二个参数是pending=>rejected时触发的回调函数

p的状态变为resolved还是rejected要取决于handler中的resolve还是reject执行。

p的回调函数能够接收参数，传递参数的方式就是handler中的resolve和reject执行时传递。只能够传递一个。

多个Promise依次调用：

如果想要在一个请求发送完毕之后再发出另外一个请求。则需要在前一个Promise的实例的then的参数函数中返回新的Promise。

此时，可以继续打点调用then方法。新的then方法监听的就不是原来的Promise实例。而是新的被返回的Promise实例。

多个Promise同时发出：

Promise.all: 它是一个静态方法。用于监听多个Promise。返回一个新的Promise。该Promise与之前的Promise一致。

then的触发条件发生变化：当监听的所有Promise都resolved了，才会执行then的第一个参数。只要有任何一个Promise变为rejected，则执行第二个参数。

Promise.race: 它是一个静态方法。用于监听多个Promise。返回一个新的Promise。该Promise与之前的Promise一致。

then的触发条件发生变化：只要有任何一个Promise变为rejected或者resolved。则监听该Promise。

catch方法：该方法是Promise实例的方法。用于捕获rejected和内部发生的异常。可以在连续调用then方法之后调用catch方法。此时可以只在then中书写一个参数。当发生错误时，会调用catch所接收的参数进行响应。

类：class关键字、static关键字、extends关键字

class用于定义类

定义语法： class ClassName {}

定义属性：

class ClassName {

constructor (name, age, sex) {

this.name = name;

this.age = age;

this.sex = sex;

}

}

定义方法：

class ClassName {

method() {}

}

定义静态方法：

class ClassName{

static method() {}

}

定义存取器:

class ClassName {

get methodName () {}

set methodName () {}

}

定义静态存取器

class ClassName {

static get methodName() {}

static set methodName() {}

}

extends关键字用于继承

继承语法: subClass extends superClass {

子类的内容

constructor(name, age, sex, grade) {

super(name, age, sex);

this.grade = grade;

}

}

继承的内容：所有内容

# 将ES6代码转换成ES5

webpack需要借助一些模块才能够实现转换:

|  |
| --- |
| 1. babel-loader@7 babel-core babel-preset-env |

webpack.config.js:

|  |
| --- |
| 1. module.exports = { 2. // 入口 3. entry: "./modules/main.js", 4. // 出口 5. output: { 6. path: \_\_dirname, 7. filename: "./hahaha.js" 8. }, 9. // 加载机 10. module: { 11. rules: [ 12. { 13. test: /\.es6$/, 14. loader: "babel-loader", 15. options: { 16. presets: ["env"] 17. } 18. } 19. ] 20. } 21. // 插件 22. } |

# ES6中的模块

ES6中，实现了模块化开发。

模块的定义：一个JS文件就是一个模块。

## 2.1 模块的定义

ES6给script标签，定义了一个新的属性值。叫做module。

以前的script标签:

|  |
| --- |
| 1. <script type="text/javascript"></script> |

现在可以新增一个：

|  |
| --- |
| 1. <script type="module"></script> |

一个JS文件就是一个模块，但是，既然是模块，必须要向外暴露内容。在模块中我们可以通过exports向外暴露内容。模块可以引入内容。

## 2.2 模块的特点

可以在html页面中，使用script并设置type为module定义一个模块区域。在该区域中的内容不会向外暴露。但是会执行。

|  |
| --- |
| 1. <script type="module"> 2. var a = 10; 3. console.log(a); // 输出语句 4. </script> |

输出：

|  |
| --- |
|  |

## 2.3 模块向外暴露内容的方式

第一种：

|  |
| --- |
| 1. export var a = 10;// 向外暴露a变量 |

第二种：

|  |
| --- |
| 1. var a = 10; 2. var b = 11; 3. export { 4. a, 5. b 6. } |

第三种： 提供as default 将某一项作为默认暴露项 该项只能有一次

|  |
| --- |
| 1. var a = 10; 2. var b = 11; 3. export { 4. a as default, 5. b 6. } |

## 2.4 模块引入方式

条件1：必须在type=module的script标签内使用import。如果在text/javascript中使用，会报错。

第一种： import {xxx} from path;

含义：引入path模块中暴露出来的xxx这一项

|  |
| --- |
| 1. <script type="module"> 2. import {a} from "./a.js"; 3. console.log(a) 4. </script> |

输出:

|  |
| --- |
|  |

第二种: import a from path

含义：引入path模块中暴露出来的默认的那一项并以变量a接收

如果path中并不存在默认项，则报错：

|  |
| --- |
|  |

此时，模块中必须有as default

|  |
| --- |
| 1. var a = 10; 2. var b = 11; 3. export { 4. a as default, 5. b 6. } |

第三种: import \* as obj from path;

含义：引入path中暴露出来的全部内容，并作为obj存在。

|  |
| --- |
| 1. <script type="module"> 2. import \* as obj from "./a.js"; 3. console.log(obj); 4. </script> |

输出：

|  |
| --- |
|  |

模块内部，也可以使用import

# Koa

## 3.1 简介

Express是基于NodeJS的快速、开放、极简的Web开发框架。

Koa是Express原班人马打造的另一套开发框架。

官网：<https://koa.bootcss.com/#>

简介：

Koa 是一个新的 web 框架，由 Express 幕后的原班人马打造， 致力于成为 web 应用和 API 开发领域中的一个更小、更富有表现力、更健壮的基石。 通过利用 async 函数，Koa 帮你丢弃回调函数，并有力地增强错误处理。 Koa 并没有捆绑任何中间件， 而是提供了一套优雅的方法，帮助您快速而愉快地编写服务端应用程序。

## 3.2 安装

通过各种下载程序下载即可。它跟Express一样，都属于NodeJS的第三方模块。

npm install koa

|  |
| --- |
|  |

## 3.2 搭建服务器

代码：

|  |
| --- |
| 1. // 引入Koa 2. var Koa = require("koa"); 3. // 搭建服务器 4. var app = new Koa(); 5. // 监听端口号 6. app.listen(3000); |

此时，访问：

|  |
| --- |
|  |

## 3.3 路由设置

Koa自身并没有携带任何功能。想要实现功能，请找它的插件。

路由相关的插件： koa-router

|  |
| --- |
| 1. npm install koa-router |

初始化：

|  |
| --- |
| 1. // 引入路由中间件 2. var Router = require("koa-router"); 3. // 初始化一个小router 4. var router = new Router(); |

挂载函数：

|  |
| --- |
| 1. // 挂载一个路由 2. router.get("/check", (ctx) => { 3. console.log(ctx); 4. }); |

挂载到app上：

|  |
| --- |
| 1. // 将router再整体挂载到app上 2. app.use(router.routes()); |

访问：

|  |
| --- |
|  |

注：在Express中，路由的处理函数中，有两个参数，分别是req和res。它们是Express通过添加原生对象而得到的。污染了原生的内容。

在Koa中，路由的处理函数中，只有一个对象。叫做ctx。但是它下面挂载了很多属性。这些属性都是子对象。

最重要的四个：

|  |
| --- |
| 1. ctx.request: Koa给创建的request对象 2. ctx.response: Koa给创建的response对象 3. ctx.req: 原生Node请求（req）对象 4. ctx.res: 原生Node响应（res）对象 |

## 3.4 获取GET请求的数据

通过ctx.request.query || ctx.query获取即可。

## 3.5 获取POST请求的数据

需要借助插件: koa-body

下载: npm install koa-body

使用:

|  |
| --- |
| 1. // 定义解析post请求中间件 2. var body = require("koa-body"); 3. // 初始化一个小router 4. var router = new Router(); 5. // 搭建服务器 6. var app = new Koa(); 7. // 配置 8. app.use(body()); 9. // 配置接口 10. router.post("/login", (ctx) => { 11. console.log(ctx.request.body); 12. }) |

## 3.6 静态中间件

需要借助插件：koa-static

下载：npm install koa-static

使用:

|  |
| --- |
| 1. // 定义静态中间件 2. var serve = require("koa-static"); 3. // 配置 4. app.use(serve("./web")); |

## 3.7 Cookie （ajax第5天cookie详解）

Koa自带Cookie的处理功能

ctx.cookies.set(key, value, options);

key: cookie的key

value: cookie的value

options: 配置对象

domain: cookie所在的域

httponly: 是否只由服务器端设置cookie

maxAge: 生效毫秒值

path: 生效的路径

……

设置：

|  |
| --- |
| 1. ctx.cookies.set("username", ctx.request.body.username); |

获取：

|  |
| --- |
| 1. ctx.cookies.get("username") |

## 3.8 Session

Koa支持Session的方式也是需要中间件： koa-session

下载: npm install koa-session

配置:

|  |
| --- |
| 1. // 引入session 2. var session = require("koa-session"); 3. // 生成服务器 4. var app = new Koa(); 5. // 为了设置session 必须给app添加一个keys属性 值是一个数组 数组中有一项 是secret 6. app.keys = ["asokdhoiewahoiaewhoihewaoihewaoihoiewhfehfoiewahfoewa"]; 7. // 应用 8. app.use(session({}, app)); |

经过配置之后，我们就可以向ctx.session中存储内容，也可以从ctx.session中获取内容.

network面板：

|  |
| --- |
|  |

## 3.9 EJS

Koa中的EJS需要下载模块，并引入。

下载: npm install koa-ejs

配置:

|  |
| --- |
| 1. // 引入模块 2. var Koa = require("koa"); 3. // koa-body 4. var body = require("koa-body"); 5. // 搭建服务器 6. var app = new Koa(); 7. // 引入ejs 8. var ejs = require("koa-ejs"); 9. // 路由 10. var router = new require("koa-router")(); 11. // 静态中间件 12. var serve = require("koa-static"); 14. // 配置 15. ejs(app, { 16. root: "./views", //目录 17. layout: false, //布局 18. extName: ".html" //格式 19. }) 20. app.use(body()); 21. app.use(serve("./web")); 22. router.post("/login", async (ctx) => { 23. // 获取用户名 24. var username = ctx.request.body.yonghuming; 25. // 渲染模板 26. await ctx.render("main", { 27. username 28. }) 29. }) 30. app.use(router.routes()); 31. app.listen(3000); |

**async：是一个关键字，用于声明一个函数是异步函数。**

**await：是一个关键字，只可以用于async标记的函数中。表示等待。**

# 微信小程序

前端：与用户相关的视觉设计及应用。

网站：让用户浏览器信息，以及特效较多。

PC端

移动端

Web应用程序：让用户使用的较多。

PC端

移动端

混合开发： 使用web技术，开发手机上的应用程序。

web-view： HBuilder为代表 主要是以浏览器内核开发

reactnative: 更多的是调用手机上的方法

微信小程序：嵌入在微信中的小程序。

## 4.1 微信开发者工具

开发微信小程序，需要安装微信开发者工具。

下载地址：https://developers.weixin.qq.com/miniprogram/dev/devtools/download.html

|  |
| --- |
|  |

根据需要下载。

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |