

Koa框架

王红元

coderwhy





认识Koa

■ 前面我们已经学习了express，另外一个非常流行的Node Web服务器框架就是Koa。

■ Koa官方的介绍：

- koa : next generation web framework for node.js；
- koa : node.js的下一代web框架；

■ 事实上，koa是express同一个团队开发的一个新的Web框架：

- 目前团队的核心开发者TJ的主要精力也在维护Koa，express已经交给团队维护了；
- Koa旨在为Web应用程序和API提供更小、更丰富和更强大的能力；
- 相对于express具有更强的异步处理能力（后续我们再对比）；
- Koa的核心代码只有1600+行，是一个更加轻量级的框架，我们可以根据需要安装和使用中间件；

■ 事实上学习了express之后，学习koa的过程是很简单的；



Koa初体验

■ 我们来体验一下koa的Web服务器

■ koa注册的中间件提供了两个参数：

■ ctx : 上下文 (Context) 对象；

- koa并没有像express一样，将req和res分开，而是将它们作为ctx的属性；

- ctx代表依次请求的上下文对象；

- ctx.request : 获取请求对象；

- ctx.response : 获得响应对象；

■ next : 本质上是一个dispatch，类似于之前的next；

- 后续我们学习Koa的源码，来看一下它是一个怎么样的函数；

```
const Koa = require('koa');
const app = new Koa();

app.use((ctx, next) => {
  console.log("middleware 01");
  next();
})

app.use((ctx, next) => {
  console.log("middleware 02");
  ctx.response.body = "Hello World";
})

app.listen(8000, () => {
  console.log("服务器启动成功~");
})
```



Koa中间件

■ koa通过创建的app对象，注册中间件只能通过use方法：

- Koa并没有提供methods的方式来注册中间件；
- 也没有提供path中间件来匹配路径；

■ 但是真实开发中我们如何将路径和method分离呢？

- 方式一：根据request自己来判断；
- 方式二：使用第三方路由中间件；

```
app.use((ctx, next) => {
  if (ctx.request.path === '/users') {
    if (ctx.request.method === 'POST') {
      ctx.response.body = "Create User Success~";
    } else {
      ctx.response.body = "Users List~";
    }
  } else {
    ctx.response.body = "Other Request Response";
  }
})
```



路由的使用

- koa官方并没有给我们提供路由的库，我们可以选择第三方库：koa-router

```
npm install koa-router
```

- 我们可以先封装一个 user.router.js 的文件：
- 在app中将router.routes()注册为中间件：
- 注意：allowedMethods用于判断某一个method是否支持：
 - 如果我们请求 get，那么是正常的请求，因为我们有实现get；
 - 如果我们请求 put、delete、patch，那么就自动报错：Method Not Allowed，状态码：405；
 - 如果我们请求 link、copy、lock，那么久自动报错：Not Implemented，状态码：501；

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

const userRouter = new Router({prefix: '/users'});

userRouter.get('/', (ctx, next) => {
  ctx.response.body = "user list~";
});

userRouter.post('/', (ctx, next) => {
  ctx.response.body = "create user info~";
});

module.exports = userRouter;
```

```
app.use(userRouter.routes());
app.use(userRouter.allowedMethods());
```



参数解析 : params - query

■ 请求地址 : <http://localhost:8000/users/123>

□ 获取params :

```
const userRouter = new Router({prefix: "/users"})

userRouter.get("/:id", (ctx, next) => {
  console.log(ctx.params.id);
  ctx.body = "Hello World";
})
```

■ 请求地址 : <http://localhost:8000/login?username=why&password=123>

□ 获取query :

```
app.use((ctx, next) => {
  console.log(ctx.request.query);
  ctx.body = "Hello World";
})
```



参数解析：json

■ 请求地址：<http://localhost:8000/login>

■ body是json格式：

```
{  
  "username": "coderwhy",  
  "password": "123"  
}
```

■ 获取json数据：

□ 安装依赖：npm install koa-bodyparser;

□ 使用 koa-bodyparser的中间件；

```
app.use(bodyParser());  
  
app.use((ctx, next) => {  
  console.log(ctx.request.body);  
  ctx.body = "Hello World";  
})
```



参数解析 : x-www-form-urlencoded

■ 请求地址 : <http://localhost:8000/login>

- body是x-www-form-urlencoded格式 :

POST http://localhost:8000/login

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> username	kobe	
<input checked="" type="checkbox"/> password	8888	

■ 获取json数据 : (和json是一致的)

- 安装依赖 : npm install koa-bodyparser;
- 使用 koa-bodyparser的中间件 ;

```
app.use((ctx, next) => {
  console.log(ctx.request.body);
  ctx.body = "Hello World";
})
```



参数解析：form-data

■ 请求地址：<http://localhost:8000/login>

- body是form-data格式

POST http://localhost:8000/login

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> username	lilei	
<input checked="" type="checkbox"/> password	8888	

■ 解析body中的数据，我们需要使用multer

- 安装依赖：npm install koa-multer;
- 使用 multer中间件；

```
const upload = multer({});  
  
app.use(upload.any());  
  
app.use((ctx, next) => {  
  console.log(ctx.req.body);  
  ctx.body = "Hello World";  
});
```



Multer上传文件

```
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, './uploads/')
  },
  filename: (req, file, cb) => {
    cb(null, Date.now() + path.extname(file.originalname))
  }
})
|
const upload = multer({
  storage
});

const fileRouter = new Router();

fileRouter.post("/upload", upload.single('avatar'), (ctx, next) => {
  console.log(ctx.req.file);
})
```



数据的响应

■ 输出结果：**body**将响应主体设置为以下之一：

- string：字符串数据
- Buffer：Buffer数据
- Stream：流数据
- Object|| Array：对象或者数组
- null：不输出任何内容
- 如果**response.status**尚未设置，Koa会自动将状态设置为200或204。

■ 请求状态：**status**

```
ctx.response.body = "Hello World";
ctx.body = {
  name: "why",
  age: 18,
  height: 1.88
};
ctx.body = ["abc", "cba", "nba"];
```

```
ctx.status = 201;
ctx.response.status = 204;
```



静态服务器

- koa并没有内置部署相关的功能，所以我们需要使用第三方库：

```
npm install koa-static
```

- 部署的过程类似于express：

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

const app = new Koa();

app.use(static('./build'));

app.listen(8000, () => {
  console.log("静态服务器启动成功~");
});
```



错误处理

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

const app = new Koa();

app.use((ctx, next) => {
  ctx.app.emit('error', new Error("哈哈哈"), ctx);
})

app.on('error', (err, ctx) => {
  console.log(err.message);
  ctx.response.body = "哈哈哈";
})

app.listen(8000, () => {
  console.log("错误处理服务启动成功~");
})
```



创建Koa的过程

The screenshot shows the Visual Studio Code interface with the Koa repository open. The Explorer sidebar on the left shows the project structure, including .github, benchmarks, dist, docs, lib, node_modules, test, and various configuration files. The main editor area displays the contents of application.js, which defines the Koa Application class. A yellow box highlights the constructor definition:

```
module.exports = class Application extends Emitter {  
  constructor(options) {  
    super();  
    options = options || {};  
    this.proxy = options.proxy || false;  
    this.subdomainOffset = options.subdomainOffset || 2;  
    this.proxyIpHeader = options.proxyIpHeader || 'X-Forwarded-For';  
    this.maxIpsCount = options.maxIpsCount || 0;  
    this.env = options.env || process.env.NODE_ENV || 'development';  
    if (options.keys) this.keys = options.keys;  
    this.middleware = [];  
    this.context = Object.create(context);  
    this.request = Object.create(request);  
    this.response = Object.create(response);  
    // util.inspect.custom support for node 6+  
    /*istanbul ignore else */  
    if (util.inspect.custom) {
```

The status bar at the bottom indicates the file is 32 lines long, has 13 columns, and is in UTF-8 LF JavaScript mode. The terminal shows the command "coderwhy@why koa-master %".



开启监听

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure under "KOA-MASTER". The "lib" folder is expanded, showing files like application.js, context.js, request.js, response.js, and others.
- Code Editor:** The "application.js" file is open. The code is annotated with JSDoc-style comments. A specific section of the code is highlighted with an orange rectangle:

```
listen(...args) {
  debug('listen');
  const server = http.createServer(this.callback());
  return server.listen(...args);
}
```
- Terminal:** The terminal shows the command "coderwhy@why koa-master %".
- Status Bar:** Shows the current file is "application.js — koa-master", line 33, column 29, and other status information like "Spaces: 2", "UTF-8", "LF", "JavaScript", and "Go Live".



注册中间件

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure for "KOA-MASTER". The "lib" folder contains several files: application.js, context.js, request.js, response.js, and others like .github, benchmarks, dist, docs, node_modules, test, .codecov.yml, .editorconfig, .eslintrc.yml, .gitignore, .mailmap, .npmrc, .travis.yml, AUTHORS, CODE_OF_CONDUCT.md, History.md, LICENSE, package.json, and Readme.md.
- Code Editor:** The "application.js" file is open. The code is annotated with comments explaining its purpose:

```
use(fn) {
  if (typeof fn !== 'function') throw new TypeError('middleware must be a function!');
  if (isGeneratorFunction(fn)) {
    deprecate('Support for generators will be removed in v3. ' +
      'See the documentation for examples of how to convert old middleware ' +
      'https://github.com/koajs/koa/blob/master/docs/migration.md');
    fn = convert(fn);
  }
  debug('use %s', fn._name || fn.name || '-');
  this.middleware.push(fn);
  return this;
}
```
- Terminal:** The terminal shows the command "coderwhy@why koa-master %".
- Status Bar:** Shows the file path "application.js — koa-master", line count "Ln 102, Col 6", character count "Spaces: 2", encoding "UTF-8", line separator "LF", language "JavaScript", and a "Go Live" button.



监听回调

The screenshot shows the VS Code interface with the file `application.js` open in the editor. The code implements a Koa application by creating a server and defining a handleRequest function.

```
application.js — koa-master
lib > JS application.js > <unknown> > Application > callback
120  * @return {Function}
121  * @api public
122  */
123  *
124  listen(...args) {
125    debug('listen');
126    const server = http.createServer(this.callback());
127    return server.listen(...args);
128  }
129
130  callback() {
131    const fn = compose(this.middleware);
132    if (!this.listenerCount('error')) this.on('error', this.onerror);
133
134    const handleRequest = (req, res) => {
135      const ctx = this.createContext(req, res);
136      return this.handleRequest(ctx, fn);
137    };
138
139    /**
140     * Handle request in callback.
141   */

coderwhy@why koa-master %
```

A yellow arrow points from the text "Handle request in callback." at the bottom of the code to the line `return handleRequest;`. The line `const handleRequest = (req, res) => {` is also highlighted with a yellow box.



compose方法

The screenshot shows the VS Code interface with the file `index.js` open in the editor. The code implements the `compose` function from the `koa-compose` module. The implementation checks if the middleware is an array and if each item in the array is a function. It then creates a new middleware function that dispatches through the array of functions.

```
function compose(middleware) {
  // 判断middleware如果不是数组，抛出异常
  if (!Array.isArray(middleware)) throw new TypeError('Middleware stack must be an array!')
  // middleware的原生如果不是函数，抛出异常
  for (const fn of middleware) {
    if (typeof fn !== 'function') throw new TypeError('Middleware must be composed of functions!')
  }

  /**
   * @param {Object} context
   * @return {Promise}
   * @api public
   */

  return function(context, next) {
    // last called middleware#
    let index = -1
    return dispatch(0)
    function dispatch(i) {
      if (i <= index) return Promise.reject(new Error('next() called multiple times'))
      index = i
      let fn = middleware[i]
      fn(context, dispatch)
    }
  }
}
```

The code is annotated with comments explaining its logic:

- // 判断middleware如果不是数组，抛出异常
- // middleware的原生如果不是函数，抛出异常
- // last called middleware#
- index = i
- fn(context, dispatch)

The VS Code interface includes the Explorer, Output, Terminal, Debug Console, and Problems panes. The status bar at the bottom shows the current file is `index.js` and the commit hash is `1:zsh`.



和express对比

■ 在学习了两个框架之后，我们应该已经可以发现koa和express的区别：

■ 从架构设计上来说：

■ express是完整和强大的，其中帮助我们内置了非常多好用的功能；

■ koa是简洁和自由的，它只包含最核心的功能，并不会对我们使用其他中间件进行任何的限制。

- 甚至是在app中连最基本的get、post都没有给我们提供；

- 我们需要通过自己或者路由来判断请求方式或者其他功能；

■ 因为express和koa框架他们的核心其实都是中间件：

- 但是他们的中间件事实上，它们的中间件的执行机制是不同的，特别是针对某个中间件中包含异步操作时；

- 所以，接下来，我们再来研究一下express和koa中间件的执行顺序问题；



案例实现

■ 我通过一个需求来演示所有的过程：

- 假如有三个中间件会在一次请求中匹配到，并且按照顺序执行；
- 我希望最终实现的方案是：
 - ✓ 在middleware1中，在req.message中添加一个字符串 aaa；
 - ✓ 在middleware2中，在req.message中添加一个 字符串bbb；
 - ✓ 在middleware3中，在req.message中添加一个 字符串ccc；
 - ✓ 当所有内容添加结束后，在middleware1中，通过res返回最终的结果；

■ 实现方案：

- Express同步数据的实现；
- Express异步数据的实现；
- Koa同步数据的实现；
- Koa异步数据的实现；

■ 具体的代码查看课堂演练。