一、Koa

- 安装Koa

```
$ mkdir koademo
$ cd koademo
$ npm init
$ npm i koa
```

- 新建index.js文件，添加如下代码：

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

app.listen(3000);
```

- 启动服务

```
$ node index.js
```

- 访问服务

打开 http://localhost:3000 网页上会显示"Not Found"，这是因为我们并没有告诉 Koa 应该显示什么内容。

- 说明

new Koa()：创建的对象被称为 Koa 应用对象。应用对象是带有 node http 服务的 Koa 接口，它可以处理中间件的注册，将http请求分发到中间件，进行默认错误处理，以及对上下文，请求和响应对象进行配置。

app.listen(3000)：用于启动一个服务的快捷方法，是对http.createServer 的简单包装，它实际上这样运行：`http.createServer(app.callback()).listen(3000);`

二、context

context是一个请求的上下文，该对象封装了一个传入的 http 消息，context有request和response属性，我们可以设置两个属性来处理和响应不同的请求。

- 代码

```
var Koa = require("koa")
var app = new Koa()

//中间件就是一个方法，通过app.use去调用中间件，
// 方法里面有两个参数，第一个是上下文ctx，第二个是next
//ctx里面封装好了两个重要的属性，request和response，因此方便我们对http进行处理
//next是一个方法，通过调用next()去调用下一个中间件
app.use(function (ctx, next) {
    console.log(ctx.request.path)
    //通过设置ctx的response的body值就可以返回数据到客户端
    ctx.response.body = 'Hello World';
    console.log(ctx.response.type)
})

//动态端口范围是1024-65535
app.listen("3000")
console.log('server is starting at port 3000')
```

- 启动服务

```
$ node index.js
```

- 访问服务

打开 http://localhost:3000 ，效果如下：

```
Hello World
```

- 控制台输出

```
server is starting at port 3000
/
text/plain
```

- 说明

app.use(function)：将给定的 function 当做中间件加载到应用中。

ctx：每一个请求都会创建一段上下文，在控制业务逻辑的中间件中，上下文被寄存在ctx对象中。为了使用方便，许多上下文属性和方法都被委托代理到他们的 `ctx.request` 或 `ctx.response`，比如访问 `ctx.type` 和 `ctx.length` 将被代理到 `response` 对象， `ctx.path` 和 `ctx.method` 将被代理到 `request` 对象。

三、网页模版

实际开发中，返回给用户的网页往往都写成模板文件。我们可以让 Koa 先读取模板文件，然后将这个模板返回给用户。需注意的是得指定response的type为 `"text/html"` 类型。

```
const Koa = require('koa')
const app = new Koa()
const fs = require("fs")

app.use(ctx => {
    console.log(ctx.path)
    //必须指定type，否则调用fs模板后设置为“application/octet-stream”类型，导致错误
    ctx.type = "text/html"
    //将文件作为响应体流式传输
    ctx.body = fs.createReadStream("./views/test.html")
    console.log(ctx.type)
  });

app.listen(3000)
console.log("正在监听3000端口。。。")
```

四、中间件

Koa 所有的功能都是通过中间件实现的。中间件处在 HTTP Request 和 HTTP Response 中间。

Koa 的中间件之间按照编码顺序在栈内依次执行，允许您执行操作并向下传递请求，之后过滤并逆序返回响应。响应的步骤如下所示：

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

// x-response-time

app.use(async (ctx, next) => {
  const start = Date.now();
  await next();
  const ms = Date.now() - start;
  ctx.set('X-Response-Time', `${ms}ms`);
});

// logger

app.use(async (ctx, next) => {
  const start = Date.now();
  await next();
  const ms = Date.now() - start;
  console.log(`${ctx.method} ${ctx.url} - ${ms}`);
});

// response

app.use(async ctx => {
  ctx.body = 'Hello World';
});

app.listen(3000);
```


```
var Koa = require("koa")
var app = new Koa()

//es6新语法:
// 函数名 = (参数) => {}
app.use((ctx,next) => {
    var start = Date.now()
    next()
    console.log(`${ctx.url} ${Date.now() - start}`)
})
app.use((ctx, next) => {
    ctx.response.body = "hahah"
})

//动态端口范围是1024-65535
app.listen("3000")
```

看看下面的例子充分了解中间件的执行流程。

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

const one = (ctx, next) => {
    console.log('>> one');
    next();
    console.log('<< one');
}

const two = (ctx, next) => {
    console.log('>> two');
    next();
    console.log('<< two');
}

const three = (ctx, next) => {
    console.log('>> three');
    next();
    console.log('<< three');
}

app.use(one);
app.use(two);
app.use(three);

app.listen(3000);
```

输出如下

```
>> one
>> two
>> three
<< three
<< two
<< one
```

五、异步中间件

由async标记的函数称为异步函数，在异步函数中，可以用await调用另一个异步函数。使用await时，它所在的方法必须使用关键字async。

```
var Koa = require("koa")
var app = new Koa()
```

```
app.use(async (ctx,next) => {
    var start = Date.now()
    await next()
    console.log(`${ctx.url} ${Date.now() - start}`)
})

app.use(async (ctx, next) => {
    ctx.response.body = "hahah"
})

//动态端口范围是1024-65535
app.listen("3000")
```

六、原生路由

在Koa中使用的原生的路由代码非常累赘，不便于扩展，实现如下。

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

app.use((ctx, next) => {
    console.log("%s %s", ctx.method, ctx.url)
})

app.use((ctx, next) => {
    if (ctx.request.path === '/') {
        ctx.response.body = 'index page';
    } else {
        next();
    }
});

app.use((ctx, next) => {
    if (ctx.request.path === '/test') {
        ctx.response.body = 'TEST page';
    } else {
        next();
    }
});

app.use((ctx, next) => {
    if (ctx.request.path === '/error') {
        ctx.response.body = 'ERROR page';
    } else {
        next();
```

```
    }
});
app.listen(3000)
console.log("正在监听3000端口。。。")
```

七、koa-router

原生路由用起来不太方便，我们可以使用封装好的koa-router模块。注意使用的是`route.routes()`绑定到中间件。

引入koa-router

```
$ npm i koa-router
```

```
var Koa = require("koa")
var app = new Koa()
//引入koa-router的时候要加上 ()
var router = require("koa-router")()

router.get("/hello", function (ctx,next) {
    ctx.response.body = "hello"
})

router.get("/bye", function (ctx, next) {
    ctx.response.body = "bye"
})

//将router路由注册到中间件
app.use(router.routes())

//动态端口范围是1024-65535
app.listen("3000")
```

八、重定向

在哪些情况下会使用到重定向呢？比如：

- 由于后台系统升级，对之前的某些页面不支持了，但是用户还可能会继续访问，因此就可以使用重定向到新的api上。
- 完成某个操作后自动跳转到上一页、首页、其它页面等，如付费成功、登录成功等。

语法：`ctx.response.redirect('/bye');`

```
var Koa = require("koa")
var app = new Koa()
//引入koa-router的时候要加上 ()
var router = require("koa-router")()

router.get("/hello", function (ctx,next) {
    ctx.response.body = "hello"
})

router.get("/bye", function (ctx, next) {
    //重定向的用处：比如登录完成以后可以重定向到首页
    ctx.response.redirect("/hello")
})

//将router路由注册到中间件
app.use(router.routes())

//动态端口范围是1024-65535
app.listen("3000")
```

在nodejs访问url中有中文时，要用全局函数encodeURIComponent(string) 对其进行编码。

九、获取get请求参数

客户端在请求获取服务的数据时，url上通常会携带参数，那么服务端如何获取get请求的参数呢?

如请求地址`http://127.0.0.1:3000/hello/kongyixueyuan`，那么获取方式是`ctx.params.name;`

如请求地址`http://127.0.0.1:3000/bye?name=kongyixueyuan`，那么获取方式是`ctx.query.name;`

注意：调用params获取参数的时候，params不是request的属性，须直接通过ctx调用获取。

具体使用方法如下

```
var Koa = require("koa")
var app = new Koa()
//引入koa-router的时候要加上 ()
var router = require("koa-router")()

router.get("/bye", function (ctx,next) {
    //获取请求参数
    var params = ctx.request.querystring
    console.log(params)

    //获取参数字段的数据
    var name = ctx.request.query.name
    console.log(name)
```

```
    ctx.response.body = "bye"
})

router.get("/hello/:name", function (ctx, next) {
    //调用params获取参数的时候，params不是request的属性，直接通过ctx调用
    var name = ctx.params.name
    console.log(name)
    ctx.response.body = "hello"
})

//将router路由注册到中间件
app.use(router.routes())

//动态端口范围是1024-65535
app.listen("3000")
```

十、获取post请求参数

get请求的参数附带在了url上，post请求的参数在请求体body里，所以要揭晓body的数据，需要使用到插件`koa-body`，通过`ctx.request.body.name;`获取参数。具体使用方法如下

引入koa-router

```
$ npm i koa-router
```

```
var Koa = require("koa")
var app = new Koa()
var router = require("koa-router")()
var koaBody = require("koa-body")

//通过命令使用curl插件模拟调用一个post请求
//curl -H "Content-Type:application/json" -X POST --data '{"username":
"kongyixueyuan"}' http://localhost:3000/hello
router.post("/hello", function (ctx,next) {
    var body = ctx.request.body
    ctx.response.body = "hello world"
    console.log(body)
    console.log(body.username)
})

//设置multipart: true，支持多个参数
app.use(koaBody({
    multipart: true
}))
```

```
app.use(router.routes())
//动态端口范围是1024-65535
app.listen("3000")
```

使用curl进行post请求

```
curl -X POST --data "username=kongyixueyuan" http://localhost:3000/hello
```

十一、加载静态资源

加载静态资源，如图片、字体、样式表、脚本等。注意，编码指定静态资源的路径是相对于./static的路径。

引入koa-static

```
$ npm i koa-static
```

```
var Koa = require("koa")
var app = new Koa()
//引入koa-router的时候要加上 ()
var router = require("koa-router")()
var static = require("koa-static")
var path = require("path")

router.get("/hello", function (ctx,next) {
    ctx.response.body = "<html><a href='/0.png'>点我</a></html>"
})

router.get("/bye", function (ctx, next) {
    ctx.response.body = "bye"
})

//静态资源的路径是相对于./static的路径
app.use(static(path.join(__dirname, "./static")))

//将router路由注册到中间件
app.use(router.routes())

//动态端口范围是1024-65535
app.listen("3000")
```

十二、模版引擎

模版引擎ejs需要配上模板渲染中间件koa-views使用。若需支持其他后缀的文件，须将文件扩展名映射到引擎。

引入koa-views与ejs。

```
$ npm i koa-views
$ npm i ejs
```

```
var Koa = require("koa")
var app = new Koa()
//引入koa-router的时候要加上 ()
var router = require("koa-router")()
//views是指定ejs模板引擎的路径
var views = require("koa-views")
var path = require("path")

router.get("/hello", async function (ctx,next) {
    //将json里面的值替换为文件里面的变量
    //go 里面使用的是template模板，他们的功能类似
    var name = "孔壹学院"
    await ctx.render("homeejs", {
        name,
        "address":"北京市海淀区"
    })
})

router.get("/bye", async function (ctx, next) {
    await ctx.render("home.html", {
        "name":"孔壹学院"
    })
})

app.use(views(
    //默认去views下面获取ejs后缀的文件
    //如果是其他类型的文件需要指定文件类型
    path.join(__dirname, "./views"),
    {extension:"ejs", map:{html:"ejs"}}
))

//将router路由注册到中间件
app.use(router.routes())

//动态端口范围是1024-65535
app.listen("3000")
```

testejs.ejs

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>

    <style>
        div{
            background-color: deepskyblue;
        }
    </style>
</head>
<body>

<div>姓名：<%= name %> 位置：<%= address %></div>
</body>
</html>
```

启动服务后访问http://localhost:3000/hello 页面显示如下

```
姓名：孔壹学院 位置：北京市海淀区
```