1. 中间件机制中setTimeout()执行next
2. 回调地狱之callback处理异步
3. promise.then处理异步
4. async+await处理异步

## 在中间件机制中settimeout()执行next(),会出现不会跳中间件的情况

const Koa = require('Koa')

const app = new Koa()

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

ctx.body = "1"

setTimeout(()=>{

next()

},2000)

ctx.body += "2"

})

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

ctx.body += "3"

next()

ctx.body += "4"

})

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

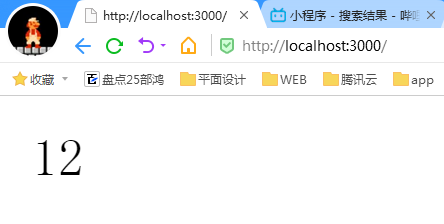
ctx.body += "5"

next()

ctx.body += "6"

})

app.listen(3000)



没有异步处理的时候是这样的：

function ajax(){

setTimeout(()=>{

console.log("你好！")

},2000)

}

ajax()

console.log("执行结束")



两个东西同时进行了，我们要异步处理：先处理完了“你好”，再去处理“执行结束”

## 回调地狱时代——callback法

将执行结束作为参数传进去



但是问题来了 ，如果很多的嵌套，就特别复杂！

## ES6 promise异步处理——孙悟空猪八戒沙僧加盟问题

function add(name){

return new Promise((reslove,reject)=>{

setTimeout(()=>{

reslove(name+'加进来了')

},2000)

})

}

add('孙悟空').then((name)=>{

console.log(name)

return add('猪八戒')

})

.then((name)=>{

console.log(name)

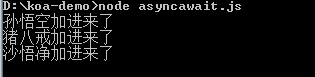
return add('沙悟净')

})

.then((name)=>{

console.log(name)

})



分析：相比callback的传参异步处理，这个不需要传参，而是通过promise返回之后的then进行异步过渡，达到处理异步的预期，关键是整体上它是并联的，不是嵌套的，代码可读性和操作性大大提高。

1. async+await优雅处理异步

function add(name){

return new Promise((reslove,reject)=>{

setTimeout(()=>{

reslove(name+'加进来了')

},2000)

})

}

async function start(){

const person1 = await add('孙悟空')

console.log(person1)

const person2 = await add('猪八戒')

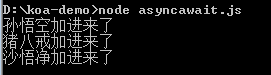
console.log(person2)

const person3 = await add('沙悟净')

console.log(person3)

}

start()



## 过关斩将：自己定义一个中间件的计时器koa-logger.js

## .koa-logger.js

module.exports = async(ctx,next)=>{

const start = new Date().getTime()

await next()

const end = new Date().getTime()

console.log(ctx.request.url,end-start,ctx.body.length)

}

## .server.js

const Koa = require('Koa')

const app = new Koa()

const koaLog = require('./koa-logger.js')

function add(name){

return new Promise((reslove,reject)=>{

setTimeout(()=>{

reslove(name+'加进来了')

},2000)

})

}

app.use(koaLog)

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

ctx.body = "1"

await next()

ctx.body += "2"

})

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

ctx.body += "3"

await add()

await next()

ctx.body += "4"

})

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

// ctx.body += "5"

// await next()

// ctx.body += "6"

// })

app.listen(3000)