第5章

云 函 数

云函数是一段运行在云端的代码，无须管理服务器，在开发工具内编写、一键上传部署即可运行后端代码。云开发中的云函数可让用户将自身的业务逻辑代码上传，并通过云开发的调用触发函数，从而实现后端的业务运作。用户可在客户端直接调用云函数，也可以在云函数之间实现相互调用，云函数现在唯一支持的语言为 Node.js 8.9。

5.1 云函数发送 HTTP 请求

在微信小程序和云函数中都可以发送 HTTP 请求，那么什么时候要用云函数发送请求呢？微信对微信小程序端发送 HTTP 请求做了限制：

(1) 微信小程序端使用 wx.request 可以发起一个 HTTP 请求，一个微信小程序被限制为同时只有 5 个网络请求，而使用云函数发送 HTTP 请求没有这个限制；

(2) 微信小程序端发送 HTTP 请求域名必须经过 ICP 备案，而云函数不需要进行域名备案；

(3) 微信小程序端发送请求域名只支持 HTTPS，且要在微信公众平台进行服务器域名配置，而云函数支持 HTTP/HTTPS，不需要进行服务器域名配置。

如果开发者自己开发后端并搭建服务器，且在开发阶段还没有进行域名备案，可以设置不校验域名和 HTTPS 证书，如图 5-1 所示，在微信开发者工具中查看，单击“详情”按钮，然后选择“本地设置”子页面，勾选“不校验合法域名、web-view(业务域名)、TLS 版本以及 HTTPS 证书”复选框。

本节介绍在云函数中使用 got 发送 HTTP 请求，got 是为 Node.js 获取人性化且功能强大的 HTTP 请求库，而云函数使用的就是 Node.js。接下来演示如何在云函数中使用 got 发送 HTTP 请求。

1. 在云函数中安装 got 依赖库

在微信开发者工具中，右击目录 cloudfunctions，在弹出的快捷菜单中选择“新建 Node.js 云函数”选项，如图 5-2 所示，输入云函数名称，比如 HTTP，这样一个云函数就建好了。

图 5-1　设置不校验域名和 HTTPS 证书

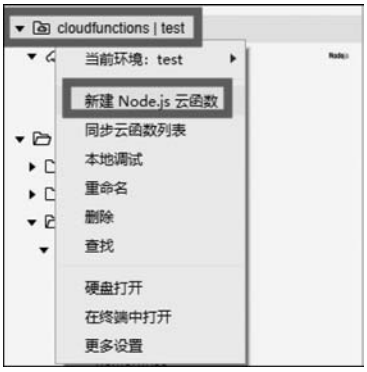

图 5-2　新建 Node.js 云函数

新建的 Node.js 云函数其实是存在于本地目录中的，在本地目录中编写完云函数，再把云函数上传到云端。上传云端的方法有两种："上传并部署：云端安装依赖（不上传 node_modules）"和"上传并部署：所有文件"。第一种方法只会上传本地的代码，安装的依赖库不会上传；第二种方法是把本地的代码和依赖库一同上传至云端。那么如果采用第一种方法，云端是怎么知道要安装哪个依赖库？答案是云端会根据云函数 package.json 文件中的 dependencies 安装相应的依赖库。云函数除了在本地调试时需要在本地目录安装依赖库以外，其他都是在云端调用的，因此通常情况下只要安装云端的依赖库就可以了，本地目录不需要安装依赖库。

根据前面的介绍，在云端安装依赖库，只需要在本地目录云函数 package.json 文件中的 dependencies 写入要安装的库就可以了，有两种方法实现：

方法一：通过 npm 安装依赖库，比如安装 got 依赖库，如图 5-3 所示。

```
npm install got
```

安装方法为：右击云函数目录，在弹出的快捷菜单中选择"在终端中打开"选项，随后在打开的控制台中输入 npm install got。该方法会在本地目录安装 got，同时写入 package.json 文件中的 dependencies。

方法二：开发者直接手动在 package.json 文件中的 dependencies 中写入需要安装的依赖库。如果不知道依赖库的版本，则可以直接写"latest"，表示最新版，比如""dependencies":{"got":"^9.6.0"}"，如图 5-4 所示。该方法不会在本地目录安装依赖库。需要说明的是，作者在写本书时最新版本 got 库不支持 Node.js 8.9，因此读者如果需要使用 got 库，则可以使用上面的 9.6.0 版本。

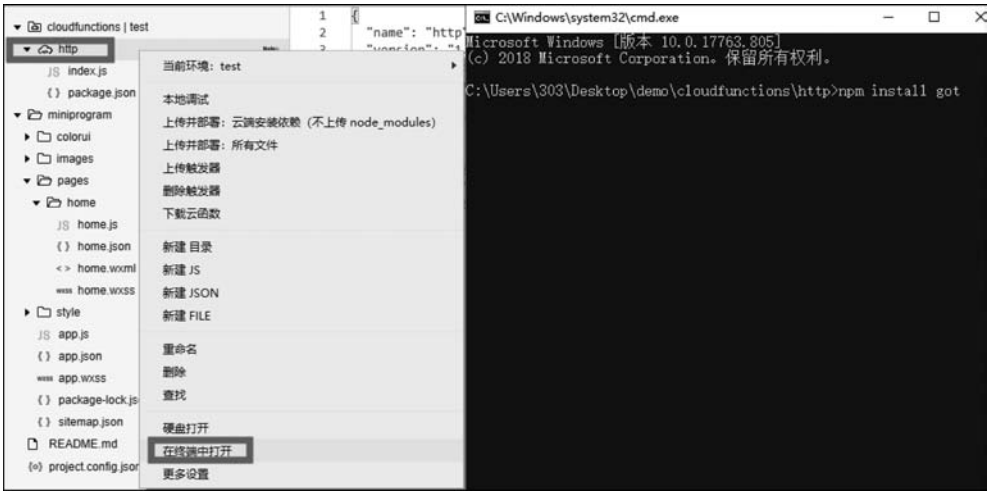

图 5-3　通过 npm 安装依赖库

图 5-4　package.json 文件中的 dependencies 依赖库

方法一通过 npm 会在本地安装依赖库，方法二则不会在本地安装依赖库，两种方法都会在 package.json 文件中的 dependencies 中写入需要安装的依赖库。

在本地配置好云函数的依赖库后，右击云函数目录，在弹出的快捷菜单中选择“上传并部署：云端安装依赖（不上传 node_modules）”选项，云端会根据 package.json 文件中的 dependencies 自动安装依赖库。如果用户不需要在本地调试云函数，则建议直接采用方法二。

2. 在云函数中使用 got 发送 HTTP 请求

在云函数中使用 got 发送 GET 请求代码如下：

```
const got = require('got');
exports.main = async (event, context) =>{
  let getResponse = await got('httpbin.org/get')
  return getResponse.body
}
```

在云函数中使用 got 发送 POST 请求代码如下：

```
const got = require('got');
exports.main = async(event, context) =>{
  let getResponse = await got('httpbin.org/get')
  let postResponse = await got('httpbin.org/post', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      title: 'title test',
      value: 'value test'
    })
  })
  return postResponse.body
}
```

3. 在微信小程序中调用云函数

在微信小程序中调用云函数，微信小程序页面 home.wxml 代码如下：

```
<cu-custom bgColor="bg-gradual-pink" isBack="{{true}}">
    <view slot="backText">返回</view>
    <view slot="content">导航栏</view>
</cu-custom>
<button bindtap='http'> http </button>
```

相应的 home.js 代码如下：

```
Page({
  http: function (event) {
    wx.cloud.callFunction({
      name: 'http'
    }).then(res =>{
      console.log(res.result)
    })
  }
})
```

单击微信小程序 http 按钮，就可以调用云函数发送 HTTP 请求，本实例中发送 GET 请求的结果如图 5-5 所示。

发送 POST 请求结果如图 5-6 所示。

除了使用 got 依赖库可以发送 HTTP 请求外，还可以使用 request-promise 和 axios 依赖库来发送 HTTP 请求，接下来介绍如何使用 axios 来发送 HTTP 请求，request-promise 的用法与 got 和 axios 类似，这里不再详细介绍。

4. 在云函数中安装 axios 依赖库

在云函数 http 的 package.json 文件中的 dependencies 中写入需要安装的依赖库：

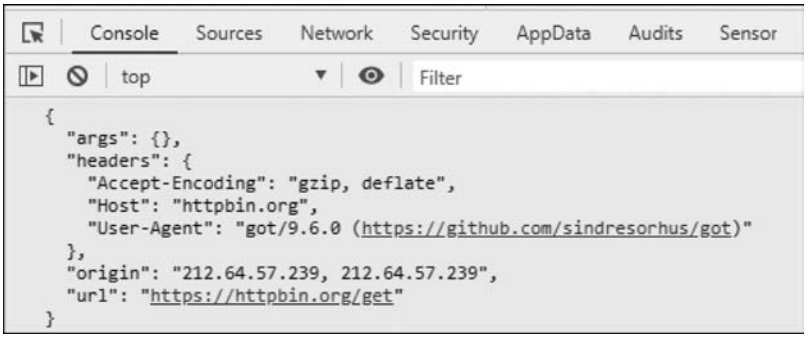

图 5-5 发送 GET 请求结果

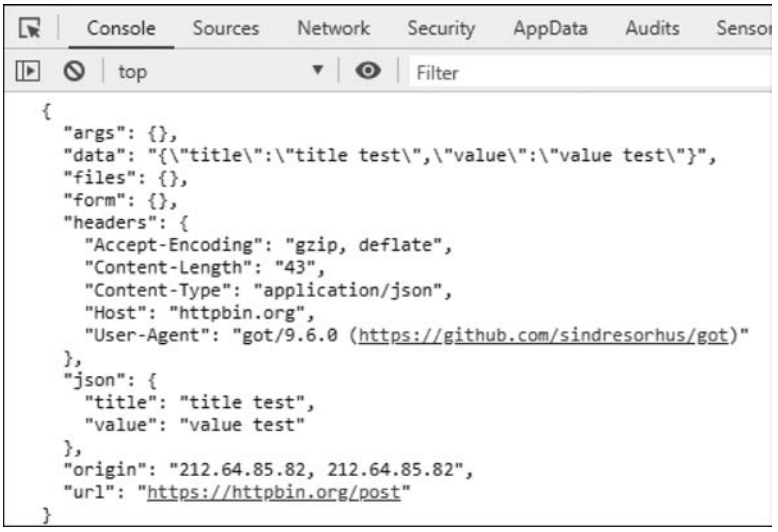

图 5-6 发送 POST 请求结果

```
"dependencies": {
  "axios": "latest"
}
```

5. 在云函数中使用 axios 发送 HTTP 请求

在云函数中使用 axios 发送 GET 请求代码如下：

```
const axios = require('axios');
exports.main = async (event, context) =>{
  let getResponse = await axios.get('http://httpbin.org/get')
  return getResponse.data
}
```

在云函数中使用 got 发送 POST 请求代码如下：

```
const axios = require('axios');
exports.main = async (event, context) =>{
```

```
3     let postResponse = await axios.post('http://httpbin.org/post', {
4       title: 'title test',
5       value: 'value test'
6     });
7     return postResponse.data
8   }
```

axios 和 got 发送 HTTP 请求的区别：

(1) 使用 axios 必须补全路径，例如采用路径 httpbin.org/post，系统会提示错误："Error: connect ECONNREFUSED 127.0.0.1:80"；

(2) got 返回的结果为 postResponse.body，而 axios 返回的结果为 postResponse.data。

6. 云函数解析豆瓣图书信息

有较多的应用需要用到豆瓣的图书信息，但是目前豆瓣图书 API 处于停用状态，而国内很多图书 API 要么需要收费，要么就是服务器不稳定，这里使用 HTTP 请求读取豆瓣图书信息，随后对 HTML 进行解析获取图书信息，例如根据 ISBN 在豆瓣搜索的网址为 https://book.douban.com/isbn/9787302224464，网址中最后 13 位数字为图书的 ISBN 号，搜索结果如图 5-7 所示。在 Chrome 浏览器中右击，在弹出的快捷菜单中选择"查看网页源代码(V)"选项，查看网页的源代码，本节使用云函数来解析豆瓣图书信息。

图 5-7　豆瓣网根据 ISBN 搜索图书信息

因为解析豆瓣图书信息不需要使用云数据库和云存储，所以作者是直接在 Visual Studio Code(VS Code)中调试 Node.js 代码，调试界面如图 5-8 所示。

图 5-8 VS Code 调试 Node.js 代码界面

这里用到了 got 和 cheerio 依赖库，cheerio 是 Node.js 的抓取页面模块，是为服务器特别定制的，快速、灵活、实施的 jQuery 核心实现，适合各种 Web 爬虫程序。安装 got 和 cheerio 命令为“npm install got; npm install cheerio”。

在微信小程序云函数中使用 got 和 cheerio 只需要在 package.json 文件中的 dependencies 项添加依赖项：

```
"dependencies": {
  "got": "^9.6.0",
  "cheerio":"latest"
}
```

云函数 douban 中 index.js 代码如下：

```
const got = require('got')
const cheerio = require('cheerio');
exports.main = async(event, context) =>{
```

```
4    var isbn = event.isbn
5    var bookinfo = {}
6    var res = await got('https://book.douban.com/isbn/' + isbn)
7      .then(response =>{
8        html = response.body
9        const $ = cheerio.load(html)
10       title = $('#wrapper h1').text().replace(/ /g, '').replace(/\n/g, '')   //获取书名
11       image_url = $('.nbg').attr('href')                                     //获取图片地址
12       introduce = $('#link-report div').last().text().replace(/ /g, '');     //获取内容简介
13       info = $('#info').text().replace(/ /g, '');                            //获取图书信息
14       for (var i = 0; i < 12; i++) {
15         info = info.replace("\n", '').replace(":\n", ':').replace("//", ';');
16       }
17       //console.log(info)
18       info = info.split('\n')
19       var str = "";
20       for (var i = 0, j = 0; i < info.length; i++) {
21         if (info[i] != '' && info[i].indexOf(':') != -1) {
22           if (j == 0)
23             info[i] = '"' + info[i].replace(":", '":"') + '"';
24           else
25             info[i] = ',"' + info[i].replace(":", '":"') + '"';
26           j++
27           str = str + info[i]
28         }
29       }
30       str = '{' + str + '}'
31       console.log(str)
32       bookinfo = JSON.parse(str)
33       bookinfo.title = title
34       bookinfo.url = image_url
35       bookinfo.内容简介 = introduce
36       bookinfo.status = 1 //获取图书信息成功
37       console.log(bookinfo)
38       return bookinfo
39     })
40     .catch(error =>{
41       //console.log(error.response.body);
42       console.log("豆瓣没有收录此书")
43       bookinfo.status = 0 //获取图书信息失败
44       return bookinfo
45     });
46   return res
47 }
```

这里采用 cheerio 来解析 HTML，cheerio 的使用本书不做重点介绍，读者可阅读 cheerio 官方文档（https://cheerio.js.org/），在解析 HTML 页面时，读者应该对照查看 HTML 网页源代码。代码第 6 行向豆瓣网发送 HTTP 请求，返回页面信息；代码第 10 行解析图书书名；代码第 11 行解析图书封面图片地址；代码第 12 行解析图书内容简介；代

码第 13～31 行获取图书信息(作者、出版社、出版年、页数、定价等),因为读取的内容为字符串,需要修改成特定的格式,其中第 14～16 行去掉文本中不需要的字符;代码第 32 行把字符串转换为 JSON 格式。

在微信小程序中调用云函数,微信小程序页面 home.wxml 代码如下:

```
<cu-custom bgColor="bg-gradual-pink" isBack="{{true}}">
    <view slot="backText">返回</view>
    <view slot="content">导航栏</view>
</cu-custom>
<button bindtap='http'>http</button>
```

相应的 home.js 代码如下:

```
Page({
  http: function (event) {
    wx.cloud.callFunction({
      name: 'douban',
      data:{
        isbn:"9787505722835"
      }
    }).then(res =>{
      console.log(res.result)
    })
  }
})
```

单击微信小程序 http 按钮,就可以调用云函数解析图书对应 ISBN 号的图书信息,本实例中微信小程序调用云函数根据 ISBN 查询图书信息结果如图 5-9 所示。

图 5-9 微信小程序根据 ISBN 查询图书信息结果

5.2 云函数将数据库数据生成 Excel

有时管理员需要把数据库中的数据导出来,方便管理员查看、核对数据,本节演示如何在云函数中使用 node-xlsx 类库把数据库数据生成 Excel。整个过程分为以下几个过程:

(1) 创建云函数,并安装 node-xlsx 类库;

(2) 读取数据库集合中的所有数据;

(3) 通过 node-xlsx 类库把数据写入 Excel;

(4) 把生成的 Excel 文件上传到云存储,供微信小程序端下载浏览。

下面分别介绍。

1. 创建云函数,并安装 node-xlsx 类库

这里采用 5.1 节中介绍的第二种方法安装依赖库,用户生成云函数后,在 package.json 文件中的 dependencies 项添加依赖项:

```
"dependencies": {
  "wx-server-sdk": "latest",
  "node-xlsx": "latest"
}
```

2. 读取数据库集合中的所有数据

因为数据库集合中存在较多记录,而采用微信小程序每次只能读取数据库集合中 20 条记录,云函数每次只能读取数据库集合中 100 条记录,因为有默认 100 条的限制,所以很可能一个请求无法取出所有数据,需要分批次取。微信小程序开发文档中给出了 Collection.get()取集合所有数据的方法,这里只需要把数据库集合修改成想要读取的数据库集合即可,代码如下:

```
1  let databasename = event.databasename
2  //先读取出集合记录总数
3  const countResult = await db.collection(databasename).count()
4  const total = countResult.total
5  //计算需分几次读取
6  const batchTimes = Math.ceil(total / 100)
7  //承载所有读操作的 promise 的数组
8  console.log(total, batchTimes)
9  const tasks = []
10 for (let i = 0; i < batchTimes; i++) {
11   const promise = db.collection(databasename).skip(i * MAX_LIMIT).limit(MAX_LIMIT).get()
12   tasks.push(promise)
13 }
14 let dbdata = (await Promise.all(tasks)).reduce((acc, cur) => {
15   return {
16   data: acc.data.concat(cur.data),
17   errMsg: acc.errMsg,
18 }
19 })
```

代码第 1 行对应读取的数据库集合名称,第 2、3 行读取集合中记录总数,因为云函数读取数据库集合时每次最多只能读取 100 条记录,所以代码第 6 行计算需要分几次读取集合中的数据,代码第 9～13 行分批读取集合中的数据。代码第 14～18 行把所有分批读取的数

据进行汇总。

3. 通过 node-xlsx 类库把数据写入 Excel

可以使用 node-xlsx 中的 xlsx.build([{name: excelname, data: exceldata}]) 生成 Excel 文件，其中 name 参数为需要生成的 Excel 文件名称，data 是一个二维数组，每个元素都对应一个单元格。

```
1  //1. 定义 Excel 文件名
2  let excelname = Math.floor(10000 * Math.random()) + 'test.xlsx'
3  //2. 定义存储的数据
4  let exceldata = [];
5  let row = ['设备 ID', '设备名称', '领用人', '产商', '存放地', '价格', '购买日期', '供应商']; //表属性
6  exceldata.push(row);
7  //console.log(dbdata.data)
8  for (let item of dbdata.data) {
9    let arr = [];
10   arr.push(item.deviceid);
11   arr.push(item.devicename);
12   arr.push(item.deviceuser);
13   arr.push(item.manufacturer);
14   arr.push(item.place);
15   arr.push(item.price);
16   arr.push(item.purchasedate);
17   arr.push(item.supplier);
18   exceldata.push(arr)
19 }
20 // console.log(exceldata)
21 //3. 把数据保存到 Excel 文件中
22 var buffer = await xlsx.build([{
23   name: excelname,
24   data: exceldata
25 }]);
```

代码第 2 行采用随机数＋test.xls 方式生成 Excel 文件名称，防止多次生成 Excel 文件在存储中出现命名冲突。这里采用逐行数据写入 exceldata，其中第 5 行写入 Excel 第一行数据的 title；代码第 8～19 行把每条记录转换成 Excel 中的一行数据；代码第 22～25 行生成 Excel 文件。

4. 把生成的 Excel 文件上传到云存储

最后为了提供微信小程序用户下载，需要把生成的 Excel 文件上传到云存储，代码如下：

```
1  await cloud.uploadFile({
2      cloudPath: excelname,
3      fileContent: buffer,     //excel 二进制文件
4    })
```

完整的数据库集合生成 Excel 文件的云函数代码如下：

```
const cloud = require('wx-server-sdk')
cloud.init()
const db = cloud.database()
var xlsx = require('node-xlsx');
const MAX_LIMIT = 100
exports.main = async (event, context) =>{
  let databasename = event.databasename
  //先读取出集合记录总数
  const countResult = await db.collection(databasename).count()
  const total = countResult.total
  //计算需分几次读取
  const batchTimes = Math.ceil(total / 100)
  //承载所有读操作的 promise 的数组
  console.log(total, batchTimes)
  const tasks = []
  for (let i = 0; i < batchTimes; i++) {
    const promise = db.collection(databasename).skip(i * MAX_LIMIT).limit(MAX_LIMIT).get()
    tasks.push(promise)
  }
  let dbdata = (await Promise.all(tasks)).reduce((acc, cur) =>{
    return {
      data: acc.data.concat(cur.data),
      errMsg: acc.errMsg,
    }
  })

  //1. 定义 Excel 文件名
  let excelname = Math.floor(10000 * Math.random()) + 'test.xlsx'
  //2. 定义存储的数据
  let exceldata = [];
  let row = ['设备 ID', '设备名称', '领用人', '产商', '存放地', '价格', '购买日期', '供应商'];//表头
  exceldata.push(row);
  //console.log(dbdata.data)
  for (let item of dbdata.data) {
    let arr = [];
    arr.push(item.deviceid);
    arr.push(item.devicename);
    arr.push(item.deviceuser);
    arr.push(item.manufacturer);
    arr.push(item.place);
    arr.push(item.price);
    arr.push(item.purchasedate);
    arr.push(item.supplier);
    exceldata.push(arr)
  }
  // console.log(exceldata)
  //3. 把数据保存到 Excel 文件中
  var buffer = await xlsx.build([{
```

```
    name: excelname,
    data: exceldata
  }]);
  //4. 把 Excel 文件保存到云存储中
  return await cloud.uploadFile({
    cloudPath: excelname,
    fileContent: buffer,     //Excel 二进制文件
  })
}
```

这里需要说明的是，调用云函数把数据库集合数据生成 Excel 文件之前需要先建立相应的数据库集合并插入数据记录，为了便于演示，这里新建了 device 数据库集合，数据从 device.json 文件中导入 104 条设备记录。调用云函数成功后，打开云开发控制台，选择“存储”→“存储管理”选项，可以看到新生成的 Excel 文件，如图 5-10 所示。

图 5-10　云函数生成的 Excel 文件

最后在微信小程序端调用云函数就可以下载并浏览 Excel 文件了，可以在微信小程序端加一个按钮，按钮的 bindtap 事件 generateExcel 代码如下：

```
generateExcel: function (event) {
    wx.cloud.callFunction({
      name: 'excel',
      data:{
        databasename:"device"
      }
    }).then(res =>{
      console.log(res.result.fileID)
      wx.showLoading({
        title: '文档打开中...',
      })
      wx.cloud.downloadFile({
        fileID: res.result.fileID
      }).then(res =>{
        // get temp file path
        console.log(res.tempFilePath)
        const filePath = res.tempFilePath
        wx.openDocument({
          filePath: filePath,
```

```
20            success: res =>{
21              console.log('打开文档成功')
22              wx.hideLoading()
23            }
24          })
25        }).catch(error =>{
26          // handle error
27        })
28      })
29    }
```

代码第2～7行调用云函数，云函数名称为excel，传递参数数据库集合名称device。调用云函数生成Excel文件后，通过wx.cloud.downloadFile()从云存储中下载Excel文档，随后使用wx.openDocument()打开下载的文档进行查看。

有些读者可能会尝试通过云函数把Excel数据导入到云数据库中，目前微信小程序端和云函数端API接口还不支持数据库数据导入功能(通过cloudbase-manager-node依赖库目前支持数据库导入功能，不过仅限于导入CSV和JSON格式的文件，详情见https://github.com/TencentCloudBase/cloudbase-manager-node)。Collection.add()一次操作只能插入一条记录，不支持批量插入数据，因此插入记录的方式为："打开数据库"→"插入一条记录"→"关闭数据库"→"打开数据库"→"插入一条记录"→"关闭数据库"，如此往复，当数据量比较大时，逐条记录插入数据库操作的资源消耗是比较大的，云函数会提示错误：{"errCode":-501004,"errMsg": "\[LimitExceeded.NoValidConnection] Connection num overrun.}。当数据量比较大时，通过云函数从Excel中读取数据，然后用Collection.add()逐条插入数据库的方法并不可行，而且默认情况下云函数超时时间为3s(云开发控制台上可以设置云函数超时时间最长为20s)，也就是说云函数在3s内没有返回结果就会超时；如果读者一定要从Excel中导入数据库，目前可以通过HTTP API数据库导入接口导入数据，见15.2.4节通过HTTP API接口批量插入数据。

5.3 本地调试

云开发提供了云函数本地调试功能，在本地提供了一套与线上一致的Node.js云函数运行环境，让开发者可以在本地对云函数调试，使用本地调试可以提高开发、调试效率。

- 单步调试/断点调试：比起通过云开发控制台中查看线上打印的日志的方法进行调试，使用本地调试后可以对云函数Node.js实例进行单步调试/断点调试。
- 集成微信小程序测试：在模拟器中对微信小程序发起的交互点击等操作如果触发了开启本地调试的云函数，会请求到本地实例而不是云端。
- 优化开发流程，提高开发效率：调试阶段不需上传部署云函数，在调试云函数时，相对于不使用本地调试时的调试流程("本地修改代码"→"上传部署云函数"→"调用")的调试流程，省去了上传等待的步骤，改成只需"本地修改"→"调用"的流程，大大提高开发、调试效率。

同时，本地调试还定制化提供了特殊的调试能力，包括Network面板支持展示HTTP

请求和云开发请求、调用关系图展示、本地代码修改时热重载等能力，帮助开发者更好地开发、调试云函数。建议开发者在开发阶段和上传代码前先使用本地调试测试通过后再上线部署。

在 cloudfunctions 文件上右击，在弹出的快捷菜单中选择“本地调试”选项，开发者可通过右击云函数名唤起本地调试界面。在本地调试界面中点击相应云函数并勾选“开启本地调试”复选框方可进行该云函数的本地调试，如图 5-11 所示。取消勾选“开启本地调试”复选框后可关闭对该云函数的本地调试。若云函数中使用到 npm 模块，需在云函数本地目录安装相应依赖才可正常使用云函数本地调试功能。在开启本地调试的过程中，系统会检测该云函数本地是否已安装了 package.json 中所指定的依赖库，如果没有安装相应的依赖库则会给出错误，出现错误提示：“Error:Cannot find module 'wx-server-sdk'…”。因为在进行云函数开发的时候，首先就会引用 const cloud = require('wx-server-sdk')。云环境会自动安装 package.json 中所指定的依赖库，问题是当前项目本地调试环境还没有这个模块。所以，需要先安装这个模块。解决方法：在本地安装 wx-server-sdk 依赖，如图 5-12 所示，在 cloudfunctions 文件上右击，在弹出的快捷菜单中选择“在终端中打开”选项，输入命令 npm install-save wx-server-sdk@latest，安装好依赖库以后进入云函数本地调试窗口，勾选“开启本地调试”复选框。如果用户要使用本地调试，在安装依赖库时建议使用方法一通过 npm install 安装依赖库，这样本地目录中会安装依赖库。

图 5-11　开启本地调试错误

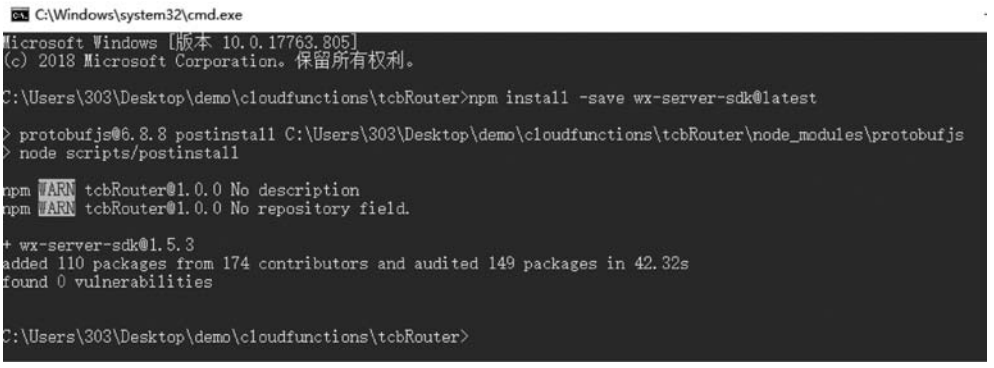

图 5-12　安装 wx-server-sdk 依赖

对于已开启本地调试的云函数，微信开发者工具模拟器中对该云函数的请求以及其他开启了本地调试的云函数对该云函数的请求，都会自动请求到该本地云函数实例。为方便调试，一个云函数在本地仅会有一个实例，实例会串行处理请求，本地云函数递归调用自身将被拒绝。本地调试云函数实例右侧的面板中可以开启“文件变更时自动重新加载”，开启后，每当函数代码发生修改，就会自动重新加载云函数实例，这就省去了关闭本地调试再重新打开本地调试开关的麻烦。

本地调试使用须知：

- npm 依赖：若云函数中使用到 npm 模块，需在云函数本地目录安装相应依赖才可正常使用云函数本地调试功能。
- native 依赖：如果云函数中使用了 native 依赖（注意 native 依赖是需要各个平台分别编译的），比如在 Windows 上本地调试时安装的 native 依赖是 Windows 上编译的结果，而线上云函数环境是 Linux 环境，因此调试完毕上传云函数注意选择云端安装依赖的上传方式，该方式会自动在云端环境下编译 native 依赖，如果由于云端编译环境不足而需要选择全量上传则需要在 Linux CentOS 7 下编译后上传结果。
- Node.js 版本：系统默认使用开发者工具自带的 Node.js，用户可通过点击本地调试面板左上方的设置进行修改。
- 云函数实例个数：本地调试下一个云函数最多只会有一个实例，对本地云函数实例的并发请求会被实例串行处理。

5.4　定时触发器

如果云函数需要定时/定期执行，也就是定时触发，可以使用云函数定时触发器。配置了定时触发器的云函数，会在相应时间点被自动触发，函数的返回结果不会返回给调用方。

在需要添加触发器的云函数目录下新建文件 config.json，格式如下：

```
{
  // triggers 字段是触发器数组,目前仅支持一个触发器,即数组只能填写一个,不可添加多个
  "triggers": [
    {
      // name: 触发器的名字,规则见下方说明
      "name": "myTrigger",
      // type: 触发器类型,目前仅支持 timer(即定时触发器)
      "type": "timer",
      // config: 触发器配置,在定时触发器下,config 格式为 Cron 表达式,规则见下方说明
      "config": "0 0 2 1 * * *"
    }
  ]
}
```

字段规则如下：

- 定时触发器名称(name)：最大支持 60 个字符，支持 a～z、A～Z、0～9、-和_。必须以字母开头，且一个函数下不支持同名的多个定时触发器。
- 定时触发器触发周期(config)：指定的函数触发时间。填写自定义标准的 Cron 表达式来决定何时触发函数。

1. Cron 表达式

Cron 表达式有 7 个必需字段，按空格分隔，如表 5-1 所示。

表 5-1 Cron 表达式的必需字段

第 1 个	第 2 个	第 3 个	第 4 个	第 5 个	第 6 个	第 7 个
秒	分钟	小时	日	月	星期	年

其中，每个字段都有相应的取值范围，如表 5-2 所示。

表 5-2 必需字段相应的取值范围

字 段	值	通配符
秒	0～59 的整数	, - * /
分钟	0～59 的整数	, - * /
小时	0～23 的整数	, - * /
日	1～31 的整数(需要考虑月的天数)	, - * /
月	1～12 的整数或 JAN、FEB、MAR、APR、MAY、JUN、JUL、AUG、SEP、OCT、NOV、DEC	, - * /
星期	0～6 的整数或 MON、TUE、WED、THU、FRI、SAT、SUN。其中，0 指星期一，1 指星期二，以此类推	, - * /
年	1970～2099 的整数	, — * /

2. 通配符(见表 5-3)

表 5-3 通配符

通配符	含 义
,(逗号)	代表用逗号隔开的字符的并集。例如，在“小时”字段中 1,2,3 表示 1 点、2 点和 3 点

续表

通配符	含　义
-（破折号）	包含指定范围的所有值。例如，在“日”字段中，1～15 包含指定月份的 1～15 号
*（星号）	表示所有值。在“小时”字段中，* 表示每小时
/（正斜杠）	指定增量。在“分钟”字段中，输入 1/10 以指定从第一分钟开始的每隔 10 分钟重复。例如，第 11 分钟、第 21 分钟和第 31 分钟，以此类推

3. 注意事项

在 Cron 表达式中的“日”和“星期”字段同时指定值时，两者为“或”关系，即两者的条件分别均生效。

4. 示例

下面展示了一些 Cron 表达式和相关含义的示例：

- * /5 * * * * * * 表示每 5 秒触发一次；
- 0 0 2 1 * * * 表示在每月的 1 日的凌晨 2 点触发；
- 0 15 10 * * MON-FRI * 表示在周一到周五每天上午 10:15 触发；
- 0 0 10,14,16 * * * * 表示在每天上午 10 点、下午 2 点和 4 点触发；
- 0 * /30 9-17 * * * * 表示在每天上午 9 点到下午 5 点内每半小时触发；
- 0 0 12 * * WED * 表示在每个星期三中午 12 点触发。

5.5　云函数高级用法——TcbRouter

微信小程序云开发的云函数都是运行在不同的开发环境中，每个云函数都是一个功能模块，传统的云函数用法是一个云函数处理一个任务，如图 5-13 所示。但是一个用户在一个环境中只有 50 个云函数，经常会出现 50 个云函数不够用的情况，而且为了方便维护管理和公用代码块复用，需要将具有相似的处理逻辑云函数合并成一个云函数，这时就需要用到路由控制。一种常见的方法是在请求云函数时多加一个参数，在云函数中根据该参数利用 switch…case 语句（或者 if…else 语句）来区别对该云函数的不同请求，这种方法比较简单粗暴，例如：

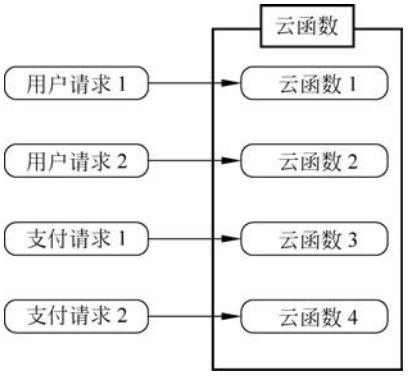

图 5-13　一个云函数处理一个任务

```
exports.main = async(event, context) =>{
    let action = event.action
    switch (action) {
      case 'actionA':
        {
          //执行 actionA 任务
        }
      case 'actionB':
        {
          //执行 actionB 任务
        }
      case 'actionC':
        {
          //执行 actionC 任务
        }
      default:
        {
          //执行 default 任务
        }
    }
  }
```

在上面的例子中，请求云函数时多加一个参数 action，就可以在云函数中接受参数 action，然后使用 switch…case 语句来执行不同的任务。

switch…case 的处理方式往往可读性差，不利于管理。为了解决这个问题，腾讯云 Tencent Cloud Base 团队开发了 TcbRouter，TcbRouter 是一个基于 Koa 风格的云函数路由库，通过 TcbRouter 路由管理云函数可以优化云函数处理逻辑，如图 5-14 所示。云函数中有一个分派任务的路由管理，将不同的任务分配给不同的本地函数处理。

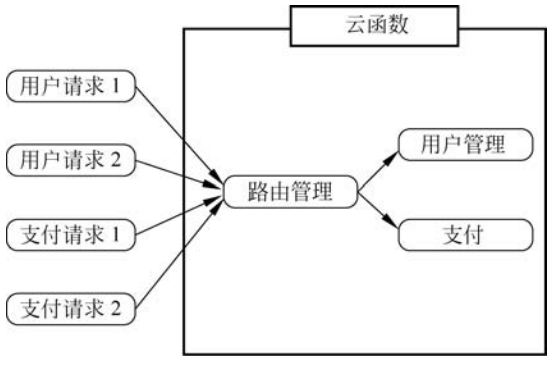

图 5-14　通过路由管理云函数

TcbRouter 安装命令如下：

```
npm install -- save tcb - router
```

TcbRouter 框架如下：

```
const TcbRouter = require('tcb-router');
//云函数入口函数
exports.main = async (event, context) =>{
  const app = new TcbRouter({ event });
  //----------------------------------------
  //使用app.use()或者app.router()处理路由
  //----------------------------------------
  return app.serve();
}
```

app.use()可以在所有的路由上进行处理：

```
//app.use()表示该中间件,适用于所有的路由
app.use(async (ctx, next) =>{
    ctx.data = {};
    await next();     //执行下一中间件
});
```

app.use()方法是支持异步的,所以为了保证正常的按照洋葱模型的执行顺序执行代码,需要在调用next()时让代码等待,等待异步结束后再继续向下执行,所以在使用TcbRouter时建议使用async/await。

app.router()可以处理某个特定的路由,也可以处理路由为数组的情况。路由为数组的情况如下：

```
//路由为数组表示,该中间件适用于 user 和 timer 两个路由
app.router(['user', 'timer'], async (ctx, next) =>{
    ctx.data.company = 'Tencent';
    await next();       //执行下一中间件
});
```

路由为字符串,适用于处理某个特定的路由：

```
app.router('user', async (ctx, next) =>{
    ctx.data.name = 'heyli';
    await next(); //执行下一中间件
}, async (ctx, next) =>{
    ctx.data.sex = 'male';
    await next(); //执行下一中间件
}, async (ctx) =>{
    ctx.data.city = 'Foshan';
    // ctx.body返回数据到微信小程序端
    ctx.body = { code: 0, data: ctx.data};
});
```

微信小程序端调用：

```
//调用名为 router 的云函数,路由名为 user
wx.cloud.callFunction({
    //要调用的云函数名称
    name: "router",
    //传递给云函数的参数
    data: {
        $url: "user", //要调用的路由的路径,传入准确路径或者通配符 *
        other: "xxx"
    }
});
```

TcbRouter 演示案例：

在主页(pages/home/home)中,home.wxml 页面里放两个按钮,分别添加两个 bindtap 事件,为 school 和 user：

```
<cu-custom bgColor="bg-gradual-pink" isBack="{{false}}">
  <view slot="backText">返回</view>
  <view slot="content">TcbRouter 演示</view>
</cu-custom>

<button type="primary" bindtap="school">school</button>
<button type="primary" bindtap="user">user</button>
```

TcbRouter 演示页面如图 5-15 所示。相应的 home.js 代码如下：

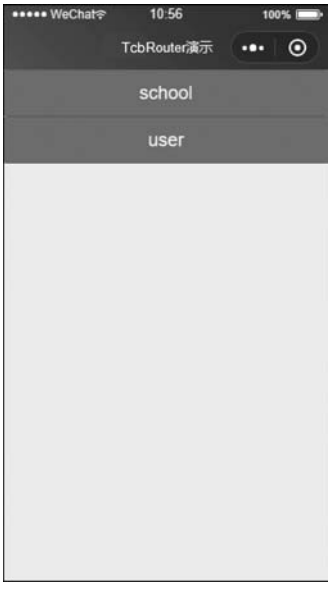

图 5-15　TcbRouter 演示页面

```
Page({
  school: function (event) {
    //调用名为 tcbRouter 的云函数,路由名为 school
```

```
4       wx.cloud.callFunction({
5         //要调用的云函数名称
6         name: "tcbRouter",
7         //传递给云函数的参数
8         data: {
9           $url: "school",          //要调用的路由的路径,传入准确路径或者通配符 *
10          other: "xxx"
11        }
12      }).then(res =>{
13        console.log(res)
14      });
15    },
16    user: function (event) {
17                                   //调用名为 tcbRouter 的云函数,路由名为 user
18      wx.cloud.callFunction({
19                                   //要调用的云函数名称
20        name: "tcbRouter",
21                                   //传递给云函数的参数
22        data: {
23          $url: "user",            //要调用的路由的路径,传入准确路径或者通配符 *
24          other: "xxx"
25        }
26      }).then(res =>{
27        console.log(res)
28      });
29    }
30  })
```

代码第 2～15 行的 school 事件中,调用 tcbRouter 云函数,路由路径为 school; 代码第 16～29 行的 user 事件中,调用 tcbRouter 云函数,路由路径为 user。

在 cloudfunctions 上右击,在弹出的快捷菜单中选择“新建 Node.js 云函数”选项并命名为 tcbRouter,然后右击,在弹出的快捷菜单中选择“在终端中打开”选项,如图 5-16 所示。在终端输入 npm install --save tcb-router,终端安装 tcbRouter 完成后的效果如图 5-17 所示。

图 5-16　新建 tcbRouter 云函数

图 5-17 终端安装 tcbRouter 完成后的效果

安装完 tcbRouter 后，在云函数 tcbRouter/package.json 中可以看到该云函数的依赖包，如图 5-18 所示，如果 tcbRouter 安装成功，在这里就会显示安装的版本。

图 5-18 云函数依赖包

安装完 tcbRouter 后，就可以在 tcbRouter/index.js 中写云函数了，云函数的代码如下：

```
//云函数入口文件
const cloud = require('wx-server-sdk')
const TcbRouter = require('tcb-router')
cloud.init()

//云函数入口函数
exports.main = async (event, context) =>{
  const app = new TcbRouter({ event })

  // app.use()表示该中间件会适用于所有的路由
  app.use(async (ctx, next) =>{
    console.log('---------->进入全局的中间件')
    ctx.data = {};
    ctx.data.openId = event.userInfo.openId
```

```
    await next();                                    //执行下一中间件
    console.log('---------->退出全局的中间件')
  });

  //路由为数组表示,该中间件适用于 user 和 school 两个路由
  app.router(['user', 'school'], async (ctx, next) =>{
    console.log('---------->进入数组路由中间件')
    ctx.data.from = '微信小程序云函数实战'
    await next();                                    //执行下一中间件
    console.log('---------->退出数组路由中间件')
  });

  //路由为字符串,该中间件只适用于 user 路由
  app.router('user', async (ctx, next) =>{
    console.log('---------->进入用户路由中间件')
    ctx.data.name = 'xiaoqiang user';
    ctx.data.role = 'Developer'
    await next();
    console.log('---------->退出用户路由中间件')
  }, async (ctx) =>{
    console.log('---------->进入用户昵称路由中间件')
    ctx.data.nickName = 'BestTony'
    ctx.body = { code: 0, data: ctx.data };   //将数据返回云函数,用 ctx.body
    console.log('---------->退出用户昵称路由中间件')
  });

  app.router('school', async (ctx, next) =>{
    ctx.data.name = '腾讯云学院';
    ctx.data.url = 'cloud.tencent.com'
    await next();
  }, async (ctx) =>{
    ctx.data.nickName = '学院君'
    ctx.body = { code: 0, data: ctx.data };   //将数据返回云函数,用 ctx.body
  });

  return app.serve();
}
```

在云函数 tcbRouter 上右击,在弹出的快捷菜单中选择“上传并部署：上传安装依赖(不上传 node_modules),等待上传云函数结束”选项。

单击图 5-15 中 user 按钮,在微信开发者工具中查看 Console 窗口输出结果,微信小程序端 tcbRouter 输出结果如图 5-19 所示。从 result.data 数据中可以看出,单击 user 按钮,在云函数 tcbRouter 中依次进入了“app.use 中间件”→“app.router(['user', 'school'])路由”→“app.router('user')路由”。

tcbRouter 每个中间件默认接收两个参数：第一个参数是 Context 对象；第二个参数是 next 函数。只要调用 next 函数,就可以把执行权转交给下一个中间件。tcbRouter 的精粹思想

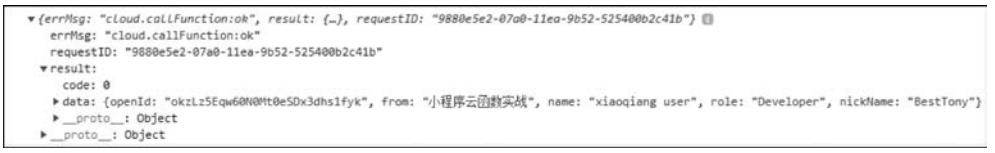

图 5-19　微信小程序端 tcbRouter 输出结果

就是洋葱模型(中间件模型),如图 5-20 所示。多个中间件会形成一个栈结构(middle stack),以“先进后出”(first-in-last-out)的顺序执行。整个过程就像先入栈后出栈的操作。

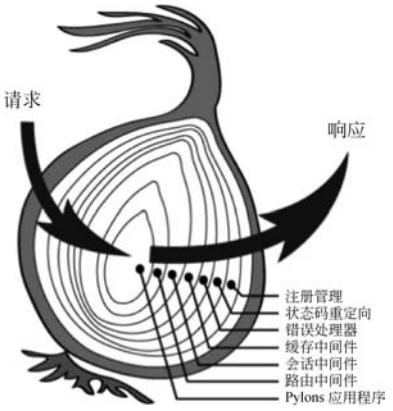

图 5-20　tcbRouter 洋葱模型

为了更好地理解 tcbRouter 洋葱模型,开发者可以进入云开发控制台,选择“云函数”→“日志”→tcbRouter 选项,可以看到云函数 tcbRouter 中间件的执行顺序,如图 5-21 所示。

图 5-21　云函数 tcbRouter 中间件执行顺序

tcbRouter 中间件模型非常好用并且简洁,但是也有自身的缺陷,一旦中间件数组过于庞大,性能会有所下降,因此在实际项目开发中,并不是把所有请求都放在同一个云函数中。