KOA实战-RESTFUL API

课程目标

- 掌握koa中编写Reastful风格API
- 掌握koa中文件上传、表单验证、图形验证码、geetest、跨域等常

见操作

编写Restful API

- Representational State Transfer翻译过来是“表现层状态转化”,它是一种互联网软件的架构原则。因为符合REST风格的Web API设计，就称它为Restful API
- `RESTful` 是目前最流行的 API 规范，适用于 Web 接口规范的设计。让接口易读，且含义清晰。本文将介绍如何设计易于理解和使用的 API，并且借助 Docker api 的实践说明。

URL 设计

1.1 动词 + 宾语

它的核心思想就是客户端发出的数据操作指令都是「动词 + 宾语」的结构，比如 GET /articles这个命令，GET是动词，/articles是宾语。

动词通常来说就是五种 HTTP 方法，对应我们业务接口的 CRUD 操作。而宾语就是我们要操作的资源，可以理解成面向资源设计。我们所关注的数据就是资源。

- GET: 读取资源
- POST：新建资源
- PUT：更新资源
- PATCH：资源部分数据更新
- DELETE：删除资源

正确的例子

- GET /zoos：列出所有动物园
- POST /zoos：新建一个动物园
- GET /zoos/ID：获取某个指定动物园的信息
- PUT /zoos/ID：更新某个指定动物园的信息（提供该动物园的全部信息）
- PATCH /zoos/ID：更新某个指定动物园的信息（提供该动物园的部分信息）

- DELETE /zoos/ID：删除某个动物园
- GET /zoos/ID/animals：列出某个指定动物园的所有动物
- DELETE /zoos/ID/animals/ID：删除某个指定动物园的指定动物

1.2 动词的覆盖

有些客户端只能使用GET和POST这两种方法。服务器必须接受 POST 模拟其他三个方法（PUT、PATCH、DELETE）。

这时，客户端发出的 HTTP 请求，要加上 `X-HTTP-Method-Override` 属性，告诉服务器应该使用哪一个动词，覆盖 POST 方法。

1.3 宾语必须是名词

就是 API 的url ，是 HTTP 动词作用的对象，所以应该是名词。例如 /books 这个 URL 就是正确的，而下面的 URL 不是名词，都是错误的写法。

错误示范：

```
GET /getAllUsers?name=jl
POST /createUser
POST /deleteUSer
```

1.4 复数名词

URL 是名词，那么是使用复数还是单数?

没有统一的规定，但是我们通常操作的数据多数是一个集合，比如 `GET /books` ,所以我们就使用复数。

统一规范，建议都使用复数 URL， 比如 获取 id = 2 的书 `GET /books/2` 要好于 `GET /book/2` 。

1.5 避免出现多级 URL

有时候我们要操作的资源可能是有多个层级，因此很容易写多级 URL，比如获取某个作者某种分类的文章。

> `GET /authors/2/categories/2` 获取作者ID = 2获取分类 = 2 的文章

这种 URL 不利于拓展，语义 也不清晰。

更好的方式就是 除了第一级，其他级别都是通过查询字符串表达。

> 正确方式： `GET /authors/2?categories=2`

查询已发布的文章

> 错误 写法： GET /artichels/published
>
> 正确写法： GET /artichels?published=true

过滤信息（Filtering）

状态码如果记录数量很多，服务器不可能都将它们返回给用户。API 应该提供参数，过滤返回结果。

下面是一些常见的参数。

> - ?limit=10：指定返回记录的数量
> - ?offset=10：指定返回记录的开始位置。
> - ?page=2&per_page=100：指定第几页，以及每页的记录数。
> - ?sortby=name&order=asc：指定返回结果按照哪个属性排

> 序，以及排序顺序。
> - ?animal_type_id=1：指定筛选条件

参数的设计允许存在冗余，即允许API路径和URL参数偶尔有重复。比如，GET /zoo/ID/animals 与 GET /animals?zoo-id=ID 的含义是相同的。推荐后者，避免出现多级URL。

2.1 状态码必须精确

客户端的请求，服务求都必须响应，包含 HTTP 状态码和数据。

HTTP 状态码就是一个三位数，分成五个类别。

- 1xx：相关信息
- 2xx：操作成功
- 3xx：重定向
- 4xx：客户端错误
- 5xx：服务器错误

2.2 2xx 状态码

200状态码表示操作成功，但是不同的方法可以返回更精确的状态码。

- GET: 200 OK
- POST: 201 Created
- PUT: 200 OK
- PATCH: 200 OK
- DELETE: 204 No Content

2.3 4xx 状态码

4xx状态码表示客户端错误，主要有下面几种。

- 400 Bad Request：服务器不理解客户端的请求，未做任何处理。
- 401 Unauthorized：用户未提供身份验证凭据，或者没有通过身份验证。
- 403 Forbidden：用户通过了身份验证，但是不具有访问资源所需的

权限。

- 404 Not Found：所请求的资源不存在，或不可用。
- 405 Method Not Allowed：用户已经通过身份验证，但是所用的 HTTP 方法不在他的权限之内。
- 410 Gone：所请求的资源已从这个地址转移，不再可用。
- 415 Unsupported Media Type：客户端要求的返回格式不支持。比如，API 只能返回 JSON 格式，但是客户端要求返回 XML 格式。
- 422 Unprocessable Entity ：客户端上传的附件无法处理，导致请求失败。
- 429 Too Many Requests：客户端的请求次数超过限额。

2.4 5xx 状态码

5xx状态码表示服务端错误。一般来说，API 不会向用户透露服务器的详细信息，所以只要两个状态码就够了。

- 500 Internal Server Error：客户端请求有效，服务器处理时发生了意外。
- 503 Service Unavailable：服务器无法处理请求，一般用于网站维护状态。

服务器响应

3.1 不要返回纯文本

API 返回的数据格式，不应该是纯文本，而应该是一个 JSON 对象，因为这样才能返回标准的结构化数据。所以，服务器回应的 HTTP 头的 Content-Type 属性要设为 application/json 。

客户端请求时，也要明确告诉服务器，可以接受 JSON 格式，即请求的 HTTP 头的ACCEPT 属性也要设成 application/json。下面是一个例子。

3.2 发生错误的时候，不要返回 200 状态码

有一种不恰当的做法是，即使发生错误，也返回200状态码，把错误信息放在数据体里面，就像下面这样。

错误例子：

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "status": "fail",
    "msg": "错误"
}
```

上面代码中，解析数据体以后，才能得知操作失败。

这种做法实际上取消了状态码，这是完全不可取的。正确的做法是，状态码反映发生的错误，具体的错误信息放在数据体里面返回。下面是一个例子。

正确方式：

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
    "status": "fail",
    "msg": "错误"
}
```

- 示例：

```
const users = ({
    id: 1,
    name: 'tom',
}, {
    id: 2,
    name: 'jack'
})
```

```
// 用户信息管理api实现
router.get('/', function (ctx, next) {
    console.log('GET /users');
    const { name } = ctx.query; //?name=tom
    let data = users;
    if (name) {
        data = users.filter(u => u.name === name);
    }
    ctx.body = { ok: 1, data };
})
router.get('/:id', function (ctx, next) {
    console.log('GET /users/:id');
    const { id } = ctx.params; //users/1
    console.log(id);

    const data = users.find(u => u.id == id);
    ctx.body = { ok: 1, data };
})
router.post('/', function (ctx, next) {
    console.log('POST /user');
    const { body: user } = ctx.request;
    user.id = users.length + 1;
    users.push(user);
    console.log(users);

    ctx.body = { ok: 1 }

})
router.put('/', function (ctx, next) {
    console.log('PUT /users');
    const { body: user } = ctx.request; //请求body
    // 修改数据
    const idx = users.findIndex(u => u.id == user.id)
    if (idx > -1) {
        users(idx) = user;
    }
    console.log(users);
```

```
45
46      ctx.body = { ok: 1 }
47  })
48  router.delete('/:id', function (ctx, next) {
49      const { id } = ctx.params;
50      // 删除数据
51      const idx = users.findIndex(u=>u.id==id);
52      if(idx > -1){
53          users.splice(idx,1);
54      }
55      console.log(users);
56      ctx.body = {ok:1};
57  })
```

解决跨域

- 下载模块： `npm i  koa2-cors`

```
1  const koa = require('koa');
2  const cors = require('koa2-cors');
3  const app = new koa();
4  app.use(cors());
```

文件上传

- 安装koa-multer: `npm i koa-multer -S`
- 配置： ./routes/users.js

```
const multer = require('koa-multer');
const upload = multer({ dest: './public/images' });
router.post('/upload', upload.single('avatar'), (ctx, next) => {
  // ctx.req.file  是avatar文件的信息
  // ctx.req.body 文本域数据 如果存在
  //  ctx.body = '上传成功';
  ctx.body = {
    ok: 1
  }
})
```

要想修改文件名

```
const multer = require('koa-multer');
//配置
const storage = multer.diskStorage({
  //文件保存路径
  destination: function (req, file, cb) {
    cb(null, 'public/images/')
  },
  //修改文件名称
  filename: function (req, file, cb) {
    var fileFormat = (file.originalname).split(".");  //以点分割成数组，数组的最后一项就是后缀名
    cb(null, Date.now() + "." + fileFormat(fileFormat.length - 1));
  }
})
//加载配置
const upload = multer({ storage: storage });
router.post('/upload', upload.single('avatar'), (ctx, next) => {
  // ctx.req.file  是avatar文件的信息
  // ctx.req.body 文本域数据 如果存在
  //  ctx.body = '上传成功';
  console.log(ctx.req.file.filename);
```

```
21
22    ctx.body = {
23      ok: 1
24    }
25  })
```

表单验证

- 安装koa-bouncer: `npm i koa-bouncer -S`
- 配置：app.js

```
1  //为koa上下文扩展一些校验的方法
2  const bouncer = require('koa-bouncer');
3  app.user(bouncer.middleware());
```

- 应用：users.js

```
1  const bouncer = require('koa-bouncer');
2  router.post('/', function (ctx, next) {
3    try {
4      ctx.validateBody('uname')
5        .required('用户名是必须的')
6        .isString()
7        .trim()
8
9      ctx.validateBody('email')
10       .optional()
11       .isString()
12       .trim()
13       .isEmail('非法的邮箱格式')
14
15     ctx.validateBody('pwd1')
16       .required('密码是必填项')
17       .isString()
18       .isLength(6, 16, '密码必须是6~16位字符')
19
```

```
    ctx.validateBody('pwd2')
      .required('密码确认为必填项')
      .isString()
      .eq(ctx.vals.pwd1, '两次密码不一致')

    // 校验数据库是否存在相同值
    // ctx.validateBody('uname')
    // .check(await db.findUserByUname(ctx.vals.uname),'Username taken')
    ctx.validateBody('uname').check('tom','用户名已存在')

    //如果代码执行到这里，校验通过
    // 校验器会用净化后的值填充·ctx.vals·对象
    console.log(ctx.vals);
    // 接下来的 对数据库进行操作，插入用户到数据库中
    //....

    console.log('POST /user');
    const { body: user } = ctx.request;
    user.id = users.length + 1;
    users.push(user);
    console.log(users);
    ctx.body = { ok: 1 }

  } catch (error) {
    // 校验异常特别判断
    if(error instanceof bouncer.ValidationError){
      ctx.status = 400
      ctx.body = '校验失败' + error.message;
      return;
    }
    throw error
  }

})
```

图形验证码

- 安装trek-captcha: `npm i trek-captcha --save`
- 使用：./routers.index.js

```
const catpcha = require('trek-captcha')
router.get('/captcha', async (ctx, next) => {
  const { token, buffer } = await catpcha({ size: 4 });
    //token的作用 前端输入完验证码与此时的token做对比
  ctx.body = buffer;
})
```

- 图片显示，index.html

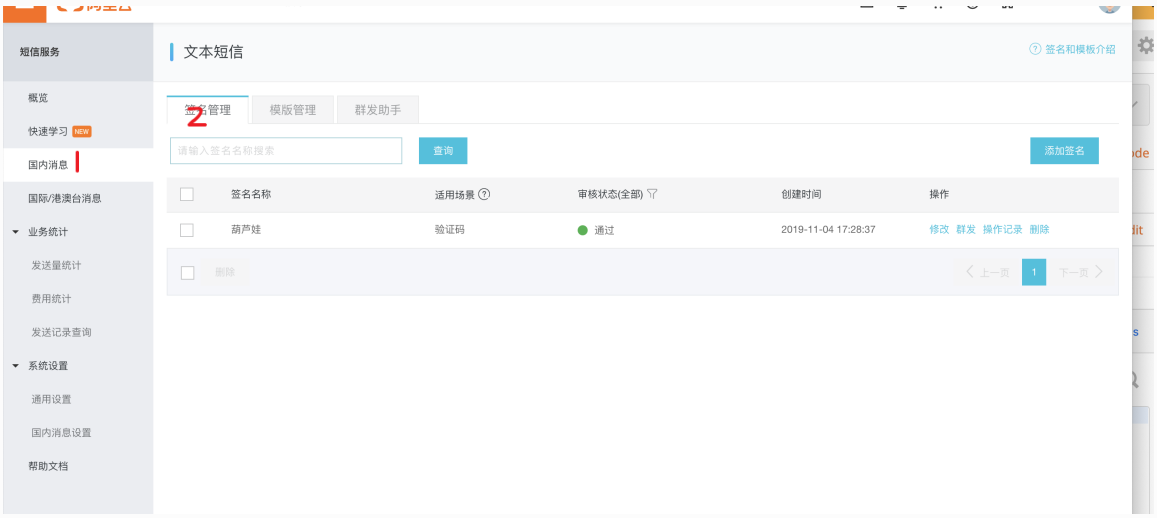
```
<img src="/captcha" id="captcha" ref='captcha'>
<button @click='changeCaptcha'>换一张</button>
<script>
    new Vue({
        ...
        methods: {
        changeCaptcha(){
            this.$refs.captcha.src = '/captcha?r=' +
Date.now();
        },
    })
</script>

```

发送短信

准备工作

注册阿里云短信服务

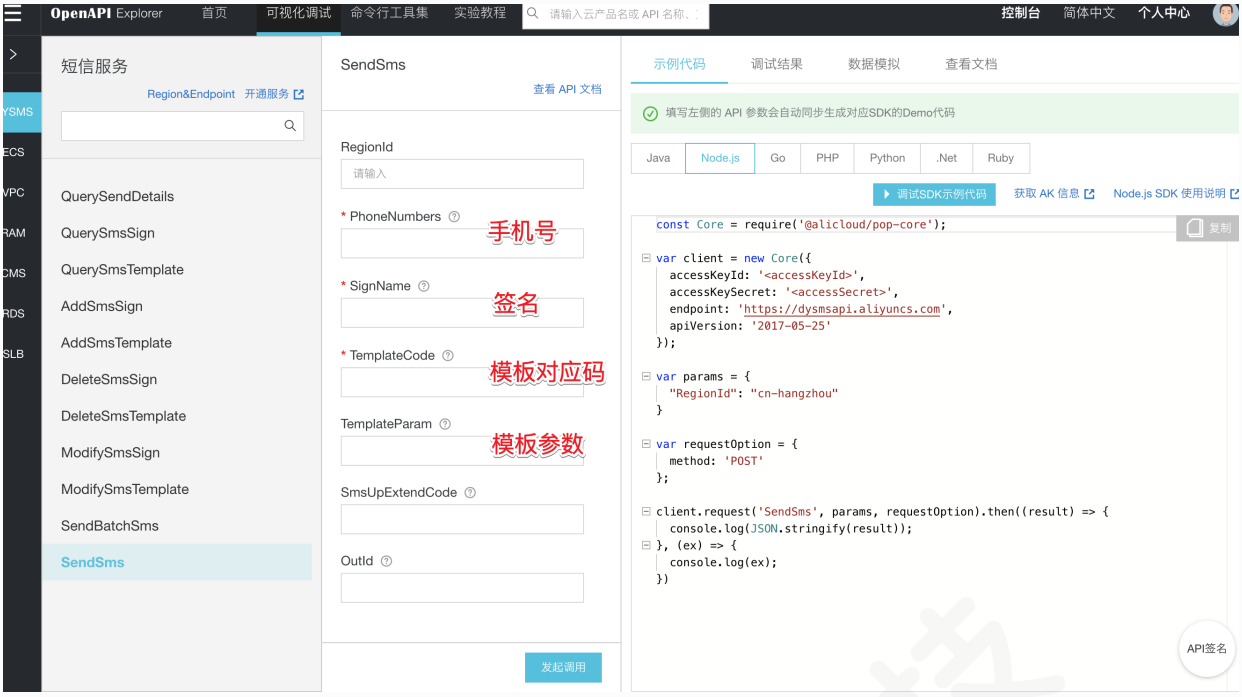
申请签名管理



申请模板管理

注册完成之后查看AccessKey

登录阿里云短信服务进行测试



这四个选项 上面都已获取到

代码测试：

- 下载 `npm i @alicloud/pop-core -S`

routes/index.js

```
/// 阿里云短信服务
const Core = require('@alicloud/pop-core');


const client = new Core({
  accessKeyId: 'LTAI4Fm3obsdQ7gHYcSwMq2s',
  accessKeySecret:
'Ywz15QLdre0U01WoTVSaM0aeBpeGmD',
  endpoint: 'https://dysmsapi.aliyuncs.com',
  apiVersion: '2017-05-25'
});
//随机验证码
function ran(num) {
  let code = "";
  for (let i = 0; i < num; i++) {
```

```
    let radom = Math.floor(Math.random() * 10);
    code += radom;
  }
  return {
    "code": code
  };
}
const moment = require('moment')
router.post('/sms', async (ctx, next) => {
  const toPhone = ctx.request.body.phone;
  const randCode = ran(6);
  const code = randCode.code
  const params = {
    "RegionId": "cn-hangzhou",
    "PhoneNumbers": toPhone,
    "SignName": "葫芦娃",
    "TemplateCode": "SMS_176941377",
    "TemplateParam": JSON.stringify(randCode)
  }
  const requestOption = {
    method: 'POST'
  };

  try {
    const response = await client.request('SendSms',
params, requestOption);
    console.log(response);

    if (response.Code === 'OK') {
      //短信发送成功，存储验证码到session，过期时间1分钟
      const expires = moment().add(1, 'minutes').toDate();
      ctx.session.smsCode = {
        toPhone,
        code,
        expires
      }
      ctx.body = {
```

```
51          "ok": 1
52        }
53      } else {
54        ctx.body = {
55          ok: 0,
56          message: response.Message
57        }
58      }
59
60    } catch (error) {
61      console.log(error);
62
63      ctx.body = {
64        ok: 0,
65        message: error.message
66      }
67    }
68
69
70
71  })
```