一、基本使用

1. 什么是mockjs

Mock.js 是一款前端开发中拦截Ajax请求再生成随机数据响应的工具.可以用来模拟服务器响应. 优点是非常简单方便, 无侵入性, 基本覆盖常用的接口数据类型.

- 前后端分离
- 开发无侵入（不需要修改既有代码，就可以拦截 Ajax 请求，返回模拟的响应数据。）
- 数据类型丰富（支持生成随机的文本、数字、布尔值、日期、邮箱、链接、图片、颜色等。）
- 增加单元测试的真实性（通过随机数据，模拟各种场景。）
- 用法简单、符合直觉的接口。
- 方便扩展（支持支持扩展更多数据类型，支持自定义函数和正则。）

安装

```
npm install mockjs
```

2. 快速入门

在项目中创建mock目录，新建index.js文件

```
//引入mockjs
import Mock from 'mockjs'
//使用mockjs模拟数据
Mock.mock('/product/search', {
    "ret":0,
    "data":
      {
        "mtime": "@datetime",//随机生成日期时间
        "score|1-800": 1,//随机生成1-800的数字
        "rank|1-100":  1,//随机生成1-100的数字
        "stars|1-5": 1,//随机生成1-5的数字
        "nickname": "@cname",//随机生成中文名字
        //生成图片
        "img":"@image('200x100', '#ffcc33', '#FFF', 'png', 'Fast Mock')"
      }
});
```

main.js里面引入

```
import './mock'
```

xxx.vue文件中调用mock.js中模拟的数据接口，这时返回的response就是mock.js中用Mock.mock('url',data）中设置的data了

```
import axios from 'axios'
export default {
  mounted:function(){
    axios.get("/product/search").then(res => {
      console.log(res)
    })
  }
}
```

注意，如果`get`请求如果带参数，会以`?a=b&c=d`，形式拼接到url上，这时mock请把接口url写为正则匹配，否则匹配不到就报错`Mock.mock(RegExp(API_URL + ".*"))`，如：

```
import axios from 'axios'
export default {
  mounted:function(){
    ///product/search?id=2
    axios.get("/product/search",{params:{id:2}}).then(res => {
      console.log(res)
    })
  }
}
```

在url后面加上`.*`

```
mockjs.mock(RegExp("/product/search"+ ".*"),'get',option=>{
    console.log(option.url) // 请求的url
    console.log(option.body)// body为post请求参数
    return {
        status:200,
        message:"获取数据成功"
    }
})
```

3. 核心方法

```
Mock.mock( rurl?, rtype?, template|function( options ) )
```

这是mock的核心方法，根据数据模板生成模拟数据。

- rurl，可选。
  表示需要拦截的 URL，可以是 URL 字符串或 URL 正则。例如：正则+变量写法 -> `RegExp(API_URL.LOGIN + ".*")`、正则写法 -> `/\/domain\/list\.json/`、字符串写法 -> `'/domian/list.json'`。
- rtype，可选。
  表示需要拦截的 Ajax 请求类型。例如`GET`、`POST`、`PUT`、`DELETE`等。
- template，可选。
  表示数据模板，可以是对象或字符串。例如`{ 'data|1-10':[{}] }`、`'@EMAIL'`。
- function(options)，可选。

  表示用于生成响应数据的函数。

- options
  指向本次请求的 Ajax 选项集，含有 `url`、`type` 和 `body` 三个属性，参见 XMLHttpRequest 规范。

具体使用：

- Mock.mock(template)： 根据数据模板生成模拟数据。
- Mock.mock(rurl, template)： 记录数据模板。当拦截到匹配 `rurl` 的 Ajax 请求时，将根据数据模板 `template` 生成模拟数据，并作为响应数据返回。
- Mock.mock(rurl, function(options))： 记录用于生成响应数据的函数。当拦截到匹配 `rurl` 的 Ajax 请求时，函数 `function(options)` 将被执行，并把执行结果作为响应数据返回。
- Mock.mock(rurl, rtype, template)： 记录数据模板。当拦截到匹配 `rurl` 和 `rtype` 的 Ajax 请求时，将根据数据模板 `template` 生成模拟数据，并作为响应数据返回。
- Mock.mock(rurl, rtype, function(options))： 记录用于生成响应数据的函数。当拦截到匹配 `rurl` 和 `rtype` 的 Ajax 请求时，函数 `function(options)` 将被执行，并把执行结果作为响应数据返回。

4. 设置延时请求到数据

不设置延时很有可能遇到坑，这里需要留意，因为真实的请求是需要时间的，mock不设置延时则是马上拿到数据返回，这两个情况不同可能导致在接口联调时出现问题。所以最好要先设置延时请求到数据。

```
//延时400ms请求到数据
Mock.setup({
    timeout: 400
})
//延时200-600毫秒请求到数据
Mock.setup({
    timeout: '200-600'
})
```

二、数据生成规则

1. 数据模板DTD

mock的语法规范包含两层规范:

- 数据模板 （DTD）
- 数据占位符 （DPD）

基本语法

数据模板中的每个属性由 3 部分构成：属性名name、生成规则rule、属性值value：

```
'name|rule': value
```

属性名和生成规则之间用竖线 | 分隔，生成规则是可选的

生成规则 有 7 种格式：

```
'name|min-max': value
'name|count': value
'name|min-max.dmin-dmax': value
'name|min-max.dcount': value
'name|count.dmin-dmax': value
'name|count.dcount': value
'name|+step': value
```

- 生成规则的含义需要依赖属性值的类型才能确定。
- 属性值中可以含有 `@占位符`。
- 属性值还指定了最终值的初始值和类型。

生成规则和示例

1. 属性值是字符串 String

```
//通过重复 string 生成一个字符串，重复次数大于等于 min，小于等于 max。
'name|min-max': string

//通过重复 string 生成一个字符串，重复次数等于 count。
'name|count': string
```

```
var data = Mock.mock({
    'name1|1-3':'a',        //重复生成1到3个a(随机)
    'name2|2':'b'           //生成bb
})
```

2. 属性值是数字 Number

```
//属性值自动加 1，初始值为 number。
'name|+1': number

//生成一个大于等于 min、小于等于 max 的整数，属性值 number 只是用来确定类型。
'name|min-max': number

//生成一个浮点数，整数部分大于等于 min、小于等于 max，小数部分保留 dmin 到 dmax 位。
'name|min-max.dmin-dmax': number
```

```
Mock.mock({
    'number1|1-100.1-10': 1,
    'number2|123.1-10': 1,
    'number3|123.3': 1,
    'number4|123.10': 1.123
})
//结果:
{
    "number1": 12.92,
    "number2": 123.51,
    "number3": 123.777,
    "number4": 123.1231091814
}
```

```
var data = Mock.mock({
    'name1|+1':4,          //生成4，如果循环每次加1
    ‘name2|1-7':2,         //生成一个数字，1到7之间
    'name3|1-4.5-8':1      ////生成一个小数，整数部分1到4，小数部分5到8位，数字1只是为了确定类型
})
```

3. 属性值是布尔型 Boolean

```
//随机生成一个布尔值，值为 true 的概率是 1/2，值为 false 的概率同样是 1/2。
'name|1': boolean
//随机生成一个布尔值，值为 value 的概率是 min / (min + max)，值为 !value 的概率是 max / (min + max)。
'name|min-max': value
```

```
var data = Mock.mock({
    'name|1':true,           //生成一个布尔值，各一半
    'name1|1-3':true         //1/4是true，3/4是false
})
```

4. 属性值是对象 Object

```
//从属性值 object 中随机选取 count 个属性。
'name|count': object
//从属性值 object 中随机选取 min 到 max 个属性。
'name|min-max': object
```

```
var obj = {
    a:1,
    b:2,
    c:3,
    d:4
}
var data = Mock.mock({
    'name|1-3':obj,       //随机从obj中寻找1到3个属性，新对象
    'name|2':obj          //随机从onj中找到两个属性，新对象
})
```

5. 属性值是数组 Array

```
//从属性值 array 中随机选取 1 个元素，作为最终值。
'name|1': array
//从属性值 array 中顺序选取 1 个元素，作为最终值。
'name|+1': array
//通过重复属性值 array 生成一个新数组，重复次数大于等于 min，小于等于 max。
'name|min-max': array
//通过重复属性值 array 生成一个新数组，重复次数为 count。
'name|count': array
```

```
Mock.mock({
    //通过重复属性值 array 生成一个新数组，重复次数为 1-3次。
    "favorite_games|1-3": [3,5,4,6,23,28,42,45],
});
```

```
var arr = [1,2,3];
var data = Mock.mock({
    'name1|1':arr,          //从数组里随机取出1个值
    'name2|2':arr,          //数组重复count次，这里count为2
    'name3|1-3':arr,        //数组重复1到3次
})
```

6. 属性值是函数 Function

```
执行函数 function，取其返回值作为最终的属性值，函数的上下文为属性 'name' 所在的对象。
'name': function
```

```
var fun = function(x){
    return x+10;
}
var data = Mock.mock({
    'name':fun(10)          //返回函数的返回值20
})
```

7. 属性值是正则表达式 RegExp

```
根据正则表达式 regexp 反向生成可以匹配它的字符串。用于生成自定义格式的字符串。
'name': regexp
```

```
Mock.mock({
    'regexp1': /[a-z][A-Z][0-9]/,
    'regexp2': /\w\W\s\S\d\D/,
    'regexp3': /\d{5,10}/
})
// =>
{
    "regexp1": "pJ7",
    "regexp2": "F)\fp1G",
    "regexp3": "561659409"
}
```

2. 数据占位符DPD

占位符只是在属性值字符串中占个位置，并不出现在最终的属性值中。

占位符的格式为：

```
@占位符
@占位符(参数 [, 参数])
```

关于占位符需要知道以下几点

- 用 `@` 标识符标识后面的字符串是占位符
- 占位符 引用的是 `Mock.Random` 中的方法。
- 可以通过 `Mock.Random.extend()` 来扩展自定义占位符。
- 占位符 也可以引用 数据模板 中的属性。
- 占位符 会优先引用 数据模板 中的属性。
- 占位符 支持 相对路径 和 绝对路径。

```
//引入mockjs
import Mock from 'mockjs'
//使用mockjs模拟数据
Mock.mock('/api/msdk/proxy/query_common_credit', {
    "ret":0,
    "data":
      {
        "mtime": "@datetime",//随机生成日期时间
        "score":  "@natural(1, 800)",//随机生成1-800的数字
        "rank":  "@natural(1, 100)",//随机生成1-100的数字
        "stars": "@natural(0, 5)",//随机生成1-5的数字
        "nickname": "@cname",//随机生成中文名字
      }
});
```

3. 用例

基础随机内容的生成

```
{
  "string|1-10": "=", // 随机生成1到10个等号
  "string2|3": "=", // 随机生成2个或者三个等号
  "number|+1": 0, // 从0开始自增
  "number2|1-00.1-3": 1, // 生成一个小数，小数点前面1到10，小数点后1到3位
  "boolean": "@boolean( 1, 2, true )", // 生成boolean值 三个参数，1表示第三个参数true出现的概率，2表示false出现的概率
  "name": "@cname", // 随机生成中文姓名
  "firstname": "@cfirst", // 随机生成中文姓
  "int": "@integer(1, 10)", // 随机生成1-10的整数
  "float": "@float(1,2,3,4)", // 随机生成浮点数，四个参数分别为，整数部分的最大最小值和小数部分的最大最小值
  "range": "@range(1,100,10)", // 随机生成整数数组，三个参数为，最大最小值和加的步长
  "natural": "@natural(60, 100)", // 随机生成自然数（大于零的数）
  "email": "@email", // 邮箱
  "ip": "@ip" ,// ip
  "datatime": "@date('yy-MM-dd hh:mm:ss')" // 随机生成指定格式的时间
  // ......
}
```

列表数据

```
{
  "code": "0000",
  "data": {
    "pageNo": "@integer(1, 100)",
    "totalRecord": "@integer(100, 1000)",
```

```
    "pageSize": 10,
    "list|10": [{
      "id|+1": 1,
      "name": "@cword(10)",
      "title": "@cword(20)",
      "descript": "@csentence(20,50)",
      "price": "@float(10,100,10,100)"
    }]
  },
  "desc": "成功"
}
```

图片

mockjs可以生成任意大小，任意颜色块，且用文字填充内容的图片，使我们不用到处找图片资源就能轻松实现图片的模拟展示

```
{
"code": "0000",
  "data": {
    "pageNo": "@integer(1, 100)",
    "totalRecord": "@integer(100, 1000)",
    "pageSize": 10,
    "list|10": [{
      // 参数从左到右依次为，图片尺寸，背景色，前景色（及文字颜色）,图片格式，图片中间的填充文字内容
      "image": "@image('200x100', '#ffcc33', '#FFF', 'png', 'Fast Mock')"
    }]
  },
  "desc": "成功"
}
```

4. Mock.Random

Mock.Random 是一个工具类，用于生成各种随机数据。

Mock.Random 的方法在数据模板中称为『占位符』，书写格式为 @占位符(参数 [, 参数]) 。
用法示例：

```
var Random = Mock.Random
Random.email()
// => "n.clark@miller.io"
Mock.mock('@email')
// => "y.lee@lewis.org"
Mock.mock( { email: '@email' } )
// => { email: "v.lewis@hall.gov" }
```

Mock.Random 提供的完整方法（占位符）如下：

Type	Method
Basic	boolean, natural, integer, float, character, string, range, date, time, datetime, now
Image	image, dataImage
Color	color
Text	paragraph, sentence, word, title, cparagraph, csentence, cword, ctitle
Name	first, last, name, cfirst, clast, cname
Web	url, domain, email, ip, tld
Address	area, region
Helper	capitalize, upper, lower, pick, shuffle
Miscellaneous	guid, id

Basic

- Random.boolean(min?max?current?)

随机生成布尔值

```
var bool1 = Random.boolean();       //true false各一半
var bool2 = Random.boolean(1,2, false)    //1/3的可能性是false 2/3是true
```

- Random.natural(min?,max?)

随机生成一个自然数，什么叫自然数，就是大于等于0的

```
var natural1 = Random.natural();       //默认值最大为 9007199254740992
var natural2 = Random.natural(4);          //随机出来的最小值是4
var natural3 = Random.natural(6,9);
```

- Random.Integer(min?,max?)

生成一个随机的整数，可以是负数。

```
var integer1 = Random.integer();
var integer2 = Random.integer(-10);         //随机最小值是-10
var integer3 = Random.integer(-10,20);
```

- Random.float(min?,max?,dmin?,dmax?)

随机生成一个小数浮点数,四个参数分别为，整数部分最小值最大值，小数部分最小值最大值。

```
var float1 = Random.float();
var float2 = Random.float(3,8);
var float3 = Random.float(1,3,5,7)
```

- Random.character(pool?)

随机生成一个字符,pool的值可以是:

1. upper: 26个大写字母
2. lower: 26个小写字母
3. number: 0到9十个数字
4. sympol: "!@#$%^&*()[]"

```
var character1 = Random.character();
var character2 = Random.character('lower');
var character3 = Random.character('upper');
var character4 = Random.character('symbol');
```

- Random.string(pool?,min?,max?)

随机生成一个字符串，pool的值同上边四个。

```
var str1 = Random.string();                 //长度3到7位
var str2 = Random.string(5);                //长度5位
var str3 = Random.string('lower',7);        //长度7位，小写
var str4 = Random.string(4,6);              //长度4到
var str5 = Random.string('新的字符串会从这里选择4到5位',4,6);    //从第一个参数里选择4到5
位
```

- Random.range(start?,stop,step?)

返回一个整型数组

1. start,可选，数组起始值，闭区间
2. stop,必选，数据结束值，开区间
3. step,可选，数据每一项间隔值

```
var range1 = Random.range(10);      //[0,1,2,3,4,5,6,7,8,9]
var range2 = Random.range(14,20);   //[14,15,16,17,18,19]
var range3 = Random.range(3,13,2); //[3,5,7,9,11]
```

- Random.date(format?)

返回一个随机日期的字符串
format的格式是'yyyy-MM-dd',可以随机组合

```
var date1 = Random.date();
var date2 = Random.date('yyyy-MM-dd');
var date3 = Random.date('y-M-d');
var date4 = Random.date('yy-MM-dd');
```

- Random.time(format?)

返回时间字符串
format的格式是'HH-mm-ss'

```
var time1 = Random.time();
var time2 = Random.time('HH-mm-ss');
var time3 = Random.time('J-m-s');
```

- Random.datetime(format?)

上边两个的结合版

```
var dt1 = Random.datetime();
var dt2 = Random.datetime('yyyy-MM-dd HH-mm-ss');
Random.now(unit?,format?)
```

返回当前时间的字符串

Image

> 一般情况下，使用dataImage更好,因为更简单，但是如果要生成高度自定义的图片，则最好用image。另外，dataImage生成的是base64编码

- Random.image(size?,background?,foreground?,format?text?)

1. size 图片宽高，格式是'宽x高'
2. background:图片的背景色，默认值#000000
3. foreground：图片的文字前景色，默认#FFFFFF
4. format：图片的格式，默认'.png'
5. text:图片上的默认文字，默认值为参数size

其中size的取值范围是

```
[
'300x250', '250x250', '240x400', '336x280',
'180x150', '720x300', '468x60', '234x60',
'88x31', '120x90', '120x60', '120x240',
'125x125', '728x90', '160x600', '120x600',
'300x600'
]
```

图片的格式可以选择`.png`、`.gif`、`.jpg`

```
var image1 = Random.image();
var image2 = Random.image('128x90');
var image3 = Random.image('120x660','#ccc');    //前景色#ccc
var image4 = Random.image('226x280','#eee','第三个参数是文字不是前景色');
var image5 = Random.image('66x31','#ddd','#123456','四个参数的时候第三个参数是前景色');
var image6 = Random.image('240x400','#333','#1483dc','.gif','全部参数的情况下');
```

- Random.dataImage(size?,text?)

返回一段base64编码，两个参数同上。

```
var di1 = Random.dataImage();
var di2 = Random.datImage('300x600');
var di3 = Random.dataImage('180x150','hahahaha');
```

Color

- Random.color()

有好几个相关的方法

```
var color = Random.color(); //格式'#rrggbb'
var hex = Random.hex();   //好像和color没什么不同
var rgb = Random.rgb();   //生成格式如rgb(133,233,244)
var rgba = Random.rgba(); //生成个事如rgba(111,222,233,0.5)
var hsl = Random.hsl();   //生成格式(345,82,71)
```

Text

- Random.paragraph(in?,max?,len?)

随机生成一段文本，

```
var para1 = Random.paragraph();    //随机生成一短文本，范围3到7
var para2 = Random.paragraph(10);  //随机生成长度是10的文本
var para3 = Random.paragraph(9,12); //随机生成9到11位长度的文本
```

- Random.sentence(min?,max?,len?)

随机生成一个句子，第一个单词的首字母大写

```
var sen1 = Random.sentence();        //默认长度12到18
var sen2 = Random.sentence(10);      //随机生成一个单词个数为10的句子
var sen3 = Random.sentence(5,10);    //随机生成一个5到9单词个数的句子
```

- Random.word(min?,max?,len?)

随机生成一个单词

```
var word1 = Random.word();           //默认长度3到10
var word2 = Random.word(7);          //随机生成长度是7的单词
var word3 = Random.word(2,12);       //随机生成2到11位长度的单词
```

- Random.title(min?,max?,len?)

随机生成一段标题，每个单词的首字母大写

```
var title1 = Random.title();         //title中的单词个数
var title2 = Random.title(6);        //title六个单词
var title3 = Random.title(7,12);     //title7到11个单词
```

另外还有四个方法，四个方法前边加一个`c`,返回中文内容

1. `Random.cparagraph`, 返回中文文本
2. `Random.csentence`, 返回中文句子
3. `Random.cword`, 返回中文文字
4. `Random.ctitle`, 返回中文标题

Name

```
var first = Random.first()          随机生成常见英文名
var last = Random.last()            随机生成常见英文姓
var name = Random.name()            随机生成常见英文姓名
var cfirst = Random.cfirst()        随机生成常见中文姓
var clast = Random.clast()          随机生成常见中文名
var cname = Random.cname()          随机生成一个常见中文姓名
```

Web

- Random.url(protocol?,host?)

随机生成一个url

1. `protocol`可选参数，表示网络协议，如`http`。
2. `host`表示域名和端口号

```
var url1 = Random.url();
var url2 = Random.url('http');
var url3 = Random.url('http','58.com');
```

- Random.protocol()

随机生成一个域名

```
var protocol = Random.protocol()
```

> protocol可以选的
> 值，'http'、'ftp'、'gopher'、'mailto'、'mid'、'cid'、'news'、'nntp'、'prospero'、'telnet'、'rlogin'、'tn3270'、'wais'。

- Random.domin()

随机生成一个域名

- Random.tld()

随机生成一个顶级域名

```
var domain = Random.domain()
var tld = Random.tld()
```

- Random.email(domain?)

随机生成一个email地址，domain表示域名

```
var email1 = Random.email();
var email2 = Random.email('58.com')     //生成xxxx@58.com
```

- Random.ip()

随机生成一个ip地址

```
var ip = Random.ip()
```

Address

- Random.region()

随机生成一个中国的大区，如华北，西南

```
var region = Random.region();
```

- Random.province()

随机生成一个中国省直辖市自治区特别行政区

```
var province = Random.province()
```

- Random.city(prefix?)

随机生成一个中国城市，prefix布尔值，表示是否标注所在省

```
var city1 = Random.city();
var city2 = Random.city(ture);
Random.country(prefix?)
```

随机生成一个中国县，`prefix`布尔值，表示是否显示所属的省市

```
var county1 = Random.county();
var county2 = Random.county(ture);
```

- Random.zip()

随机生成一个六位数邮政编码

```
var zip = Random.zip();
```

Helper

- Random.capitlize(word)

把第一个字母转成大写

```
var capitalize = Random.capitalize('hello')
```

- Random.upper(str)

转成大写

```
var upper = Random.upper('zhang');
```

- Random.lower(str)

转成小写

```
var lower = Random.lower('JINGWEI');
```

- Random.pick(arr)

从数组中随机选取一个元素

```
var arr = [1,4,5,6,7,8];
var pick = Random.pick(arr);
```

- Random.shuffle(arr);

打乱数组的顺序并返回

```
var arr  = [1,2,3,4,6];
var shuffle = Random.shuffle(arr);
```

Miscellaneous

- Random.guid()

随机生成一个GUID

- Random.id()

随机生成一个18位身份证id

```
var guid = Random.guid();
var id = Random.id();
```

扩展

Mock.Random 中的方法与数据模板的 `@占位符` 一一对应，在需要时还可以为 Mock.Random 扩展方法，然后在数据模板中通过 `@扩展方法` 引用。例如：

```
Random.extend({
    constellation: function(date) {
        var constellations = ['白羊座'`, '金牛座', '双子座', '巨蟹座', '狮子座', '处女座', '天秤座', '天蝎座', '射手座', '摩羯座', '水瓶座', '双鱼座']
        return this.pick(constellations)
    }
})
Random.constellation()
// => "水瓶座"
Mock.mock('@CONSTELLATION')
// => "天蝎座"
Mock.mock({
    constellation: '@CONSTELLATION'
})
// => { constellation: "射手座" }
```