正则表达式

20180803 宁勃

正则表达式
Regular_Expressions

正则表达式描述了一种字符串匹配的模式。

可以用来检查一个串是否含有某种子串、将匹配的子串替换或者从某个串中取出符合某个条件的子串等。

正则表达式 格式

/pattern/flags
/abc/
/abc/g
let reg = /abc/;

正则表达式字面量

new RegExp(pattern,flags)
let reg = new RegExp('abc','g');

RegExp对象的构造函数

正则表达式 方法

用于 RegExp 的 exec 和 test 方法，以及 String 的 match 、replace 、search 和 split 方法

- exec 在字符串中查找匹配，返回一个数组，未匹配到则返回null
- test 测试是否匹配，返回 true 或者 false

- match 在字符串中查找匹配，返回一个数组，未匹配到则返回null
- search 测试匹配的方法，返回匹配到的位置索引，或者在失败时返回-1
- replace 在字符串中查找匹配，并且使用替换字符串替换匹配到的子字符串
- split 使用正则表达式或者一个固定的字符串分割字符串，并将分割后的子字符串存储到数组中。

正则表达式 使用（RegExp）

```
var myRe = /abc/;
var myArray = myRe.exec("abcdef");
console.log(myArray);
```

```
var myRe = /abc/;
var myTest = myRe.test("abcdef");
console.log(myTest);
```

```
var myArray = /abc/.exec("abcdef");
console.log(myArray);
```

```
var myRe = new RegExp('abc');
var myArray = myRe.exec("abcdef");
console.log(myArray);
```

正则表达式 使用（String）

```
var myRe = /abc/g;
var myArray = "abcdef abcdef".match(myRe);
console.log(myArray);
```

```
var myArray = "abcdef abcdef".match(/abc/g);
console.log(myArray);
```

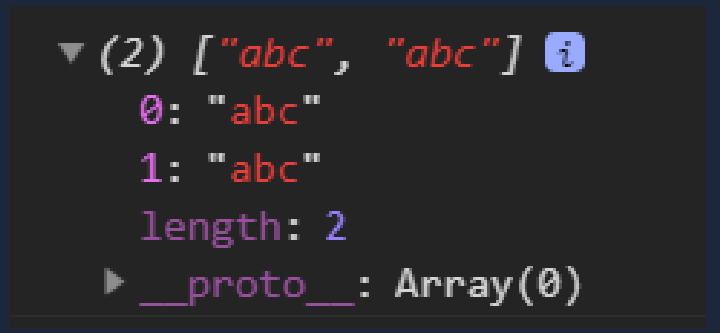
```
var mySearch = "abcdef".search(/abc/);
console.log(mySearch);
```

```
var myReplace = "abcdef".replace(/abc/, '123');
console.log(myReplace);
```

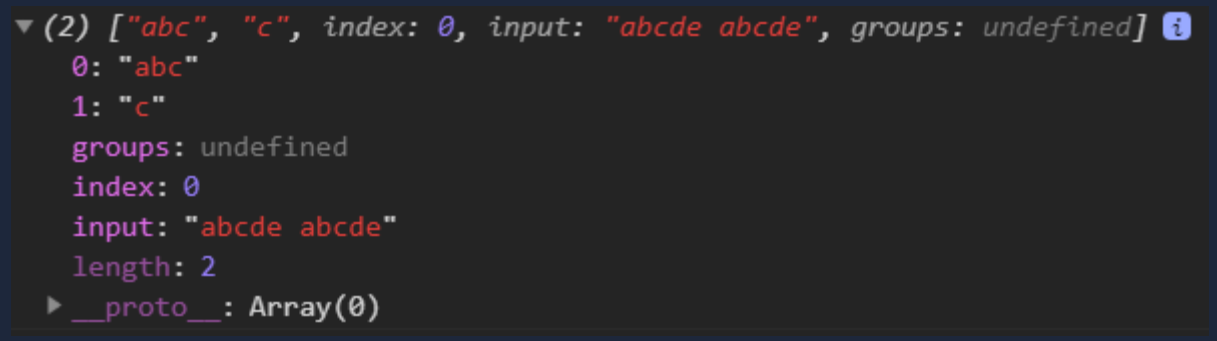
```
var myArray = "abcdefabcdef".split(/b/);
console.log(myArray);
```

正则表达式 返回信息

```
var myRe = /abc/g;
var myArray = "abcdef abcdef".match(myRe);
console.log(myArray);
```




```
var myRe = /ab(c)/;
var myArray = myRe.exec("abcdef abcdef");
console.log(myArray);
```



正则表达式 返回信息

对象	属性或索引	描述	在例子中对应的值
myArray		匹配到的字符串和所有被记住的子字符串。	[“abc", “c"]
	index	在输入的字符串中匹配到的以0开始的索引值。	0
	input	初始字符串。	“abcde abcde"
	[0]	匹配到的所有字符串（并不是匹配后记住的字符串）	“abc"
myRe	lastIndex	下一个匹配的索引值。	3
	source	模式文本。在正则表达式创建时更新，不执行。	“ab(c)"

正则表达式 标志

- g 全局匹配
- i 不区分大小写匹配
- m 多行匹配
- y 粘性匹配，匹配从目标字符串的当前位置开始

正则表达式 特殊字符匹配

- \ 转义字符前缀，使用 new RegExp(“pattern”) 的时候要将 \ 进行转义，因为 \ 在字符串里面也是一个转义字符。
- ^ 匹配输入的开始，如果 多行标志 为 true ，也匹配换行符后紧跟的位置
- $ 匹配输入的结束，如果 多行标志 为 true ，也匹配换行符前的位置
- * 匹配前一个表达式0次或多次
- + 匹配前一个表达式1次或多次
- ? 匹配匹配前一个表达式0次或者1次
- .（小数点）匹配除换行符之外的任何单个字符
- x|y 匹配 x 或者 y

正则表达式 带括号的匹配

- (x) 匹配 x 并且记住匹配项，捕获括号
  - 一个正则表达式模式使用括号，将导致相应的子匹配被记住。例如，/a(b)c /可以匹配字符串“abc”，并且记得“b”。回调这些括号中匹配的子串，使用数组元素[1],……[n]。
- (?:x) 匹配 x 但是不记住匹配项，非捕获括号。
- x(?=y) 匹配 x ，条件是当 x 后面跟着 y ，叫做正向肯定查找，y 不是匹配结果的一部分
- x(?!y) 匹配 x ，条件是 x 后面不跟着 y ，叫做正向否定查找

正则表达式 其他括号匹配

- {n} n为正整数，匹配前面一个字符发生了n次
- {n,m} n和m都是整数。匹配前面的字符至少n次，最多m次
- [xyz] 字符集合。匹配方括号中的任意字符，可以使用 -（破折号）来指定一个字符范围，[a-z]。
- [^xyz] 反向字符集。匹配没有包含在方括号中的字符。
- [\b] 匹配一个退格

正则表达式 转义字符匹配

- \d 匹配一个数字
- \D 匹配一个非数字字符
- \s 匹配一个空白字符，包括空格、制表符、换页符和换行符。
- \S 匹配一个非空白字符。
- \w 匹配一个单字字符（字母、数字或下划线）
- \W 匹配一个非单字字符。
- \b 匹配一个单词的边界。
- \B 匹配一个非单词边界。

正则表达式 转义字符匹配

- \f 匹配一个换页符
- \n 匹配一个换行符
- \r 匹配一个回车符
- \t 匹配一个水平制表符
- \v 匹配一个垂直制表符
- \cX 当X是处于A到Z之间的字符的时候，匹配字符串中的一个控制符
- \0 匹配NULL字符
- \xhh 匹配十六进制数字
- \uhhhh 匹配十六进制数字
- \u{hhhh} 使用Unicode值匹配十六进制数字

正则表达式 \n 与 $n

- \n 为整数，返回最后第n个子子捕获的子字符串
- $n 表示 n 个括号的子字符串匹配
- \n 是用在正则表达式的匹配环节。在正则表达式的替换环节，则要使用像 $n 这样的语法

```
var myRe = /(a)(b)/;
var myArr = (myRe).exec('abab');
console.log(myArr);// ab
```

```
var myRe = /(a)(b)\1\2/;
var myArr = (myRe).exec('abab');
console.log(myArr);// abab
```

```
var re = /(\w+)\s(\w+)/;
var str = "John Smith";
var newstr = str.replace(re, "$2, $1");
console.log(newstr); // Smith, John
```

正则表达式 应用

```
// 手机号码
var regMobile = /^0?1[3|4|5|8][0-9]\d{8}$/;
var mflag = regMobile.test(15225252525);
console.log(mflag);
```

```
function setHTTP() {
    // 判断接口地址
   var url = location.href;
    if (url.search(/test\.go\.163/) > -1) {//测试地址
       urlHost = url.search(/auto/) > -1 ? "//test.go.163.com/api/auto" : "//test.go.163.com/api/go";
    } else {//正式地址
       urlHost = url.search(/go\.163/) > -1 ? "//go.163.com/api" : "//s.auto.163.com/api";
    }
}
```

正则表达式 应用

```
var u = navigator.userAgent;
var ua =  { //移动终端版本信息
    mobile: u.search(/(iPhone|iPod|Android|ios|Mobile)/i)>-1, //是否为移动终端
    pc: u.search(/(iPhone|iPod|Android|ios|Mobile)/i) === -1, //是否为pc终端
    ios: u.search(/\(i[^;]+;( U;)? CPU.+Mac OS X/)>-1, //是否为ios终端
    android: u.search(/Android/) > -1, //是否为android终端
    weixin: u.search(/MicroMessenger/i)>-1, //是否为微信客户端
    newsapp: u.search(/NewsApp/) > -1//是否为网易新闻客户端
};
```

```
//获取链接参数方法  getPara("参数名");
 function getPara(paraName){
    var urlPara = location.search;
    var reg = new RegExp("[&|?]"+paraName+"=([^&$]*)", "gi");
    var a = reg.test(urlPara);
    return a ? RegExp.$1 : "";
}
```

正则表达式 应用

```
// 引入一个文件夹内的多个文件:
const assetsObject = require.context("../assets", true, /\.(png|jpg)$/i);
const assets = assetsObject.keys().map(item=>{return assetsObject(item);});
//assets --> ['assets/img.ae07488d.jpg','assets/img2.ae07488d.jpg'...]
```

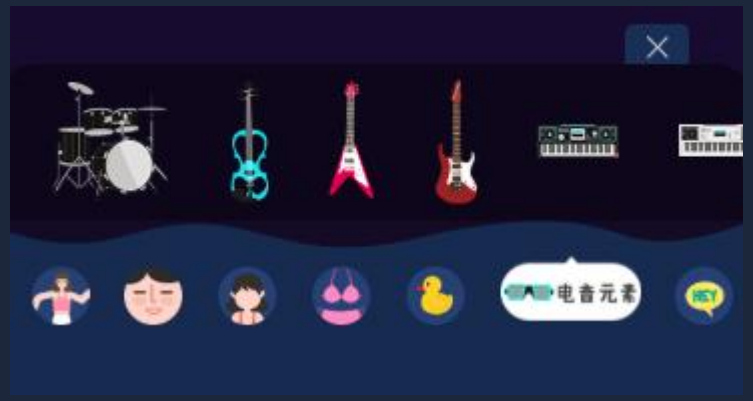
```
// 参数:
// dir:目录路径，例如../assets
// filter:正则表达式，例如: filter传入/^aaa/ 作用为: 返回dir下以aaa开头的上下文。
// 注: 如果正则中有组的话，会返回自一个组列表(这种情况是只想获取比如index号的情况);
// extension,是否需要文件后缀名

@each $item in readdir('../assets',/^clothes_(/d+)/){
.clothes_$item{
        background: url("../assets/clothes_$item.png");
    }
}
```

正则表达式 应用

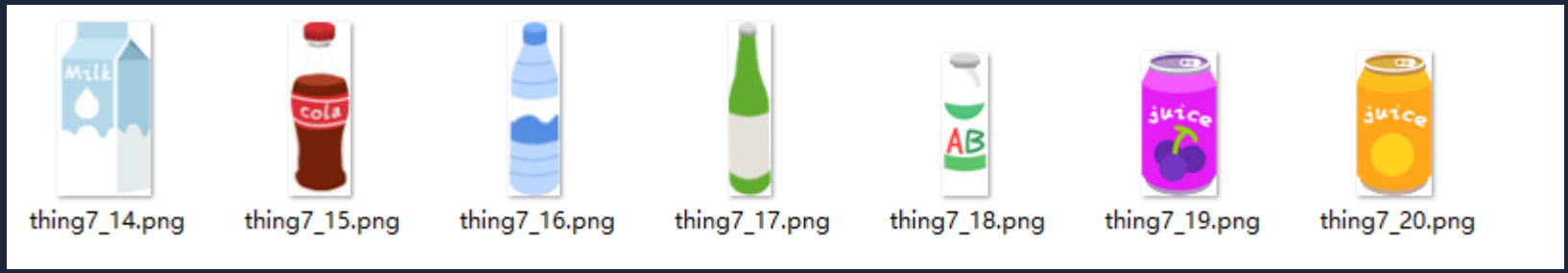

```
@each $item in readdir('../assets/',/^item(\d+)/){
    .nav_item_$item{
        background: url("../assets/item$item.png") no-repeat;
    }
    .nav_item_$item.ani{
        background: url("../assets/item_ac_$item.png") no-repeat;
    }
}
```

正则表达式 应用




```
&.thing_type_8{
    @each $item in readdir('../assets/',/^thing8_(\d+)/){
        .thing_$item{
            background: url("../assets/thing8_$item.png") no-repeat;
        }
    }
}
```

正则表达式 应用




```
const assets = {
    7:require.context("../assets/pixi/", true, /thing7_/i),
};

const getName = (navIndex,thingIndex) => {
       return assets[navIndex].keys()[thingIndex-1].match(/([^\/]+)\.(png|jpg)/)[1];
}
const name = getName(navIndex,thingIndex);

const thingSprite = new Sprite(loader.resources[name].texture);

app.stage.addChild(thingSprite);
```

THANK YOU