JS实践 探测器与正则表达式

探测器

实践-探测器

```
!function(global) {
    function DetectorBase(configs) {
        if (!this instanceof DetectorBase) {
            throw new Error('Do not invoke without new.');
        }
        this.configs = configs;
        this.analyze();
    }

    DetectorBase.prototype.detect = function() {
        throw new Error('Not implemented');
    };

    DetectorBase.prototype.analyze = function() {
        console.log('analyzing...');
        this.data = "###data###";
    };

    function LinkDetector(links) {
        if (!this instanceof LinkDetector) {
            throw new Error('Do not invoke without new.');
        }
        this.links = links;
        DetectorBase.apply(this, arguments);
    }

    function ContainerDetector(containers) {
        if (!this instanceof ContainerDetector) {
            throw new Error('Do not invoke without new.');
        }
        this.containers = containers;
        DetectorBase.apply(this, arguments);
    }

    // inherit first
    inherit(LinkDetector, DetectorBase);
    inherit(ContainerDetector, DetectorBase);
```

实践-探测器

```
// expand later
LinkDetector.prototype.detect = function() {
    console.log('Loading data:' + this.data);
    console.log('Link detection started.');
    console.log('Scaning links: ' + this.links);
};

ContainerDetector.prototype.detect = function() {
    console.log('Loading data:' + this.data);
    console.log('Container detection started.');
    console.log('Scaning containers: ' + this.containers);
};

// prevent from being altered
Object.freeze(DetectorBase);
Object.freeze(DetectorBase.prototype);
Object.freeze(LinkDetector);
Object.freeze(LinkDetector.prototype);
Object.freeze(ContainerDetector);
Object.freeze(ContainerDetector.prototype);

// export to global object
Object.defineProperties(global, {
    LinkDetector : {value : LinkDetector},
    ContainerDetector : {value : ContainerDetector},
    DetectorBase : {value : DetectorBase}
});

function inherit(subClass, superClass) {
    subClass.prototype = Object.create(superClass.prototype);
    subClass.prototype.constructor = subClass;
}
}(this);

var cd = new ContainerDetector('#abc #def #ghi');
var ld = new LinkDetector('http://www.taobao.com http://www.tmall.com
http://www.baidu.com');

cd.detect();
ld.detect();
```

实践-探测器

```
analyzing...
analyzing...
Loading data:###data###
Container detection started.
Scaning containers: #abc #def #ghi
Loading data:###data###
Link detection started.
Scaning links: http://www.taobao.com http://www.tmall.com http://www.baidu.com
```

正则表达式的简单见绍

regexp对象方法
RegExp 对象有 3 个方法：test()、exec() 以及 compile()。
test() 方法检索字符串中的指定值。返回值是 true 或 false。
exec() 方法检索字符串中的指定值。返回值是被找到的值。如果没有发现匹配，则返回 null。
compile() 方法用于改变 RegExp。compile() 既可以改变检索模式，也可以添加或删除第二个参数。
new RegExp(pattern, attributes);参数 pattern 是一个字符串，指定了正则表达式的模式或其他正则表达式。
参数 attributes 是一个可选的字符串，包含属性 "g"、"i" 和 "m"，分别用于指定全局匹配、区分大小写的匹配和多行匹配。ECMAScript 标准化之前，不支持 m 属性。如果 pattern 是正则表达式，而不是字符串，则必须省略该参数。
正则表达式的三个
global/全局性
ignorecase/大小写
multiline多行匹配

正则基础

.	任意字符（除换行符以外：\n, \r, \u2028 or \u2029）	/.../.test('1a@');
\d	数字0-9	/\d\d\d/.test('123');
\D	非\d，即不是数字0-9的字符	/\D\D\D/.test('ab!');
\w	数字0-9，或字母a-z及A-Z（大小写），或下划线	/\w\w\w\w/.test("aB9_");
\W	非\w	/\W\W\W/.test("@!#");
\s	空格符、TAB、换页符、换行符	/\sabc/.test(" abc");
\S	非\s	
\t \r \n \v \f	tab 回车 换行 垂直制表符 换页符	

范围符号

[...]	字符范围	[a-z] [0-9] [A-Z0-9a-z_]
[^...]	字符范围以外	[^a-z] [^abc]
^	行首	^Hi
$	行尾	test$
\b	零宽单词边界	\bno
\B	非\b	

分组

(x)	分组，并记录匹配到的字符串	/(abc)/
\n	表示使用分组符(x)匹配到的字符串	/(abc)\1/.test('abcabc');
(?:x)	仅分组	/(?:abc)(def)\1/.test('abcdefdef');

重复

x* x+	重复次数>=0 重复次数>0 贪婪算法	正则表达式：abc*将匹配ab、abc、abccccc 正则表达式：abc+将匹配abc、abcccc、却不匹配ab
x*? x+?	同x*, x+，非贪婪算法	正则表达式：abc*?在字符串abccccc中将匹配ab，abc+?则匹配abc。
x?	出现0或1次	
x|y	x或者y	x|y匹配x，也匹配y 再比如：ab|cd|ef匹配ab或cd或ef
x{n} x{n,} x{n,m}	重复n次，重复>=n次，重复次数x满足：n <=x<=m	x{5}匹配xxxxxoo，不匹配xxo x{1,3}匹配x，xx，xxx

string类型与正则相关的方

- String.prototype.search
- String.prototype.replace
- String.prototype.match
- String.prototype.split

```
"abcabcdef" .search(/(abc)\1/); // 0
"aabbbbcc".replace(/b+?/, "1"); // aa1bbbcc
"aabbbbcc".match(/b+/); //[ "bbbb" ]
"aabbbbccbbaa".match(/b+/g); // ["bbbb", "bb"]
"aabbbbccbbaa" .split(/b+/); // ["aa", "cc", "aa"]
```