JavaScript 使用

JavaScript 的正则表达式是由 RegExp 对象表示的，同时也可以使用正则表达式字面量。

1. 使用 RegExp 对象

```
let pattern = new RegExp("pattern", "flags");
```

`pattern` 是字符串形式的正则表达式模式。`flags` 是字符串形式的修饰符，可以包含 `i`, `g`, `m` 等。

```
let pattern = new RegExp("abc", "i");           // 匹配 "abc"，不区分大小写
let globalPattern = new RegExp("abc", "g");     // 匹配所有的 "abc"
let multilinePattern = new RegExp("^abc", "m"); // 匹配每一行的开头是 "abc"
```

2. 使用字面量

```
let pattern = /pattern/flags;
```

`pattern` 是正则表达式的模式，可以包含字符、字符集、量词等。

`flags` 是修饰符，可以是以下之一或它们的组合：

- `i`：忽略大小写匹配。
- `g`：全局匹配，匹配所有符合条件的字符串。
- `m`：多行匹配，`^` 和 `$` 匹配每一行的开头和结尾。

```
let pattern = /mozilla/i;
```

3. 常用的正则表达式方法

3.1. test

`test` 方法用于检测字符串是否匹配正则表达式，返回布尔值：

```
let pattern = /\d+/;
let result = pattern.test("123abc"); // true
```

上例检测字符串 `"123abc"` 是否包含一个或多个数字。

3.2. exec

`exec` 方法返回第一个匹配的结果数组，或者在没有匹配时返回 `null`：

```
let pattern = /\d+/;
let result = pattern.exec("123abc");  // ["123"]
```

上例在字符串 `"123abc"` 中查找第一个匹配模式 `\d+`（即一个或多个数字）的子字符串，并返回包含匹配结果的数组 `["123"]`。

3.3. match

`match` 方法在字符串中查找一个或多个匹配，返回一个包含匹配结果的数组：

```
let pattern = /\d+/;
let result = "123abc".match(pattern);  // ["123"]
```

上例在字符串 `"123abc"` 中查找第一个匹配模式 `\d+`（即一个或多个数字）的子字符串，并返回包含匹配结果的数组 `["123"]`。

与 `exec` 方法相比，`match` 方法用于在字符串中查找第一个匹配，但返回结果的形式略有不同。

3.4. search

`search` 方法返回字符串中第一个匹配的索引，如果没有匹配则返回 -1：

```
let pattern = /\d+/;
let result = "abc123".search(pattern);  // 3
```

上例在字符串 `"abc123"` 中查找是否包含匹配模式 `\d+`（即一个或多个数字）的子串，并返回匹配的子串在原字符串中的索引，即返回 3。

3.5. replace

`replace` 方法用指定的字符串或函数替换匹配的子串：

```
let pattern = /\d+/;
let result = "abc123".replace(pattern, "X");  // "abcX"
```

上例将字符串 `"abc123"` 中匹配模式 `\d+`（即一个或多个数字）的子串替换为字符串 `"X"`，返回替换后的新字符串 `"abcX"`。

3.6. split

`split` 方法使用正则表达式或指定的子字符串拆分字符串，并返回一个数组：

```
let pattern = /\s+/;
let result = "This is a sentence".split(pattern);  // ["This", "is", "a",
"sentence"]
```

上例将字符串 `"This is a sentence"` 根据空白字符拆分为一个数组，每个数组元素都是原字符串中的一个单词，返回的结果是 `["This", "is", "a", "sentence"]`。