**转义的各种情况**

* + [⭐ 可无脑转义元字符: ^ $ . \* + ? | \ / ( ) [ ] { } - ,](https://blog.csdn.net/weixin_44100002/article/details/114373990#____________________3)
  + [⭐ ^ $ . \* + ? | \ / ( ) [ 等字符，不在字符组内，都需要转义](https://blog.csdn.net/weixin_44100002/article/details/114373990#______________17)
  + [⭐ 不在字符组内，不用转译右中括号；在字符组内，需要转译右中括号](https://blog.csdn.net/weixin_44100002/article/details/114373990#__18)
  + [⭐ js字符串中的反斜杠也需要转义](https://blog.csdn.net/weixin_44100002/article/details/114373990#_js_26)
    - * [✅字符组内需要转义的字符： ^ - [ ] \](https://blog.csdn.net/weixin_44100002/article/details/114373990#______39)
      * [✅要匹配字符串"[abc]"，只需要转义左方中括号——[](https://blog.csdn.net/weixin_44100002/article/details/114373990#abc_55)
      * [✅要匹配字符串"{abc}"，只需要转义左大括号——{](https://blog.csdn.net/weixin_44100002/article/details/114373990#abc_73)
  + [其余情况：](https://blog.csdn.net/weixin_44100002/article/details/114373990#_84)
    - * [✅括号，即()，需要前后括号都转义](https://blog.csdn.net/weixin_44100002/article/details/114373990#_87)
      * [✅正向先行断言和负向先行断言，即`(?=p)`和`(?!p)`中，需要转义`?`，`=`和`!`不用转译](https://blog.csdn.net/weixin_44100002/article/details/114373990#pp_88)
      * [✅非捕获分组，即`(?:p)`中，需要转义`?`，`:`不用转译](https://blog.csdn.net/weixin_44100002/article/details/114373990#p_107)
      * [✅正向回溯和负向回溯，即`(?<=p)`和`(?<!p)`中，需要转义`?`，`<`、`=`和`!`不用转译](https://blog.csdn.net/weixin_44100002/article/details/114373990#pp_114)

所谓元字符，就是正则中有特殊含义的字符。  
所有结构里，用到的元字符总结如下：

**⭐ 可无脑转义元字符: ^ $ . \* + ? | \ / ( ) [ ] { } - ,**

当匹配上面的字符本身时，可以无脑一律转义，总之不会出错：

var string = "^$.\*+?|\\/()[]{}=!:-,";

var regex = /\^\$\.\\*\+\?\|\\\/\(\)\[\]\{\}\=\!\:\-\,/;

console.log( regex.test(string) );

// => true

**但这些字符不是什么时候都需要转义，可以看情况不转义。**

**⭐ ^ $ . \* + ? | \ / ( ) [ 等字符，不在字符组内，都需要转义**

**⭐ 不在字符组内，不用转译右中括号；在字符组内，需要转译右中括号**

// ^ $ . \* + ? | \ / ( ) [ 等字符，不在字符组内，都需要转义

var string = "^$.\*+?|\\/()[]"; // 不在字符组内，右侧的中括号可以不转义

var regex = /\^\$\.\\*\+\?\|\\\/\(\)\[]/g;

console.log( string.match(regex) );

// => ['^$.\*+?|\/()[]']

**⭐ js字符串中的反斜杠也需要转义**

var string = "\\"; // js中的反斜杠也需要转义

var regex = /\\/g;

console.log( string.match(regex) );

// => ["\"]

var string = "\"; // 不转译会报错

var regex = /\\/g;

console.log( string.match(regex) );

// Uncaught SyntaxError: Invalid or unexpected token

**✅字符组内需要转义的字符： ^ - [ ] \**

跟字符组相关的元字符有[]、^、-。

因此在会引起歧义的地方进行转义。

例如开头的^必须转义，不然会把整个字符组，看成反义字符组。

var string = "^$.\*+?|\\/[]{}=!:-,";

var regex = /[\^$.\*+?|\\/\[\]{}=!:\-,]/g; // 右侧的中括号必须要转义

console.log( string.match(regex) );

// => ["^", "$", ".", "\*", "+", "?", "|", "\", "/", "[", "]", "{", "}", "=", "!", ":", "-", ","]

**✅要匹配字符串"[abc]"，只需要转义左方中括号——[**

我们知道[abc]，是个字符组。如果要匹配字符串"[abc]"时，该怎么办？

可以写成/\[abc\]\

也可以写成/\[abc]/，测试如下：

var string = "[abc]";

var regex = /\[abc]/g;

console.log( string.match(regex)[0] );

// => "[abc]"

因为后面的方括号不构成字符组，正则不会引发歧义，自然不需要转义。  
同理，要匹配字符串"{3,5}"，只需要把正则写成/\{3,5}/即可。

**✅要匹配字符串"{abc}"，只需要转义左大括号——{**

另外，我们知道量词有简写形式{m,}，却没有{,n}的情况。虽然后者不构成量词的形式，但此时并不会报错。  
匹配字符串直接写作 /{,n}/g，**此时不需要转义大括号**

var string = "{,n}";

varregex = /{,n}/g;

console.log( string.match(regex) );

//=> ["{,n}"]

**其余情况：**

**✅括号，即()，需要前后括号都转义**

**✅正向先行断言和负向先行断言，即(?=p)和(?!p)中，需要转义?，=和!不用转译**

var result = "he??lo".replace(/(?=\?)/g, '#');

console.log(result);

// => he#?#?lo

var result = "he==o".replace(/(?==)/g, '#');

console.log(result);

// => he#=#=o

var result = "he!!o".replace(/(?!!)/g, '#');

console.log(result);

// => #h#e!!#o#

**✅非捕获分组，即(?:p)中，需要转义?，:不用转译**

var result = "he::o".replace(/(?::)/g, '#');

console.log(result);

// => he##o

**✅正向回溯和负向回溯，即(?<=p)和(?<!p)中，需要转义?，<、=和!不用转译**

var result = "he<<o".replace(/(?<!<)/g, '#');

console.log(result);

// => #h#e#<<o#

var result = "he<<o".replace(/(?<=<)/g, '#');

console.log(result);

// => he<#<#o