## 1：正则的概念

正则表达式(regular expression)是一个描述字符规则的对象。可以用来检查一个字符串是否含有某个子字符串，将匹配的子字符串做替换或者从某个字符串中取出符合某个条件的子串等。

为什么要用正则：

前端往往有大量的表单数据校验工作，采用正则表达式会使得数据校验的工作量大大减轻。常用效果：邮箱、手机号、身份证号等。

## 2：创建方式

第一种方式：

var reg = new RegExp(“study”， “ig”); // 第二个参数为修饰符

i：表示忽略大小写。

g：表示全局匹配，查找所有匹配而非在找到第一个匹配后停止。

m：多行匹配。

第二种方式：

var reg = /study/ig;

## 3：正则对象方法

test：检索字符串中指定的值。返回true或false。

exec：用于检索字符串中的正则表达式的匹配。返回一个数组，其中存放匹配的结果。如果未找到匹配，则返回值为null。

注：如果没有指定g修饰符，那么每次匹配都是从头开始匹配，如果指定g修饰符以后，下次匹配则从上次匹配的结束位置开始匹配。

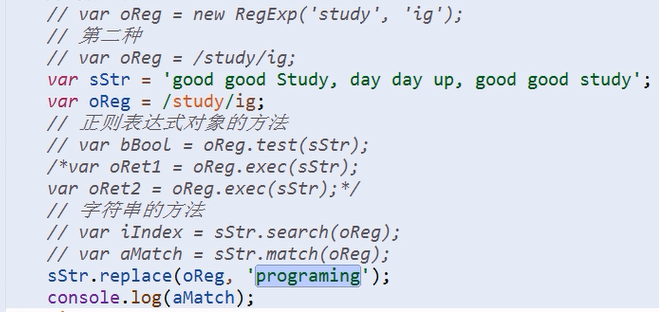
## 4：字符串函数

search：检索与正则表达式相匹配的值。返回字符串中第一个与正则表达式相匹配的子串的起始位置。如果没有找到则返回-1。

match：找到一个或多个正则表达式的匹配。返回数组包含所有匹配的字符串。

replace：替换与正则表达式匹配的子串。

split：把字符串分割为字符串数组。



## 5：正则表达式构成

正则表达式是由普通字符（例如字符a到z）以及特殊字符（称为元字符）组成的文字模式。正则表达式作为一个模板，将某个字符模式与所搜索的字符串进行匹配。

**元字符---限定符：**

限定符可以指定正则表达式的一个给定组件必须要出现多少次才能满足匹配**。**

\*：匹配前面的子表达式零次或多次。（从左向右读取，只要不是条件的全都返回空，空格也返回空）

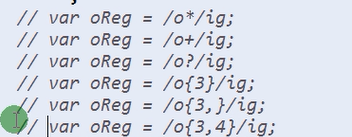
+：匹配前面的子表达式一次或多次。（返回将没有空的字符串）

？：匹配前面的子表达式零次或一次。（从左向右读取，只要不是条件的全都返回空，空格也返回空）

{n}：匹配确定n次。

{n,}：至少匹配n次。

{n, m}：最少匹配n次且最多匹配m次。



注：在限定符后紧跟 ? 则由贪婪匹配变成非贪婪匹配。



**元字符---字符匹配符：**

字符匹配符用于匹配某个或某些字符。

[xyz]：字符集合。匹配所包含的任意一个字符。

[^xyz]：负值字符集合（^表示非的意思）。匹配未包含xyz的任意字符。

[a-z]：字符范围。匹配指定范围内的任意字符。

[^a-z]：负值字符范围。匹配任何不在指定范围内的任意字符。（若有空格，将也会返回）

例如：[0-9]、[0-9a-z]、[0-9a-zA-Z]

\d：匹配一个数字字符。

\D：匹配一个非数字字符。

\w：匹配包含下划线的任何单词字符。等价于[a-z0-9A-Z\_]

\W：匹配任何非单词字符。等价于[^a-z0-9A-Z\_]

\s：匹配任何空白字符。

\S：匹配任何非空白字符。

.：匹配除”\n”之外的任何单个字符。

**元字符---定位符：**

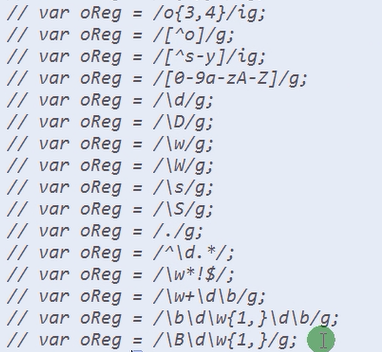
定位符可以将一个正则表达式固定在一行的开始或结束。也可以创建只在单词内或只在单词的开始或结尾处出现的正则表达式。

^：匹配输入字符串的开始位置。

$：匹配输入字符串的结束位置。

\b：匹配一个单词边界，也就是单词和空格间的位置。置于前则限定开头是XX的单词，置于末尾限定结尾是XX的单词；

\B：匹配非单词边界（不以XX开头或结尾的）。

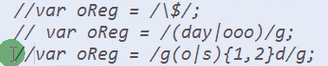


**元字符---转义符：**

\：用于匹配某些特殊字符。

**元字符---选择匹配符：**

|：可以匹配多个规则。



**特殊用法：**

()：捕获性分组，也称为子表达式。使用\1、\2、\3…对子组的引用。当子组发生嵌套时，顺序是从外到内。

(?:pattern)：非捕获性分组。匹配pattern但不获取匹配结果。也就是说这是一个非获取匹配，不进行存储供以后使用。这在使用 "或" 字符 (|) 来组合一个模式的各个部分是很有用。例如， 'industr(?:y|ies) 就是一个'industry|industries' 更简略的表达式。

(?=pattern)：正向预查，在任何匹配 *pattern* 的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如， 'Windows (?=95|98|NT|2000)' 能匹配 "Windows 2000" 中的 "Windows" ，但不能匹配 "Windows 3.1" 中的 "Windows"。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。

(?!pattern)：负向预查，在任何不匹配*pattern* 的字符串开始处匹配查找字符串。这是一个非获取匹配，也就是说，该匹配不需要获取供以后使用。例如‘Windows (?!95|98|NT|2000)’ 能匹配 “Windows 3.1” 中的 “Windows”，但不能匹配 “Windows 2000” 中的 “Windows”。预查不消耗字符，也就是说，在一个匹配发生后，在最后一次匹配之后立即开始下一次匹配的搜索，而不是从包含预查的字符之后开始。



## 应用：

1：掌握正则的创建方式。

2：掌握正则表达式的语言。

3：邮政编码检测。

4：文件格式监测。

5：字符串首尾去空格。

6：邮件格式监测。

## 7：手机号监测

## 8：身份证监测

## 9：日期监测

## 10：中文监测

unicode编码中文监测：/^[\u2E80-\u9FFF]+$/

## 11：用户名监测

正则：/^[a-z0-9\_-]{3,16}$/

## 12：字符串过滤

用字符串replace方法。

## 13：常用的正则

邮箱：

/^([a-z0-9\_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/

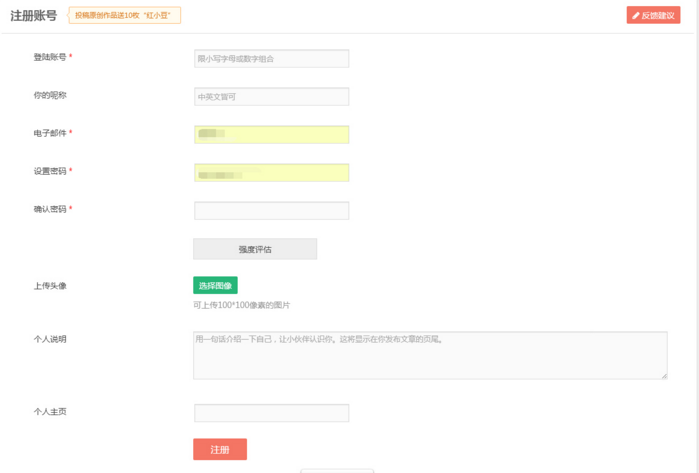
/^[a-z\d]+(\.[a-z\d]+)\*@([\da-z](-[\da-z])?)+(\.{1,2}[a-z]+)+$/（？）

URL：/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]\*)\*\/?$/

HTML标签：/^<([a-z]+)([^<]+)\*(?:>(.\*)<\/\1>|\s+\/>)$/（？）

## 应用：

1：表单的应用。



## 综合应用：

1：掌握正则的封装。

2：掌握综合表单验证。