**正则表达式：**它是用来处理字符串的，验证字符串是否符合某个规则(正则匹配)，或者是把字符串中符合规则的内容获取出来(正则的捕获)的一个规则

一个正则就是由各种元字符组成的，而且//之间包起来的都是元字符

创建正则也有两种方式(有一些区别的)

var reg = /\d/;//**字面量方式**

var reg = new RegExp("\\d");//**实例创建方式**

**实例创建个字面量创建的区别:**

1、实例创建的方式需要多转译一次，把具有特殊意思并且带\的都要多转译一次

var reg = /\d/;

var reg = new RegExp("\\d");

2、在字面量方式中，用"//"包起来的都是元字符，有具有特殊意义的元字符、代表出现次数的量词元字符、就代表本身意思的普通元字符

字面量方式无法识别变量，实例创建的方式可以

var c = "w100";

var reg = /^"+c+"$/;//以"开头，出现一到多次，然后是c出现一到多次，最后以"结尾，而不是字符串拼接

var reg = new RegExp("^" + c + "$");//只能包含w100的 /^w100$/

**具有特殊意义的元字符**

^：以某一个元字符开始

$：以某一个元字符结束

.：匹配除了\n以外的任何字符

x|y：x或者y

()：分组，或者是大正则中的一个小正则

[xyz]：x、y、z三个中的一个

[^xyz]：除x、y、z三个中的任何一个 这里的^是取反的意思

[a-z]：a-z之间的任意一个字符

[^a-z]：除了a-z之间的任意一个字符

\d：0-9之间的任意一个数字

\w：数字、字母、下划线 [a-zA-Z\_0-9]

\n：一个换行

\b：匹配一个边界(字母和空格之间也是边界)

\s：匹配任何的空白字符：空格、制表符、换页符

\：转译字符

**量词元字符(代表出现多少次的)**

\*：出现零到多次

+：出现一到多次

?：出现零或一次

{n}：出现n次

{n,}：出现n到多次

{n,m}：出现n到m次

**普通元字符:**所有的东西都可以作为我们的元字符，除了特殊的以外都代表本身的意思

var reg=/^zhufeng$/;//匹配的只有"zhufeng"

var reg=/^周$/;//周 只代表了本身的意思

**正则中还包含修饰符：i(ignoreCase 忽略大小写)、m(multiline 匹配换行)、g(global 全局匹配)**

var reg = /^[a-z]$/i;

var reg = new RegExp("^[a-z]$", "i");

console.log(reg.test("Z"));

[]包起来的里面都是代表本身意思的字符，没有特殊意义，例如：下面的.就是小数点

var reg=/^[12.5]$/;

var reg = /^1.5$/;//这里面的.代表的是除了换行外的任意字符而不是小数点，如果我想让他只代表小数点 var reg=/^1\.5$/; 需要用转译字符来处理（\.代表的就是一个小数点）

var reg = /^[0-9]$/;// ==> /^\d$/

var reg = /^[12-65]$/;//这个正则不是12-65之间，是1、2-6之间的一个、5 三种之间的一个

正则中的匹配方法

**test：**匹配字符串(验证某个字符串是否符合正则的规则的) 不符合返回false，符合返回true

**exec:**正则捕获的方法

1、首先拿我们的字符串个正则进行匹配，如果不匹配，捕获的结果是null

2、只有匹配才按照我们的规则进行捕获

正则的捕获是**贪婪的**，默认按照匹配最长的捕获，例如：捕获的是2015而不是2

? 如果放在普通元字符后面它就是量词元字符，代表出现0-1次；? 如果放在量词元字符后面，代表取消正则的贪婪捕获；?: 在分组中的作用是 只匹配不捕获

正则的捕获是**懒惰的**，默认只捕获第一个匹配到的，例如：只得到了2015但是没有获取2014

可以加一个全局的修饰符g，需要捕获几次我们就执行几次，这样就能都捕获到了

var reg = /\d+?/g;

var str = "zhufeng2015peixun2014";

var ary = [];

var res = reg.exec(str);

while (res) {

ary.push(res[0]);

res = reg.exec(str);

}

console.log(ary);

通过exec如果可以获取到内容的话是一个数组(跟平时的长得不是很一样)

例如：["2015", index: 7, input: "zhufeng2015peixun2014"]

第一项是捕获到的内容；index是捕获开始的索引；input我们捕获的那个原始的字符串

字符串中中的**match**的方法，可以**一次捕获到所有的内容**

不加全局匹配符g的时候，和exec一样，大正则和分组的内容都可以捕获到

加g的时候，虽然执行一次就可以把所有大正则匹配的内容都捕获到了(而exec需要多次)，但是**math在加上g的时候不能捕获小分组中的内容**

var reg = /\d+/g;

var str = "珠峰2015第四期51人，周末班15人，全日制32人";

console.log(str.match(reg)); // ["2015", "51", "15", "32"]

字符串中的**replace**方法，第一个参数如果写的是正则，每匹配一次replace就执行一次(不要忘记加g)

字符串中的replace方法，可以捕获正则匹配的内容，如果我第二个参数传递的是一个function，每捕获一次我们的函数就执行一次，不光光是执行，执行的时候浏览器还默认传递了参数

需求：把所有的单词的首字母大写

var str = "welcome to zhu feng pei xun! good good study, day day up!";

var reg = /(?:^| +)?([a-z]+)(?: +|$)?/g;

str = str.replace(reg, function () {

//console.dir(arguments);

//我们当前能够进行捕获的分组只有一个，我们传进来的参数有四个

//第一个：大正则捕获的内容 ==>exec ary[0]

//第二个：第一个分组捕获的内容（我们想操作的就是这个） ==>exec ary[1]

//第三个：大正则捕获的开始索引 ==> exec index

//第四个：原始字符串 ==> exec input

var tar = arguments[1];//每一次匹配，第一个分组捕获的内容

//var tar = RegExp.$1;//每一次匹配，第一个分组捕获的内容 $2就是第二个分组捕获的内容....(存在兼容问题，我们通常用arguments来处理即可)

var f = tar.charAt(0).toUpperCase();

tar = f + tar.substr(1);

return " " + tar;//用tar把每一次大正则捕获到的内容替换掉

});

console.log(str);

**分组：**用小括号包起来的就是一个分组，也是大正则中的一个小的正则

**1、改变默认优先级**

**2、可以进行分组捕获**

捕获到的是一个数组：第一项是大正则捕获的内容，接下来分别是每一个小分组捕获的内容，index是大正则捕获的开始索引

var reg=/(\d+)([a-z]+)/;

console.log(reg.exec("zhufeng2015peixun2014"));

var ary=["2015peixun", "2015", "peixun", index: 7, input: "zhufeng2015peixun2014"]

"2015peixun" 大正则捕获的内容 ary[0]

"2015" 第一个分组捕获的内容 ary[1]

"peixun" 第二个分组捕获的内容 ary[2]

....

我不想捕获第二个分组中的内容，只需要在小括号中加?:即可

**?: 在分组的最前面，只匹配不捕获**

var reg=/(\d+)(?:[a-z]+)/;

console.log(reg.exec("zhufeng2015peixun2014"));

**分组引用:**

当我们发现正则中的某一部分需要和另外一部分匹配的内容的一模一样才可以，这样的情况下我们用分组引用来解决

var reg=/^([a-z])([a-z])\2\1$/i;

\1 第一个分组的引用，是必须和第一个分组一模一样的东西才可以

\2 第二个分组的引用，是必须和第二个分组一模一样的东西才可以

console.log(reg.test("woow"));//true

**1、匹配一个有效数字：**正数、负数、0、小数

var reg = /^[+-]?(?:\d|[1-9]\d+)(?:\.\d+)?$/;

[+-]? 验证可能是负数也可能是正数

(?:\d|[1-9]\d+) 一位数的时候0-9的一个数字，两位数及以上0不能作为开头

(?:\.\d+)? 首先可以有小数点也可以没有,如果出现小数点，后面必须至少跟一位数字

**2、只能是正整数**  var reg = /^[1-9]\d\*$/;

**只能是正整数或者零的**  var reg = /^[+]?(\d|[1-9]\d+)$/;

**3、手机号码：**第一位是1，剩余的是数字即可(10位数字)

var reg = /^1\d{10}$/;

**4、年龄18-65：**18、19、2-5 0-9、6 [0-5]

var reg = /^(?:1[8-9])|(?:[2-5]\d)|(?:6[0-5])$/;

**5、中国标准真实姓名：**最多四个，最少两个汉字

var reg = /^[\u4e00-\u9fa5]{2,4}$/;

**6、昵称：**数字、字母、下环线、中线、汉字

var reg = /^(\w|-|[\u4e00-\u9fa5])+$/;

**7、邮箱：**xxx@xxx.xxx.xxx xxx@xxx.xxx

var reg = /^(?:[0-9\_a-z.A-Z-]+)@(?:[0-9a-zA-Z]+)(?:\.[a-zA-Z]{2,4}){1,2}$/;

**8、身份证号：**130 828(省市县) 1990(年) 12(月) 04(日) 06 1(奇数是男 偶数是女) 7(可能是数字，可能是X)

var reg = /^(\d{3})(\d{3})(\d{4})(\d{2})(\d{2})(?:\d{2})(\d)(?:\d|X)$/;

console.log(reg.exec("130828199012040617"));

//["130828199012040617", "130", "828", "1990", "12", "04", "1", index: 0, input: "130828199012040617"]