复习：

1. 背景
2. 环境搭建，写出hello
3. 数据类型

值类型 string number boolean

特殊类型 null undefined

引用类型 Array String

1. 运算符
2. 逻辑结构 if..else.. switch..case

while do..while for for..in

1. 通用小程序
2. 语言中的组件： function object
3. Array：

length

join() slice() splice() concat() push() pop() sort() reverse()

1. String 内容不可变

length

indexOf() lastIndexOf() slice() split() substring() toLowerCase() charAt() charCodeAt() concat() replace() match() search()

目标：

1. 正则表达式——5.5星 重点&难点
2. Math、Boolean、Number、Date——有用
3. 错误处理——理解
4. 正则表达式

Regular Expression，规则的、有模式的、有规律的，表示式——用于描述一种字符串模式的表达式。

定义一个正则表达式对象的两种方式：——RegExp对象不是String对象

1. **var patt1 = /表达式/特性;**
2. **var patt2 = new RegExp(‘表达式’, ‘特性’);**

可以在两种情形下使用一个正则表达式对象：

(1) patt.test( str ); //用正则表达式对象检索字符串对象

(2) str.replace( patt ); //调用String的方法，把正则表达式对象作参数

正则表达式pattern的写法

|  |  |  |
| --- | --- | --- |
| **内容形式** | **含义** | **示例** |
| 字符、数字、汉字 | 匹配这种字符本身 | /com/ /139/ |
| n? | 指定字符出现0、1次 |  |
| n\* | 指定字符出现0~多次 | /ab\*/ /(ab)\*/ |
| n+ | 指定字符出现1~多次 |  |
| n{x} | 指定出现的确切次数 | /x{9}/ |
| n{x,y} | 指定出现的次数>=x且<=y | /ab{2,4}/ |
| n{x,} | 指定出现的次数>=x | /ab{2, }/ |
| [abcd] | 匹配多个之一 |  |
| [^abcd] | 不允许出现多个之一 |  |
| [0-9] | 相当于[0123456789] |  |
| [a-z] | 匹配任意一个小写字母 |  |
| [A-Z] | 匹配任意一个大写字母 |  |
| [A-z] | 匹配任意大小写字母，以及[ \ ] ^ \_ ` |  |
| [A-Za-z] | 匹配任意一个大小写字母 |  |
| [A-Za-z0-9\_$] | 匹配大小写字母、数字、下划线、$其中之一，**注意：[]中元字符可以不用转义** |  |
| 表达式1|表达式2 | 匹配两个表达式其中之一  注意：[]只匹配一个字符；而|匹配是一个完整的表达式 | /abc|xyz/  /m(abc|xyz)/ |
| (表达式) | 使用()创建一个子表达式/表达式小分组。注意：每个()创建的分组会自动分配一个从1开始的编号，在正则表达式中想引用某个分组的值使用“**\编号**”，在正则表达式的外部想引用某个分组的值使用“**$编号**” | /ab\*/  /(ab)\*(cd)\*/  /(ab)(cd)(ef)/  分组ab的编号为1  分组ef的编号为3 |
| \d | 匹配任意一个数字(decimal) | /\d/ <=> /[0-9]/ |
| \D | 匹配任意一个非数字字符 | /\D/ <=> /[^0-9]/ |
| \w | 匹配任意一个单词字符(word)，如大小写字母、数字、\_ | /\w/ <=> /[a-zA-Z0-9\_]/ |
| \W | 匹配任意一个非单词字符 | /\W/ <=> /[^a-zA-Z0-9\_]/ |
| \s | 匹配任意个空白字符(space)， | /\s/ <=> /[ \n\t\r]/ |
| \S | 匹配任意个非空白字符 | /\S/ <=> /[^ \n\t\r]/ |
| . | 匹配除了换行和回车之外的任意一个字符 | /./ <=> /[^\n\r]/ |
| 上面的匹配符都是“字符匹配符”  下面的匹配符都是“位置匹配符”，匹配字符串中的特定位置 | | |
| ^ | 匹配字符串的开头 | /^a/ |
| $ | 匹配字符串的结尾 | /a$/ |
| \b | 匹配单词的边界(boundary) | /\bis\b/ |
| \B | 匹配非单词边界 | /\Bis\b/ |
| ?=x | 匹配后面紧跟指定字符x的字符串，注意：匹配的结果中不包含x | /do(?=not)/ |
| ?!x | 匹配后面没有紧跟指定字符x | /do(?!not)/ |

2.正则表达式中的特性修饰符：

var regexp = /表达式/特性修饰符 ;

特性修饰符可以取下列的一个或多个：

1. **i**  ignoreCase,匹配时忽略大小写
2. **g** global，全局匹配，默认情况下所有的匹配仅进行一次；若声明了全局匹配，则可以把所有的匹配情形全部找出来
3. **m** multiline，执行多行匹配，m只对^和$位置匹配符有影响，默认情况下，不执行多行匹配，即

默认情况下： ^只匹配字符串的开头；$只匹配字符串的结尾

若声明了m： ^可以匹配每一行的开头；$匹配每一行的结尾

练习：

* + 定义可以匹配邮箱的正则表达式
  + 定义可以匹配手机号码的正则表达式
  + 定义可以匹配身份证号码的正则表达式
  + 定义可以匹配中文姓名的正则表达式
  + 定义可以匹配HTML标记(包括单标记和双标记)的正则表达式

3.RegExp类型的常用成员

**成员属性：**

global 只读

ignoreCase 只读

multiline 只读

source 只读，返回正则表达式原始式子

RegExp.$1.... RegExp.$9 其中保存着最近一次成功匹配时指定编号的的子表达式具体匹配的值；即读取这9个变量前，必须要有一次成功的 regexp.test(str) 或者 regexp.exec( str ) 方法的调用。

lastIndex 可读/可写，返回下一次匹配搜索的起始位置——必须声明全局搜索才有此属性——此属性可以用于计算匹配的总次数

**成员方法：**

test(): 需要一个字符串参数，用于验证该字符串是否匹配定义好的模式，返回true/false。该方法会影响lastIndex和$1..$9属性

exec() 执行一次对字符串参数的匹配，返回此次匹配成功的实际子串，以及匹配到的下标；若找不到匹配了返回null。此方法可以多次调用，会不停的影响lastIndex属性——常用于“找到一篇文章中所有的电话/邮箱/钱数”等功能。

compile() 修改和重新编译表达式，可以加快执行速度。

4.String类中可以使用正则表达式作参数的方法

str.方法(patt);

(1)replace( value/patt, replacement) 将指定的子串/正则表达式匹配的内容替换为replacement

(2)split( value / patt ) 使用指定的子串或正则表达式拆分原始字符串，如拆分出一条语句中所有的英文单词，即可以 /\s+/ 来拆分

练习：

(1)判定字符串是否为合法的身份证号码

(2)获取一段HTML文本中所有超链接的href属性的值

(3)去除一个字符串开头和结尾的空白字符

(4)摘取出一段字符串中出现的所有的手机号码

(5)判定一段字符串中是否存在违禁字符，如"发票"/"增值税"/"中奖"/"抽奖"等，有则替换为同等字符数的\*