JS第五周第一天

1.正则的元字符

```
<script>
  console.log(RegExp.prototype);
  //两个方法
  //匹配test 正则.test(字符串)  ->true/false
  //捕获exec 正则.exec(字符串)  ->数组/null

  //元字符和修饰符组成
  var  reg=/123456asdhhslaj/;

  //1.元字符 只要是正则//中任意一个符号都是元字符
  //^  $ :如果 有^和$ 确定 中间没有量词元字符 那么匹配的字符的length就确定了

  //1).特殊元字符
  // \d :0-9 任意一个
  // \D :不是数字0-9中的任意一个

  var reg=/^\d$/;
  // \w  数字 字母  _ 63个
  // \W  除了\w中的任意一个

  // \b  匹配一个边界
  // \B  非边界

  // \s  空白符
  // \S  非空白符

  // \n 换行

  //  . 除了\n以外的任意字符
```

```
    // \  转义符   一般都是将有特殊意义变成本身意义

    //  | 或

    //[]


    //2).普通元字符

    //3).量词元字符   只有放在元字符后面才表示量词元字符
      //* 0-多
      //? 0-1
      //+ 1-多
      //{n} n次
      //{n,} n-多
      //{n,m} n-m
      var reg=/\++/;
</script>
```

2.小括号

```
小括号
优先级
分组

\1 跟第一个分组(第一个())中匹配的内容一样的字符串
\m 没有底那个分组匹配就是\m
```

3.或的问题

```
//将正则一分为二
    //匹配首尾空格
    reg=/^ +| +$/;

    //[]  出现在中括号中的字符，除了-，^ ，?!之外都是本身的意思。
    reg1=/1[7-9]/; // 17 18 19
    console.log(reg1.test("8"));

//匹配数字
    //正数 负数 小数 整数
    //整数部分两位以及两位以上第一位不可以是0
    //0
```

```
reg=/^-?([1-9]\d+|\d)(\.\d+)?$/;
//-? 正负
//([1-9]\d+|\d) 整数部分
//(\.\d+)? 小数部分 要么有(1)要么没有(0)
console.log(reg.test("1.1"));
```

4.正则的捕获

```
exec:返回值是一个数组/null
1.懒惰性:一次捕获一个 解决:加全局修饰符g 进行多次捕获

字符串中的方法match
1).不加g 得到的结果跟exec一样
2).加g 一次全部捕获到 没有index,input,groups属性
3).分组情况下 加g 但是不会捕获分组内容

2.贪婪性:一次捕获符合要求最长的 主要是因为量词元字符的存在导致的 解决：在量词元字符后面加？，就是解决量词的取值

//捕获分组问题
  //exec 的返回值就是多几项 有几个分组就多几项
  //索引0:大正则捕获的内容
  //    1:第一个分组捕获的内容
  //    2:第二个
  //...
  //index:  大正则捕获的内容首字符索引
  //input:  原字符串
  //如果想让分组的内容不被捕获出来加上?:
```

5.构造函数创建正则

```
//里面的内容写在第一个参数""中，注意转义 \写成\\
第二个参数是修饰符
RegExp.$1  第一个分组捕获的内容  $2 第二个... ，  没有就是""
```

6.字符串的方法正则处理

```
split  按照指定字符拆分为数组                str.split(reg)
search
match
```

```
replace(reg,b) 一次全部替换，第一个参数写一个正则且加上全局修饰符g；第二个参数也可以是个函数
str=str.replace(reg,function () {
      //捕获多少次就执行多少次
      //每一次执行函数都默认传参数
      //arguments[0]: 大正则捕获的内容
      //arguments[1]: 第一个分组捕获的内容
      //arguments[2]: 第二个分组捕获的内容
      //....
      //倒数第二项:大正则捕获的内容首字符的索引
      //最后一项:原字符串

      //console.log(arguments);
      //return 的值去替换 大正则捕获的内容
      return arguments[1].toUpperCase()+":¥"+arguments[2];
    });
    console.log(str);
```

7.解析URL

```
<script>
  var url = "www.zhufengpeixun.com?course1=JS&course2=JQ&course3=vue&course4=react&course5=node";
  //{"course1":"JS","course2":"JQ","course3":"vue","course4":"react","course5":"node"};

  //1.拿到?后面的内容
  //console.log(url.split("?"));
  // course1=JS&course2=JQ&course3=vue&course4=react&course5=node
  //方法1
  //思路:
  //1.将url按照?拆分获取?后面的内容
  //2.将其按照&拆分为一个数组arr ["course1=JS", "course2=JQ", "course3=vue", "course4=react", "course5=node"]
  //3.创建一个备用的对象obj={}
  //4.循环遍历数组arr将数组的每一项按照=拆分为一个数组 a  ["course1","JS"]
  //5.将数组 a 第一项当做属性名 第二项当做属性值
  function f1(url) {
    url = url.split("?")[1];
    var arr = url.split('&');
    var obj = {};
    arr.forEach((item) => {
      var a = item.split("=");
      obj[a[0]] = a[1];
    });
    return obj
```

```
  }

  console.log(f1(url));

  //方法2
  //思路
  //1.将url 按照?&=拆分为数组
  //2.数组中第一项没有用删除掉
  //3.循环数组 隔一项循环一次 i+=2
  //4.将arr[i]作为属性名 后一项arr[i+1]作为属性值
  function f2(url) {
    let arr = url.split(/[?&=]/);
    arr.shift();
    //["course1", "JS", "course2", "JQ", "course3", "vue", "course
4", "react", "course5", "node"]
    let obj = {};
    for (let i = 0; i < arr.length; i += 2) {
      obj[arr[i]] = arr[i + 1]
    }
    return obj
  }

  console.log(f2(url));

  //方法3
  //思路
  //1.将url按照?拆分获取?后面的内容
  //2.将=全部替换成 :' 将&全部替换成 ',
  // 注意三点1).想要全部替换必须使用正则加g的形式
  //         2).属性值一定要加上引号 才表示字符串 要不然就是变量
  //         3).最后面还少一个' 不要忘了加上
  //3.将替换好的字符串变成真实对象 使用eval 注意{}问题
  function f3(url) {
    url = url.split("?")[1];
    eval("var obj={" + url.replace(/=/g, ":'").replace(/&/g, "',")
+ "'}");
    return obj;
  }

  console.log(f3(url));


  //方法4
  function f4(url) {
    let reg=/[?&](\w+)=(\w+)/g;
    let obj={};
    url.replace(reg,(...rest)=>{
      obj[rest[1]]=rest[2];
    });
```

```
    return obj;
  }

  console.log(f4(url));

  //方法5
  function f5(url) {
    url = url.split("?")[1];
    let reg=/&?(\w+)=(\w+)/g;
    return eval("({"+url.replace(reg,"$1:'$2',")+"})")
  }

  console.log(f5(url));
</script>
```

8.表单元素

```
autofocus:自动获取焦点
autocomplete:自动记录匹配之前填的内容
pattern :加正则验证
type:可以默认校验
novalidate:给form加的 关闭校验
required:必填的
disabled:不可用

<body>
<form action="9.获取数据.html" autocomplete="on">
    <input type="text" name="user" autofocus pattern="^\\w{6,15}$">
    <input type="text" name="pw" required>
    <!--<input type="password">-->
    <!--<input type="date">-->
    <!--<input type="color">-->
    <input type="number" disabled value="1">
    <input type="email">
    <!--<input type="url">-->
    <!--<input type="file">-->
    <!--<input type="search">-->
    <!--<input type="button">-->
    <!--<input type="checkbox">-->
    <!--<input type="radio">-->
    <!--<input type="range">-->
    <input type="reset">
    <input type="submit">
</form>
</body>
```