day15

- 1.正则表达式概念
  - 正则表达式 是一种中立于语言 用于描述特定字符规则的表达式
  - 关键字 RegExp
  - 子主题
    - ECMAScript提供了正则表达式的构造函数
    - 使用构造函数 创建 正则表达式
    - 正则表达式是引用类型的对象
  - 正则表达式的创建
    - 1.使用构造函数创建
      ```
      var reg = new RegExp('abc'); // 将字符串转作为参数传入
      console.log(typeof reg);//object
      ```
    - 2.使用字面量创建
      ```
      var reg2 = /abc/2;
      console.log(reg2);// /abc/
      ```
  - 特殊字符
    - 正则表达式中有些字符有特殊的语法含义，
      是不能直接使用的，必须使用 \ 进行转义后才能使用
    - .\/*?+()[]{}^|$
  - RegExp.prototype.test();
    - 语法: reg.test(string);
    - 描述: 通过正则表达式reg的规则检测string是否符合reg的规范 并返回一个布尔值
    - ```
      var reg = new RegExp('abc');
      console.log(reg.test('abc'));
      ```
- 2.字符集
  - 概念
    - 在正则表达式中表示在一个范围内匹配字符
    - 字符集使用[]表示
    - 每一个[]表示一个字符
    - 如果^ 出现在中括号中 表示非
    - 注意 ^在不同的场景有不同的应用
      - 在 | 中表示非
      - 在其余地方表示以后面的表达式为开头
  - 分类
    - [abc] 表示匹配 abc中任意一个字符
    - [abc][ac] 表示匹配两个字符 第一个字符表示abc中任意一个 第二个字符表示ac中任意一个
    - [^abc] 表示不匹配abc中的任意一个字符
    - [0-9] 表示匹配0-9中的任意一个数字
    - [^0-5] 表示不匹配0-5中的任意一个数
    - [a-z] 表示匹配一个小写字母
    - [A-Z] 表示匹配一个大写字母
    - [A-z] 表示匹配一个大小写字母
    - [A-Za-z] 表示匹配一个大小写字母
    - [A-z0-9_] 表示匹配一个数字字母下划线
- 3.预定义字符集
  - 预先设置好的字符集 字符集表示的是一个字符
  - 分类
    - \d 表示匹配一个数字 等同于[0-9]
    - \D 表示匹配一个非数字 等同于[^0-9]
    - \w 表示匹配字母 数字 下划线 等同于[a-z0-9_]
    - \W 表示匹配 非字母 数字 下划线 等同于[^A-z0-9_]
    - \s 表示匹配 空白符 (空格 制表符)
    - \S 表示匹配 非空白符
    - . 表示匹配 除回车换行(\r\n)外的 任意字符
- 4.匹配数量
  - 匹配字符的次数
  - 分类
    - n? 表示匹配0次或1次字符
    - n* 表示匹配0次或多次字符
    - n + 表示匹配1次或多次字符
    - n{x} 表示匹配x次字符
    - n{x,y} 表示匹配x至y次字符
    - n{x,} 表示匹配至少x次字符
    - ```
      var reg = /abc{2}/;//字符串可以匹配abccc等多次
      var reg = /abc*/;//字符串可以匹配abc等多次
      var reg = /abc+/;//字符串一定要出现abc
      var reg = /abc{2,5}/;//字符串可以匹配abccccc
      var reg = /abc{2,}/;//字符串可以匹配abccccccccccc
      ```
- 5.匹配边界
  - 分类
    - ^ 匹配开头
    - $ 匹配结尾
  - ```
    var reg = /^1[3-9]\d{9}$/;
    ```
- 6.选择分组
  - 分类
    - | 在正则中表示选择 逻辑或 相当于if中的||
    - () 表示分组 一般情况 一个小括号被看做一个整体
  - 一般将 | 和 () 搭配使用
  - ```
    var reg = /a((abc)|(123)|(zzz))/;
    ```
- 7.字符串与正则相关API
  - String.prototype.match();
    - 语法:str.match(regexp);
    - 参数:regexp 正则对象
    - 描述:在字符串中查找与正则所匹配的文本
    - 返回值
      - 数组，内容是正则所匹配到的文本内容
      - 如果没有匹配到任何内容 则返回 null
  - 正则表达式属性
    - global 修饰符 g 全局匹配
    - ignoreCase 修饰符 i 忽略大小写
  - 正则表达式匹配中文 [\u4e00-\u9fa5]
  - String.prototype.search();
    - 语法: str.search(regexp);
    - 描述: 在字符串中通过正则表达式 匹配文本内容
      找到文本在字符串中所对应的索引值
    - 返回值
      - number 索引值
      - 如果没有匹配到结果 则返回 -1
  - String.prototype.replace(regexp,string);
    - 应用 计算字符串的长度
    - 一个双字节的字符串长度为2 ASCII字符长度1
    - [^\x00-\xff] 匹配双字节字符(包括所有汉字)
    - ```
      String.prototype.len = function() {
          return this.replace(/[^\x00-\xff]/g, 'aa').length;
      }//首先将所有的双自己字符，替换成aa的两个字符，然后再计算替换后的字符的长度即可（如果直接使用长度，双字节字符的长度是1
      var str = 'ss1我是aa跟SS界';
      console.log(str.len());
      ```
- 8.正则API
  - Regexp.prototype.exec();
    - 语法: reg.exec(string);
    - 参数: string 字符串
    - 描述: 在字符串中执行一个搜所匹配
    - 返回结果
      - 返回一个结果数组
      - null
    - 如果一个字符串使用多个exec()函数，那么就会从上次索引结束的地方开始检索
    - ```
      var str = 'az1bc112de33';
      var reg = /\d+/g;

      var result = reg.exec(str);
      console.log(result);//21
      result = reg.exec(str);
      console.log(result);//123
      result = reg.exec(str);
      console.log(result);//33
      result = reg.exec(str);
      console.log(result);//null
      ```
    - while补充
      - 当在while和do-while循环条件中放入赋值语句时 循环条件是就是赋值表达式的结果
      - ```
        while(result = reg.exec(str)){//^reg.exec(str)作为条件 循环判断 ，为真则 执行循环体
            arr.push(result[0]);
        }

        console.log(arr);
        ```
- 9.去掉前后空格
  - ```
    // 编写一个trim函数
    // 参数 一个string 去除字符串前后空格
    window.onload = function() {
        var username = document.querySelector('#username');
        // ECMAScript5 已经为string 添加了 trim 方法

        if (!String.prototype.trim) { // 没有trim函数 就创建
            //
            String.prototype.trim = function() {
                return this.replace(/^\s+|\s+$/g, "");
                //^匹配开头 一或多一个空格 |或 $匹配结尾 一或多一个空格
                //匹配一个以上的空格 并以空格结束的的地方
                //替换为空 即空字符串替换
            }
        }

        username.onchange = function() {
            console.log(this.value.trim());
        }
    }
    ```
- 10.文本采集
  - ```
    // 防止 XSS 攻击（跨站脚本攻击）
    // 1. 替换
    // 2. 加密/转义 < >
    window.onload = function() {
        var input = document.
        getElementById('input');
        var output = document.
        getElementById('output');
        var submit = document.
        getElementById('submit');

        submit.onclick = function() {
            // 
            output.value = input.value.
            replace(/</g,'<');
            //[]内的/表示单，{*x} 表示匹配除了 x 以外的字符 这个例子表示 这样写，就能匹配所有的标签了
        }
    }
    ```
- 11.密码强度
  - ```
    <script>
        // 1.字符 大小写 数字 特殊符号
        // 密码长度小于 8 字符串
        // 密码长度大于等于8字符串数据
        window.onload = function() {
            var password = document.querySelector('#password');
            var result = document.querySelector('#result');

            // 初始时间
            // 1.小写字母 大写字母 数字 特殊符号
            // 密码长度 ≤,≥
            // 密码字符数小于等于 8个 字
            //                    8个 中
            //                    4个 英

            var passReg = [
                /^.{0,8}$|^.{9,}$/,// 密码长度
                /[A-Z]+/,// 至少大写字母
                /[a-z]+/,// 至少小写字母
                /[0-9]+/,// 至少数字
                /[^A-za-0-9]+/// 至少特殊符号
            ];

    if (check[0]) {
        switch (strength.count) {
            case 3:
                result.innerHTML = '弱';
                break;
            case 4:
                result.innerHTML = '中';
                break;
            case 5:
                result.innerHTML = '强';
                break;
        }
    } else {
        result.innerHTML = '密码长度不足8位';
    }
    ```
    - ```
      password.oninput = function() {
          var that = this;
          var check = passReg.map(function(ele) {
              // 遍历数组
              // 使用数组种的每一个正则 验证输入框的字符串
              // 将所有验证的结果 返回成一个数组 数组种存放了 5个布尔值
              return ele.test(that.value);
          });

          var strength = check.reduce(function(obj,
          current) {//判断满足几种条件
              current && obj.count++;
              return obj;
          }, {
              count: 0
          });
      ```
- 12.自定义属性
  - 来自HTML5标准 为元素 添加自定义的属性
  - data- 进行命名
  - JS添加了自定义属性的 访问接口 dataset
  - ```
    <style>
        #box::after {
            display: inline;
            content: attr(data-name);
            /*content只支持字符串*/
            /*attr(data-name) 表示将获取id为box的data-name的值*/
            /*然后将此content显示在box的伪元素下，最后以文本的形式在页面中显示*/
        }
    </style>
    <script>
        // 在HTML5标准 为元素 添加自定义的属性
        // data- 进行命名

        // 在JS添加了自定义属性的 访问接口 dataset
        window.onload = function() {
            var box = document.getElementById('box');

            console.log(box.dataset.name);

            box.onclick = function() {
                this.dataset.name = "hello world";
            }
        }
    ```
    - ```
      <body>
          Lorem ipsum dolor sit amet consectetur adipisicing elit. Doloribus totum illo ipsa aut voluptas cupiditate reiiciendis at facere, nihil autem sit rem at pariatur quis harum accusamus itaque labore eligendi.
      </body>
      ```
      ```