正则表达式

创建正则表达式

- 字面量定义
  - 双斜杠包裹

```
let hd = 'hdsufgaiusdhuf123123sdnjfak';
console.log(/u/.test(hd));//true 检测是不是有‘u’这个字符
```

- 使用对象创建正则表达式

```
let reg = new RegExp('u','g');//1.要匹配的字符 2.匹配的模式
console.log(reg.test(hd));//true
```

  - 可以使用变量来代替要匹配的字符

```
let hd = 'hdsufgaiusdhuf123123sdnjfak';
let a = 'abc';
let reg = new RegExp(a,'g');
console.log(reg.test(hd));//在hd中匹配‘abc’
```

选择符

- 匹配有其中一个即为真

```
let hd = 'hdsufgaiusdh@@uf123123sdnjfak';
console.log(/a|@/.test(hd));//true hd中有a或者@都会返回
```

- 原子表与原子组中的选择符

```
//原子表
let reg = /[123456]/;//匹配1或2或3或。。。
let hd = '3';
console.log(hd.match(reg));

//原子组
let reg1 = /(12|34)/;//匹配12或34
let hd1 = '12';
console.log(hd1.match(reg1));
```

转义

```
//转义
    let price = 23.34;
    // . 的含义 除换行外任何字符 因此要转义变成小数点含义
    // d 的含义 \d 表示数值(0-9)
    // + 表示要匹配一个或多个这种类型的值
    console.log(/\d+.\d/.test(price));//true 但不正确 因为 . 在这里的意思是除换行外任何字符
    console.log(/\d+\.\d/.test(price));//true 依然为真 但此时 . 经过转义，此时含义为小数点
    //在对象里面又不一样了
    let reg = new RegExp('\d+\.\d+');
    console.log(reg.test(price));//false
    //以上原因是因为 在字符串里面 '/d' == 'd';是为真的 因为上一个正则规则表示匹配一个或多个d中间放任何字符
    console.log(reg.test('dddddd@dddd'));//true
    //因此为了让正则表达式reg正确 需要再转义（再加斜杠）
    let reg1 = new RegExp('\\d+\\.\\d+');
    console.log(reg1.test(price));//true
```

字符边界约束

```
  //字符边界约束
    let hd = 'addsfjsdh3sdfus'
    let hd1 = '3'
    console.log(/\d/.test(hd));//true 只要包含一个数字就为真
    console.log(/^\d/.test(hd));//false 开头包含数字就为真
    console.log(/^\d$/.test(hd));//false 起始和结尾包含数字就为真
    console.log(/^\d$/.test(hd1));//true 起始和结尾包含数字就为真
```

- 例子

```
//html部分
<input type="text" name="user">
//js
    document
    .querySelector('[name="user"]')
    .addEventListener('keyup',function(){
        // console.log(this.value.match(/[a-z]{3,6}/));
        //字母a-z 数量3-6位 如果不满足则输出null 但是你输入7位还是能匹配 因为不严谨啊
        console.log(this.value.match(/^[a-z]{3,6}$/));//加上起始和结束符号就更严谨了
    });
```

元字符

```
    //元字符
    let hd = 'loisxu2020'
    console.log(hd.match(/\d/g));//数组 2 0 2 0
    console.log(hd.match(/\d+/g));//字符串 ‘2020’

    let hd1 = `
    张三：010-999999，李四：020-888888
    `;
    console.log(hd1.match(/\d{3}-\d{5,6}/g));// ["010-999999", "020-888888"]
    // \d是数字 \D是非数字

    console.log(hd1.match(/[^-\d:,]+/g));//匹配中文 中括号中是原子表（？） 在前面加^表示只匹配除了^后的东西
    //["↵    张三：", "，李四：", "↵    "]

    // \s 空白 表示空格换行制表符等 \S 非空白

    console.log(/\s/.test('  loi s'));//true

    console.log(hd1.match(/[^-\d： ，\s]+/g));//["张三", "李四"]
```

```
 //\w元字符 字母数字下划线
    let foo = 'lois2020-'
    console.log(foo.match(/\w+/))//lois2020 并不能匹配到‘-’

    let mail = '1965641462@qq.com'
    console.log(mail.match(/^\w+@\w+\.\w+$/));//这样就能匹配整个邮箱了

    //\W 除了字母数字下划线
    console.log('@xuxuux'.match(/\W/));//@

    //点元字符 . 包含除了换行外的任何
    let ss = 'sdfjashdk/f,f'
    console.log(ss.match(/.+/));//能够匹配变量中的所有字符

    //匹配所有
    let html = `
    <sapn>
        lois @@@@
        dfhsjsbhd
    <span/>
    `
    console.log(html.match(/<span>[\s\S]+<span\/>/));
```