## 周一、周二

只看英语，并且做相关的笔记记录

## 周三、周四

只看相关的文章以及动手写代码，做练习

## 周五

做英语以及代码相关的整理、总结

## 周六以及周日

1. 上午九点半之前必须起床收拾好；
2. 上午十点到十二点学习英语；
3. 中午十二点到两点之前吃饭以及午休时间；
4. 下午两点到六点看教程或写代码或继续学习英语；
5. 下午六点到八点吃饭休息时间；
6. 八点之后视情况，安排其他事情；

# 阶段一、Javascript

## Javascript基础篇

1. 数据类型与类型转换
2. 转换为数值类型：

Number(value)：

1. 若是布尔值，true和false分别转换为1和0
2. 若是数值则返回自身
3. 若是null返回0
4. 若是undefined则返回NaN
5. 若是字符串，则遵循:

>如果字符串是有效的数字，则将其转换为十进制（忽略前导0）

>如果字符串是有效的浮点格式，则将其转换位浮点数值（忽略前导0）

>如果是空字符串则将其转换为0

>如果字符串非以上格式，则将其转换为NaN

>如果是对象则先调用valueOf()方法，然后依据上述规则转换返回的值。如果转换的结果是NaN，则调用对象的toString()方法，再次依照上述规则转换返回的字符串值；

parseInt(value, radix)：

1. 忽略字符串前面的空格，直至找到第一个非空字符；
2. 如果第一个字符不是数字或负号，则返回NaN；
3. 如果第一个字符是数字，则继续解析直至字符串解析完毕或者遇到一个非数字符号为止；
4. 如果上步解析的结果以0开头，则将其当作八进制来解析，如果以x开头，则将其当作十六进制来解析；
5. 如果指定radix参数，则以radix为基数进行解析

parseFloat(value, radix)：

1. 字符串中第一个小数点符号是有效的，另外会忽略前导0，如果字符串包含一个可解析为整数的数，则返回整数值而不是浮点数
2. 其余规则同parseInt();

进制转换：

1. 八进制转十进制：按权相加法，72.45整数部分从右往左分别乘以8的0次方加上8的1次方…，小数部分从左往右分别乘以8的-1次方加上8的-2次方…，然后整数部分加上小数部分
2. 二进制转十进制：按权相加法，1110.1101整数部分从右往左分别乘以2的0次方加上2的1次方…，小数部分从左往右分别乘以2的-1次方加上2的-2次方…，然后整数部分加上小数部分
3. 十六进制转十进制：按权相加法，6a.3b整数部分从右往左分别乘以16的0次方加上16的1次方…，小数部分从左往右分别乘以16的-1次方加上16的-2次方…，然后整数部分加上小数部分
4. 十进制转八进制：直接法，整数部分除8取余，直至商为0，小数部分乘8取整，直至小数部分为0
5. 十进制转二进制：直接法，整数部分除2取余，直至商为0，小数部分乘2取整，直至小数部分为0
6. 十进制转十六进制：直接法，整数部分除16取余，直至商为0，小数部分乘16取整，直至小数部分为0
7. 二进制转八进制&&八进制转二进制：二进制的数从左往右，每三位划分一组，不够位的补0转换为，然后根据对照表进行转换

|  |  |
| --- | --- |
| 二进制 | 八进制 |
| 000 | 0 |
| 001 | 1 |
| 010 | 2 |
| 011 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | 7 |

1. 二进制或八进制转十六进制：先转换为十进制或二进制，在转换位十六进制

|  |  |
| --- | --- |
| 二进制 | 十六进制 |
| 0000 | 0 |
| 0001 | 1 |
| 0010 | 2 |
| 0011 | 3 |
| 0100 | 4 |
| 0101 | 5 |
| 0110 | 6 |
| 0111 | 7 |
| 1000 | 8 |
| 1001 | 9 |
| 1010 | A |
| 1011 | B |
| 1100 | C |
| 1101 | D |
| 1110 | E |
| 1111 | F |

（2）转换为字符串类型：

toString(radix):

1、除undefined和null之外的所有类型的值都具有toString()方法，其作用是返回对象的字符串表示

|  |  |
| --- | --- |
| 对象 | 返回值 |
| Array | 将Array元素转换为字符串，结果字符串由逗号分隔字符串 |
| Boolean | “true”或”false” |
| Date | 日期的文字表示法，"Tue May 14 2019 00:09:15 GMT+0800 (中国标准时间)" |
| Error | 返回一个包含错误信息的字符串，"Error: dads" |
| Function | 返回"function() {var a = 234;}"函数字符串表示 |
| Nmber | 数字的字符串表示 |
| String | 返回字符串String对象的值 |
| Object | 返回"[object Object]" |
| RegExp | 参数为空时返回"/(?:)/"，有值时返回匹配规则 |

String(mix)函数

1. 如果有toString()方法，则调用该方法并返回结果；
2. 如果是null，返回’null’
3. 如果是undefined,返回’undefined’

（3）转换为布尔值：

1、这些值会被转换为false：false、空字符、0、NaN、null、undefined，其它值都会被转换为true

（4）隐式转换

1、isNaN(value)全局方法会在内部调用Number(value)方法处理一下传入值，然后进行恒等操作；Number.isNaN(value),直接进行恒等比较传入的value，不会做任何的处理；

2、递增递减操作符--、++（包括前置和后置）、一元正负符号操作符-、+：

（1）如果是一个有效的数字字符串，先转换为数值在进行加减的操作

（2）如果是一个非有效的数字字符串，则转换为NaN

（3）如果是布尔值则转换成0或者1

（4）如果是一个对象，则先调用valueOf()方法执行前面的规则，如果结果是NaN，则再调用toString()方法在执行前面的规则，否则是NaN;

3、加法运算操作符，也用于字符串连接符，先将操作值调用toString()转换为字符串基础数据类型，然后有两种情况：

（1）若果两个操作数都是数值时：

>如果有一个为NaN则结果为NaN

>如果是Infinity + Infinity, 则结果为Infinity

>如果是-Infinity + (-Infinity),则结果为-Infinity

>如果是Infinity + (-Infinity),则结果为NaN

>如果是+0 + (+0),则结果为0

>如果是-0 + (-0), 则结果为-0

>如果是+0 + (-0),则结果为0

（2）如果一个值为字符串：

>一个值为字符串时，将另一个值转换为字符串然后拼接起来

>如果操作值为对象、数组或者布尔值，则调用toString()方法转换为字符串，然后拼接起来

>如果第一个值为数值，另一个值为对象、数组或者布尔值，则将数值的转换为字符串，非基础数据类型值调用toString()方法转换为字符串，然后拼接

>如果是null或undefined，则分别显式调用String()转换为字符串，然后拼接起来

>基础数据类型相加时，直接将其转换为数值，然后相加

>Symbol类型的由于内部没有相应的计算机制，所以直接报错

4、乘除、减号运算符、取模运算符：通过Boolean()函数将它的操作值转换为布尔值

5、逻辑运算符（&&、||、!）、逻辑非操作符

（1）&&运算中，第一个值转换为布尔值为false时则返回第一个值，为true时则返回第二个值

（2）||运算中，第一个值转换为布尔值为false时则返回第二个值，为true时则返回第一个值

（3）!运算中，通过Boolean()转换为布尔值，然后求反

6、关系操作符（<、>、<=、>=）

（1）如果两个操作值都是字符串，则比较对字符串对应的字符编码值

（2）如果只有一个操作值是数值，则将另一个操作值转换为数值，然后进行比较

（3）如果一个操作数是对象时，调用valueOf()方法，如果没有valueOf()则调用toString()方法，然后按照上述规则进行比较

（4）如果一个操作值是布尔值时，转换为数值进行比较，NaN是一个特殊的数值，它与任何值比较都返回false，且不等于任何值包括自身

7、相等操作符（==）会对操作值进行隐式转换后进行比较

（1）如果一个操作值为布尔值，则将其转换为数值在进行比较

（2）如果一个操作值为字符串，另一个操作值为数值时，调用Number()将字符串转换为数值在进行比较；

（3）如果一个操作值为对象时，先调用valueOf()方法，得到结果后再根据上述规则进行比较；

（4）null == undefined 返回true

（5）如果一个操作值为NaN，则返回false

（6）如果两个操作值都是对象，则比较两个对象是否指向同一个对象

1. This指向与修改指向
2. ES5中this实在运行时绑定到调用函数的对象上
3. ES5中严格模式下若不使用window调用函数，函数的this指向undefined,使用时指向window
4. ES5中严格模式下，在定时器setTimeout、setInterval中this都指向undefined,非严格模式下指向window，这是因为定时器是运行在全局环境下的
5. ES6中的箭头函数this的指向与定义时环境中this的指向一致
6. DOM事件中this指向触发该事件的DOM对象
7. 构造函数中的this指向通过构造函数所创建出的对象实例
8. 可以通过bind()、call()、apply()、以及在函数内定义变量接收this来改变this的指向
9. 在使用bind()、call()、apply()改变this指向的时候，如果传入的不是一个对象， 则会调用相对应的构造函数，进行隐式类型转换
10. 正则对象：正则表达式是匹配模式，要么匹配字符，要么匹配位置
11. 横向模糊匹配：可匹配的字符串长度是不固定的
12. 纵向模糊匹配：匹配的字符，具体到某一位字符时，它可以不是某个固定的字符
13. 字符组：即匹配一组字符中的任意一个字符，如：[a-z0-9A-Z],表示小写字母a-z、数字0-9、大写字母A-Z之中的任何一个；
14. 排除字符组：即某位字符可以是任何字符，但就不能时’a’、’b’、’c’，如[^abc]，则表示排除a、b、c这三个字符，^（脱字符），表示求反的概念。范围排除表示法：[^a-z0-9A-Z]
15. 常见的简写形式：如果要匹配任意字符则可以使用[\d\D]、[\w\W]、[\s\S]、[^]其中的一种方式

|  |  |
| --- | --- |
| 简写形式 | 表示的内容 |
| \d | [0-9]，表示匹配0-9中的任意一个数字 |
| \D | [^0-9]，表示匹配除0-9之外的任意一个字符 |
| \w | [0-9a-zA-Z\_]，表示匹配0-9、a-Z、a-z以及下划线(\_)中的任意一个字符 |
| \W | [^0-9a-zA-Z\_]，表示匹配非单词、数字以及下划线的字符 |
| \s | [\t\v\n\r\f]，表示匹配空白符，包括：空格()、水平制表符(\t)、垂直制表符(\v)、换行符(\n)、回车符(\r)、换页符(\f) |
| \S | [^\t\v\n\r\f]，表示匹配非空白符 |
| .(点符号) | [^\n\r\u2028\u2029]，通配符，表示匹配几乎任意字符，注意：换行符、回车符、行分隔符(\b)、段分隔符除外 |

1. 量词：也称重复，即{m,n}形式，以下是一些常见简写形式

|  |  |
| --- | --- |
| 简写形式 | 表示内容 |
| {m,} | 表示至少出现m次 |
| {m} | 等价于{m,m}，表示出现m次 |
| ? | 等价于{0,1},表示出现或者不出现 |
| + | 等价于{1,}，表示出现至少一次 |
| \* | 等价于{0,},表示出现任意次，有可能不出现 |

1. 贪婪匹配：尽可能多的匹配，在能力范围内越多越好。如：

reg = /\d{2,5}/g str = ‘123 1234 12345 123456’

str.match(reg) => [‘123’, ‘1234’, ‘12345’, ‘12345’]

1. 惰性匹配：尽可能少的匹配，在量词后面加上?即可实现惰性匹配。如：

reg = /\d{2,5}?/g str = ‘123 1234 12345 123456’

str.match(reg) => [‘12’, ‘12’, ‘34’, ‘12’, ‘34’, ‘12’, ‘34’, ‘56’]

1. 多选分支：一个模式可以实现横向和纵向模糊匹配，而多选分支可以支持多个子模式任选其一，且是惰性匹配，当前面的模式匹配了则不再尝试后面的模式，形式如：(p1|p2|p3)，即使用|（管道符）分隔，表示其中任何一个
2. 有限自动机：分为确定性有限自动机（DFA）和非确定性有限自动机（NFA）

（1确定性有限自动机：

>以文本为主导

>编译时确定所有的匹配可能性，运行速度快

>只有一次匹配机会，如果失败则结束匹配

（2非确定性有限自动机：

>以正则表达式为主导

>编译时无法确定所有匹配可能性，只有在运行时逐个尝试所有的可能性，所以编译速度快

>表达能力更强，开发者控制度高

>如果匹配失败，则返回至起始点重新开始尝试新的可能性

（3符合标准的正则引擎叫做POSIX NFA引擎，其余的叫做传统型NFA引擎。Javascript使用的是传统型NFA引擎

1. 回溯：由于NFA引擎是用表达式去匹配文本，而表达式又可能会有若干分支和范围，一个分支或范围匹配失败并不意味着最终匹配失败，引擎会尝试下一个分支或者范围

（1避免回溯方法：

>减少分支

>缩减文本长度

>控制备选状态的数量以及顺序

（2产生回溯的原因：

>贪婪量词

>惰性量词

>分支结构

（12）捕获组：即圆括号包裹的正则表达式，若有嵌套的捕获组，则先由外而内递归，递归完毕后进入下一个顶级捕获组，即深度优先

（1正则内捕获：即使用”\数字”的形式，也叫做反向引用

（2正则外捕获：即使用”RegExp.$数字”的形式访问对应的捕获组，若有多个正表达式时，只显示最后一个正则的捕获组

（3捕获命名：在捕获组内部最前面即圆括号内最左侧使用”?<name>”来命名捕获组，使用”\k<name>”来进行已命名捕获组的引用

1. 非捕获组：在捕获组内部最前面即圆括号内最左侧加上”?:”即可，此时不会有引用创建，只是作为一个匹配组使用其他地方无法使用此捕获组，性能更好
2. 零宽断言：匹配一个位置，本身没有宽度，不影响后续的正则表达式匹配

（1断言：是一种判断，断言之前或之后应该有什么或应该没有什么

（2零宽肯定先行断言：（?=），在圆括号内最左边加上?=标识

>肯定：即判断有什么，而不是判断没有什么

>先行：即指的是向前看（lookahead），断言的这个位置是为前面的规则服务的

注：即要匹配一段文本，但是这段文本后面必须紧跟着另一段特定的文本，零宽肯定先行断言就是一个界碑，即要满足前面和后面所有的文本，但是我只要前面的文本

例子1：’CoffeeScript JavaScript javascript’.match(/\b\w{4}(?=Script\b)/);

// Java

例子2：’ CoffeeScript JavaScript javascript’.match(/\b\w{4}(?=Script\b)\w+);

// JavaScript

如上例：零宽肯定先行断言已经匹配过Script一次了，后面的\w+却依然可以匹配Script成功，足以说明它的零宽特性，它仅为紧贴在它前面的规则服务，并且不影响后面的匹配规则

（3零宽肯定后行断言：（?<=），在圆括号内最左边加上?<=标识

>后行：指的是向后看，断言的这个位置是为后面的规则服务的

例子：'CoffeeScript JavaScript javascript'.match(/\b\w+(?<=Coffee)\w+\b/);

// CoffeeScript

（4零宽否定先行断言：（?!），在圆括号内最左边加上?!标识

>否定：指的是判断没有什么

（5零宽否定后行断言：（?<!），在圆括号内最左边加上?<!标识

1. 修饰符：

（1修饰符g：开启全局匹配模式，找到所有匹配的结果

（2修饰符i：开启全局忽略大小写

（3修饰符y：与g修饰符有重合的功能，都是全局匹配，但是y修饰符要求必须从文本的开始实施匹配，匹配到的文本的下一个字符就是下一次文本的开始，这就是所谓的粘连。

>本质： 1、全局匹配

2、从文本的lastIndex位置开始新的匹配，lastIndex是正则表达式的一个属性

3、每执行一次则返回一个结果

>注：match()方法没有lastIndex属性，所以要配合g修饰符使用才能一次性返回所有的结果，使用正则表达式的方法才会没执行一次返回一个结果

（4修饰符s：要和点符号（.）搭配使用，可以让点（.）操作符匹配换行符，即singleline得缩写

（5修饰符U：即Unicode得缩写，正常情况下Unicode字符超过一个字节正则就无法识别，U修饰符就是用来处理这些情况的，且之匹配最近的一个字符，不重复匹配

（6修饰符m：开启多行文本匹配模式，搭配^和$符，即为行的开始和结束

（15）锚字符：即位置，^开始位置、$结束位置、\b单词边界、\B非单词边界、\(?=p)零宽肯定先行断言、(?!p)零宽否定先行断言、(?<=p)零宽肯定后行断言、(?<!p)零宽否定

后行断言

（1 \b：即单词边界，具体就是\w和 \W之间的位置，也包括\w和^、\w和$之间的位置，即除[0-9a-zA-Z\_]之外的都是单词边界

（2对于位置的理解，可以堪称是空字符串’’：‘Hello’ = ‘’+’H’+’’+’e’+’’+’l’+’’+’l’+’’+’o’+’’

案例：

>不匹配任何字符：/.^/

>数字的千位分隔符表示法：/(?!^)(?=\d{3})+$/g

>包含某种字符：/(?=.\*[0-9])^[a-z0-9A-Z]{6,12}$/

（16）操作符优先级：转义符(\) > 圆括号、中括号> 量词限定符> 位置和序列、’^’、 ‘$’、’\元字符’、一般字符> 管道符(|)

（17）如何提高匹配的效率：

（1 使用具体型字符组来代替通配符

（2 使用非捕获分组

（3 独立出确定的字符，提取分支的公共部分

（4减少分支数量，缩小它们的范围

1. 时间对象
2. 通过setTime(millisec)方法，参数millisec为毫秒数，如果为负数则表示1970年1月1日之前的时间
3. 通过setFullYear(year,month,day)方法可以设置时间对象的年月日，也可以求出某个月有多少天，即参数day的值为0时表示（month - 1）月的最后一天，为-1时则是（month - 1）月的最后一天的前一天，如果month有31天，day值为32时则是（month + 1）月的第一天
4. GMY(格林尼治平时)：又称格林尼治标准时间，由于地球每天的自转有些不规则，并且正在缓慢减速，因为格林尼治时间已经不再被作为标准时间使用，现在的标准时间改为由原子钟报时的协调世界时（UTC）
5. UTC(世界标准时间)：又称协调世界时或世界协调时间
6. CST(北京时间)：中国标准时间，在时区划分上，属东八区，比协调世界时早8小时，记为UTC+8
7. Math对象
8. Math的属性：

|  |  |  |
| --- | --- | --- |
| 属性名 | 描述 | 值 |
| Math.E | 欧拉常数，也是自然对数的底数 | 约为2.718 |
| Math.LN2 | 2的自然对数 | 约为0.693 |
| Math.LN10 | 10的自然对数 | 约为2.303 |
| Math.LOG2E | 以2为底E的对数 | 约为1.443 |
| Math.LOG10E | 以10为底E的对数 | 约为0.434 |
| Math.PI | 圆周率 | 约为3.14 |
| Math.SQRT1/2 | 1/2的平方根 | 约为0.707 |
| Math.SQRT2 | 2的平方根 | 约为1.414 |

1. Math的方法：分为三类三角函数、数学运算方法、数值运算方法

>三角函数：

|  |  |
| --- | --- |
| 方法名 | 描述 |
| Math.sin(x) | 返回x正弦值 |
| Math.sinh(x)ES6新增 | 返回x双曲正弦值 |
| Math.cos(x) | 返回x的余弦值 |
| Math.cosh(x)ES6新增 | 返回x的双曲余弦值 |
| Math.tan(x) | 返回x的正切值 |
| Math.tanh(x)ES6新增 | 返回x的双曲正切值 |
| Math.asin(x) | 返回x的反正弦值 |
| Math.asinh(x)ES6新增 | 返回x的反双曲正弦值 |
| Math.acos(x) | 返回x的反余弦值 |
| Math.atan(x) | 返回x的反正切值 |
| Math.atan2(x, y) | 返回x/y的反正切值 |
| Math.atanh(x)ES6新增 | 返回x的反双曲正切值 |

>数学运算方法

|  |  |
| --- | --- |
| 方法名 | 描述 |
| Math.sqrt(x) | 返回x的平方根 |
| Math.exp(x) | 返回欧拉常数e的x次幂的值 |
| Math.pow(x,y) | 返回x的y次幂，若y未传，则返回NaN |
| Math.expm1(x)ES6新增 | 返回拉常数e的x次幂减去1的值 |
| Math.log(x) | 返回x的自然对数 |
| Math.log1p(x)ES6新增 | 返回x+1后的自然对数 |
| Math.log2(x) ES6新增 | 返回x以2为底的对数 |
| Math.log10(x) ES6新增 | 返回x以10为底的对数 |
| Math.cbrt(x) ES6新增 | 返回x的立方根 |
| Math.clz32(x) ES6新增 | 返回一个数字x再转换成32位无符号整数型数字的二进制形式后，开头的零的个数 |
| Math.hypot(x,y,z) ES6新增 | 返回所有参数的平方和的平方根 |
| Math.imul(x,y) ES6新增 | 返回两个参数的类C的32位整数乘法运算的运算结果 |

>数值运算方法

|  |  |
| --- | --- |
| 方法名 | 描述 |
| Math.abs(x) | 返回x的绝对值 |
| Math.floor(x) | 向下取整 |
| Math.ceil(x) | 向上取整 |
| Math.trunc(x) ES6新增 | 返回x的整数部分 |
| Math.fround(x) ES6新增 | 返回离x最近的单精度浮点数形式的数字 |
| Math.min(x,y,z) | 返回多个数中的最小值 |
| Math.max(x,y,z) | 返回多个数中的最大值 |
| Math.round(x) | 返回四舍五入后的整数 |
| Math.random() | 返回0到1之间的随机数 |
| Math.sign(x) ES6新增 | 返回一个数的符号（5种返回值，分别是1、-1、0、-0、NaN，代表的各是正数、负数、正零、负零、NaN） |

1. Number.prototype的toFixed()方法用于将数字转换为指定小数位数的形式；

>在没有传参或者参数为0时，会返回该值四舍五入后的整数形式，等同于Math.round(x)

>其他情况下返回该值指定小数位数的四舍五入后的结果

（4） 位运算：是一种直接进行二进制运算的运算方式

>负数转二进制时采用补码表示，即符号位不变，其他位取反后+1

例：-x的二进制=-( x的二进制取反+1)

>按位与（&）：用于连接两个数，连接的两个数它们的二进制补码形式的值每位都将参与运算，只有相对应位上的值都为1，才会返回1，是一种趋向减小最大值的运算，且满足如下规律：

（1、数值与自身（或者-1）按位与运算返回数值本身

（2、2的整数次方与它的相对数按位与运算返回它自身

（3、任意整数与0进行按位与运算都将返回0

（4、任意整数与1进行按位与运算，奇数时返回1，偶数时返回0

（5、按位与运算的结果不大于两个数中的最大值

>按位或（|）：不同于&运算，只要其二进制补码各位上有一个为1，则该位的原酸就返回1，是一种趋向增大最小值的运算，且满足如下规律：

（1、数值与自身按位或运算返回自身

（2、2的整数次方的值与它的相对数按位或运算返回它的相对数

（3、任意整数与0进行按位或运算，都将会返回它本身

（4、任意整数与-1进行按位或运算，都将会返回-1

（5、按位或运算的结果不小两个数中的最小值

>按位非（~）：返回数值二进制补码形式的反码，即数值二进制补码形式中的每一位都将取反，若该位为1取反为0，该位为0取反为1，且满足如下规律：

（1、按位非操作一个数值，等同于这个数值+1，然后符号改变。即：~x=-(x+1)

（2、~NaN返回-1

>按位异或（^）： 连接的两个数的二进制补码形式每位的值不同返回1，否则返回0，且满足如下规律：

（1、由于按位异或运算符的特殊性，数值与自身按位异或运算返回0

（2、任意整数与0进行按位异或运算，都将会返回它本身

（3、任意整数x与1（2的0次方）进行按位异或运算，若它为奇数则返回x-1，若为偶数则返回x+1,同理，任意整数x与2的n次方进行按位异或运算，若他的二进制补码形式的第n+1位是1，则返回x-2的n次方，反之若为0则返回x+2的n次方

（4、任意整数x与-1（负2的1次方+1）进行按位异或运算，返回-x-1,相当于~x运算

（5、任意整数连续按位异或两次相同的数值，返回它本身

（6、按位异或满足操作数与运算结果3个数值之间的交换律：按位异或的两个数值，以及它们运算的结果，共三个数值可以两两异或得到另外一个数值

>有符号左移（<<）：表示将数值的32位二进制补码形式的除符号之外的其他位都往左移动若干位数，当x为整数时，有：x<<n === x\*Math.pow(2,n)。

如：100<<4 === 1600,即Math.pow(2,n)可简写为1<<n

1. 对于表达式x<<n，当运算数x无法被转换为整数时，运算结果为0，即x为NaN时；
2. 对于表达式x<<n，当运算数n无法被转换为整数时，运算结果为x,即相当于x<<0
3. 当运算数x和n均无法被转换为整数时，运算结果为0

>有符号右移（>>）：除了符号方向向右，其它和有符号左移一致，当x为整数时，有x>>n === Math.floor(x\*Math.pow(2,-n))

（1、右移负整数时，返回值最大为-1

（2、右移正整数时，返回值最小为0

>无符号右移（>>>）：连同符号也一起右移

（1、无符号右移会把负数的二进制码当成整数的

（2、使用无符号右移，右移0位，对于负数而言也是翻天覆地的变化，但是对于正数却没有变化，由此可以判断一个数字是正数还是负数

>运算符优先级：

|  |  |  |
| --- | --- | --- |
| 优先级 | 运算符 | 描述 |
| 1 | 后置++、后置--、[]、()、点符号（.） | 后置++、后置--、数组下标、圆括号、属性选择 |
| 2 | -、前置++、前置--、!、~ | 负号、前置++、前置--、逻辑非、按位非~ |
| 3 | \*、/、% | 乘、除、取模 |
| 4 | +、- | 加、减 |
| 5 | <<、>> | 有符号按位左移、有符号按位右移 |
| 6 | >、>=、<、<= | 大于号、大于等于号、小于号、小于等于号 |
| 7 | ==、!= | 值相等、值不相等 |
| 8 | & | 按位与 |
| 9 | ^ | 按位异或 |
| 10 | | | 按位或 |
| 11 | && | 逻辑与 |
| 12 | || | 逻辑或 |
| 13 | …?...:… | 三元条件运算符 |
| 14 | =、/=、\*=、%=、+=、-=、<<=、>>=、&=、^=、|=后赋值 | 各种运算后的赋值 |
| 15 | , | 逗号 |

补码：在计算机系统中，数值一律用补码来表示和存储，且正数的原码和补码相同，负数的补码等于其原码按位取反在+1

注：浮点数不支持位运算，运算前会将其调用Math.trunc()取整在进行位运算

1. 数组方法操作

（1）创建数组：[]、new Array()、Array.of(el1,el2,…)、Array.from(likeArray)

（1、Array.of(el1,el2,…)：创建一个新数组实例，与Array构造函数的区别在于处理整数参数：Array.of(7)创建一个具有单一元素7的数组，而Array(7)创建一个有7个空位的数组而不是7undefined组成的数组

（2、Array.from(arrayLike[,mapFn[,thisArg]])：从一个类似数组或可迭代对象中创建并返回一个新的，浅拷贝的数组实例,第二个参数为处理数组元素的回调函数，第三个参数为执行回调函数时this对象

（2）合并数组：Array.prototype.concat(arr1[,arr2…]),合并两个或多个数组，不更改现有数组，而是返回一个新的数组

（3）转化为字符串：

（1、Array.prototype.join(separator)：以separator为分隔符拼接为字符串并返回

（2、Array.prototype.toString()：把数组转换为字符串，数组中的元素之间用逗号分隔

（4）填充数组：Array.prototype.fill(value[,start[,end]]),用一个固定值value填充从[start,end]位置的元素

（5）筛选指定类型的数组元素：

（1、Array.prototype.filter(cb(ele[,index[,arr]])[,thisArg])：返回一个新的、由通过测试的元素组成的数组，否则返回空数组

>参数cb是一个用来测试数组元素的函数，返回true则通过测试并保留该元素，否则不保留

>参数thisArg执行cb回调函数时，用于指定this的值

（2、Array.prototype.map(cb(ele[,index[,arr]])[,thisArg])：返回一个新数组，新数组中的每个元素都是cb回调函数的处理结果

>参数cb一个用于处理源数组arr的函数，并把处理结果返回

>参数thisArg执行cb回调函数时用于指定this的值

（6）Array.prototype.reduce(callback[,initialValue])：对数组中的每一个元素执行callback函数，并将其结果汇总为单个返回值，且第二个参数指定初始的默认值，若不给则取数组的第一个元素的值

（1、callback函数有四个参数：Accumulator(结果累计器)、CurrentValue(当前函数操作的值)、CurrentIndex(当前操作元素的索引值)、SourceArray(源数组)

（2、如果没有提供initialValue，则Accumulator的值为数组的第一个元素的值，如果提供了则为initialValue的值；

（3、如果数组长度为1，且initialValue未提供，或者数组为空且initialValue有值，则直接返回数组的这个元素或者initialValue的值，如果initialValue未提供且数组为空，则会抛出TypeError错误

1. 字符串方法操作
   1. String.prototype.charAt(index)：返回字符串中指定位置的字符，如果给的index值超出0到str.length – 1这个范围，则返回空字符串，若不传入index则从0开始
   2. String.prototype.charCodeAt(index)：方法返回0到65535之间的数，表示给定所以出的UTF-16代码单元，若不能被一个UTF-16编码单元单独表示则只能匹配Unicode代理对的第一个编码单元，若想获取完整的码点可以使用String.prototype.codePointAt(index)来获取
   3. String.prototype.codePointAt(index)：返回在字符串中指定的编码单元体现的数字，如果在所以处没找到元素则返回undefined
   4. String.prototype.endsWith(searchStr[,length])：用来判断searchStr是否是在string的末尾，且大小写敏感，返回boolean值，length用来指定源字符串的查找长度，默认是string.length
   5. String.prototype.indexOf(searchValue[,fromIndex])：查找searchValue在指定字符串中第一次出现的索引值，从fromIndex开始查找，默认为0，区分大小写
      * 1. fromIndex小于0时，则相当于传入0，查找整个字符串
        2. fromIndex大于等于str.length时，返回-1
        3. 当searchValue是一个空字符串时，fromIndex小于等于0时返回0，fromIndex大于0小于等于str.length时，返回fromIndex,fromIndex大于str.length时返回str.length
        4. 没有找到则返回-1
   6. String.prototype.localCompare(compareString[,locals[,options]])：返回一个数字用来指示引用字符串str在排序中位于比较字符串compareString的前面、后面，或者二者相同
      * 1. 如果compareString在str之前则返回正数，compareString在str之后则返回负数，二者相等时返回0
        2. 当比较大量字符串时，特别是比较大量数组时最好是创建一个Intl.Collator对象并使用compare函数，即用语言敏感字符串比较的collators构造函数
   7. String.prototype.matchAll(regexp)：返回一个包含所有匹配正则表达式及分组捕获结果的迭代器
   8. String.prototype.search(regexp)：若匹配成功则返回正则表达式regexp在字符串中首次匹配项的索引，否则返回-1
   9. String.prototype.repeat(count)：返回一个由string重复count次之后的一个新字符串，当count等于1时相当于返回一个新的从源字符串复制出来的字符串
2. Object方法操作
   1. Object.assign(target,…sources)：只会**浅拷贝**源对象sources自身的可枚举的属性到目标对象target,且此方法会调用源对象的[[[Get]]和目标对象target的[[Set]]，所以它会调用相关的getter和setter，返回目标对象
      * 1. 源对象中如果包含getter，可能不适合将新属性合并到目标对象target的原型中，若想将属性定义以及可枚举性复制到原型中，应使用Object.getOwnPropertyDescriptor()方法获取原对象sources中某一个属性的描述符对象，然后使用Object.difineProperty()方法为目标对象定义对应的一个属性以及该属性的描述符对象
        2. String类型和Symbol类型的属性都会被拷贝
        3. 此方法不会再sources对象的属性值为null或者undefined时报错
        4. 继承属性和不可枚举的属性是不能拷贝的
        5. 主要用于合并对象
   2. Object.create(proto, propertiesObject)：创建一个新对象，并可指定原型对象，以及向新对象添加多个带有描述符的自身可枚举属性而非原型链上的枚举属性，返回一个带着指定的原型对象和属性的新对象
      * 1. 使用proto参数来做为新创建的对象的\_\_proto\_\_，即原型对象
        2. 使用propertiesObject参数来定义新创建对象的自身可枚举属性，包含属性的描述符以及属性名称，对应Object.defineProperties()方法的第二个参数
   3. Object.defineProperties(obj, props)：直接在对象obj上定义一个或多个新的属性或者修改一个或多个现有属性，并返回该对象
      * 1. 参数props是一个对象，属性名是obj的属性名，每一个属性也是一个对象里面是相应的属性描述符以及属性的值value
        2. 是定义Symbol类型属性的方法之一
        3. 默认情况下定义的属性是不可修改、不可枚举、不可配置的
   4. Object.defineProperty(obj, prop, descriptor)：直接在对象obj上定义一个新属性或者修改一个现有属性，并返回该对象
      * 1. 参数prop要定义或修改的对象obj的属性名
        2. 参数descriptor将要定义或修改的属性的描述符对象
        3. 是定义Symbol类型属性的方法之一
        4. 默认情况下定义的属性是不可修改、不可枚举、不可配置的
   5. 属性描述符：目前对象中存在的属性描述符有两种主要形式：数据描述符和存取描述符（访问器描述符），且描述符是一个对象
      * 1. 数据描述符：是一个具有值的属性，该值可能是可写的也可能是不可写的。
        2. 存取描述符（访问器描述符）：是由getter-setter函数对描述的属性
        3. 描述符必须是二者之一，不能同时是两者
        4. 数据描述符和存取描述符均具有以下可选键值：

|  |  |  |
| --- | --- | --- |
| 键名 | 值 | 描述 |
| configurable | boolean | 仅该键值为true时，对应的属性的描述符才能被修改，同时该属性也能从对应的对象上被删除；为false时除value和writable之外的描述符对象的属性不可以被修改，对应属性不能从对象删除，默认为false |
| enumerable | Boolean | 仅该键值为true时，对应的属性才能出现在对应对象的枚举属性中，默认为false |
| writable | Boolean | 仅该键值为true时，键值value才能被赋值运算符改变，默认为false，仅数据描述符对象具有该属性 |
| value | 任意类型的值 | 任何有效的Javascript值，默认为undefined，仅数据描述符对象具有该属性 |
| get | 一个getter方法 | 一个给属性提供getter的方法，若没有getter则为undefined,当访问对应属性的时候会调用此方法，执行时没有参数传入，但是会传入this对象（由于继承关系，this可能不是定义对应属性的对象），默认为undefined，仅存取描述符对象具有该属性 |
| set | 一个setter方法 | 一个给属性提供setter的方法，若没有setter则为undefined，当属性值修改时，会执行该方法，且接受唯一参数即对应属性的新值，默认为undefined，仅存取描述符对象具有该属性 |

* + - 1. 如果一个描述符不具有value,writable,get,set任意一个关键字，那么将被认为是一个数据描述符
      2. 如果一个描述符同时有（value 或 writable）和（set 或 get）关键字，将会产生一个异常
      3. 如果被访问的对象的属性是被继承的，那么该属性的get和set方法会在子对象的属性被访问或修改时被调用，若这些方法用一个变量存值，那么该变量被所有子对象共享，根据this对象在get和set方法中指向的问题，可以在对应的子对象中定义变量来处理这种共享的情况
  1. Object.entries(obj)：返回给定对象obj自身可枚举属性的键值对数组，其排列与使用for…in循环遍历该对象的时返回的顺序一致，但是for…in循环也会枚举原型链中的可枚举属性，返回的数据结构如：[[key, value],[key, value]]这样的二维数组
     + 1. 将obj转换为Map对象，因为new Map()构造函数接受一个可迭代的entries二维数组，借助Object.entries()方法可以很容易的将Object转换为Map对象
  2. Object.freeze(obj)：可以**冻结**一个对象，返回处理后的obj
     + 1. 被冻结的对象再也不能被修改
       2. 不能向冻结的对象添加新的属性，不能删除现有属性
       3. 不能修改冻结的对象的已有属性的可枚举性、可配置性、可写性，以及不能修改已有属性的值
       4. 冻结的对象的原型也不能被修改
       5. 如果冻结的对象的属性是一个对象，则这个对象的属性是可以被修改的，除非这个对象也是一个冻结对象
       6. 如果传入的参数obj不是一个对象，在es5中会导致TyoeError，在es6中非对象参数会被视为要被冻结的普通对象，并被简单的返回
  3. Object.preventExtensions(obj)：让对象obj**变的不可扩展**，即永远不能添加新的属性，返回已经不可扩展的对象obj
     + 1. 对象obj仍然可以删除已有的属性
       2. 仍然可以向对象obj的原型上添加属性
       3. 若obj不是一个对象，在ES5中会导致TypeError,在ES6中会被视为一个不可扩展的普通对象，并直接返回
  4. Object.seal(obj)：**封闭**对象obj，阻止向对象添加新的属性并将所有现有属性标记为不可配置，且只要当前属性的描述符writable为true就可以改变该属性的值,返回已经封闭的对象obj
     + 1. 对象obj已有属性不能被删除
       2. 不能重新定义为访问器属性或者定义为数据属性
       3. 不会影响从原型链上继承的属性
       4. 若obj不是一个对象在ES5中会导致TypeError,在ES6中会被视为已被密封的普通对象，并直接返回它
  5. Object.fromEntries(iterable)：把键值对二维数组转换为一个对象,返回转换后的新对象
     + 1. 参数iterable应该是一个能够实现@iterator方法的的对象，返回一个迭代器对象，
       2. 是Object.entries()方法的反转，可以将Map、Array二维数组转换为Object
  6. Object.getOwnPropertyDescriptor(obj, prop)：返回指定对象obj上的一个自身属性prop对应的属性描述符对象，否则返回undefined
     + 1. ES5中若obj不是一个对象会导致TypeError,在ES6中会被强制转换为对象
  7. Object.getOwnPropertyDescriptors(obj)：获取对象obj所有的自身属性的描述符对象，如果没有属性则返回空对象
     + 1. 例子：浅拷贝一个对象，由于Object.assign()只能拷贝原对象自身的可枚举属性，而且无法拷贝属性的特性，并且访问器属性会被装换成数据属性，同时也无法拷贝原型上的属性，因此配合Object.getOwnPropertyDescriptors()方法，可以实现上述的内容，即：

Object.assign(

Object.getPrototypeOf(obj),

Object.getOwnPropertyDescriptors(obj)

)

实现对象的浅拷贝

* 1. Object.getOwnPropertyNames(obj)：返回一个由给定对象obj自身的包括不可枚举的且不是Symbol类型的属性的名字组成的数组
     + 1. 如果参数obj不是一个原始对象类型，在ES5中会导致TypeError,在ES6中会被强制转换为对象
  2. Object.getOwnPropertySymbols(obj)：返回一个由给定对象obj自身的所有Symbol属性组成的数组，否则返回一个空数组
  3. Object.getPrototypeOf(obj)：返回一个由给定对象obj的原型对象即（内部[[Prototype]]属性的值），如果没有继承属性则返回null
     + 1. 如果obj不是一个对象在ES5中会导致TypeError,在ES6中参数obj会被强制转换为Object
  4. Object.is(v1,v2)：判断两个值是否是相同的值，返回true或false
     + 1. 当v1和v2满足一下条件之一，则两个值v1和v2相同：
          1. 都是undefined
          2. 都是null
          3. 都是true或false
          4. 由相同个数的字符按照相同的顺序组成的字符串
          5. 指向同一个对象
          6. 都是数字且：都是正零+0、都是负零-0、都是NaN、都是除零和NaN以外的其它同一个数字
       2. 不同于==运算符，它会对两边的操作数在类型不同的情况下做隐式类型转换，然后在进行比较，但是Object.is()不会做这种类型转换
       3. 也不同于===运算符，===运算符会将数值-0和+0视为相等，并认为Number.NaN不等于NaN
  5. Object.isExtensible(obj)：判断一个对象**是否是可扩展的**，即能不能向该对象obj添加新的属性，返回true或false，对应的将对象变成不可扩展的方法是Object.preventExtensions()
  6. Object.isFrozen(obj)：判断对象obj**是否被冻结**，返回true或false
     + 1. 对象obj被冻结是指该对象obj不可扩展、所有属性不可配置、所有数据属性（即没有getter或setter组件的访问器的属性）不可写
       2. 对应的将对象冻结的方法是Object.freeze()
       3. 若参数obj不是一个对象，在ES5中会导致TypeError,在ES6中会被视为已冻结的普通对象，因此会返回true
  7. Object,isSealed(obj)：判断对象obj是否被密封，返回true或false
     + 1. 对应的将对象密封的方法是Object.seal()
  8. Object.keys(obj)：返回给定对象obj自身除Symbol类型以外的所有可枚举属性组成的数组
     + 1. 若obj不是一个对象在ES5中会导致TypeError,在ES6中会被强制转换为一个对象
  9. Object.hasOwnProperty(prop)：返回一个布尔值，即对象obj自身属性中是否有指定的属性prop
     + 1. 与in运算符不同，会忽略那些从原型链上继承的属性
  10. Object.isPrototypeOf(obj)：用于测试一个对象prototypeObj是否存在于另一个对象obj的原型链上，返回布尔值
      + 1. 语法结构：prototypeObj.isPrototypeOf(obj)
        2. 若prototypeObj为undefined或null，则会抛出TypeError
  11. Object.propertyIsEnumerble(prop)：返回一个布尔值，表示指定的属性prop是否可枚举，若没有指定prop则返回false，且不考虑原型上的属性
  12. Object.setPrototypeOf(obj, prototypeObj)：设置对象obj的原型（即内部的[[Prototype]]属性）到另一个对象prototypeObj或null
      + 1. 如果prototypeObj不是一个对象或null,则什么都不做
  13. Object.values(obj)：返回给定对象obj自身所有可枚举属性的值的数组

1. BOM操作：即Browser Document Model，操作浏览器窗口的API
   1. window对象：通过JavaScript访问浏览器窗口的一个接口，又是ECMAScript规定的Global对象
      * 1. 直接定义的全局变量不能通过delete运算符删除，而直接定义在window对象上的属性可以通过delete运算符删除
        2. window.applicationCache：即当前window中的应用缓存，返回的是一个offlineResourceList对象的引用，即离线资源列表对象
        3. Window.closed：只读属性，判断所引用的窗口是否关闭，true为关闭，false为打开
        4. window.crypto：只读属性，返回与全局对象关联的Crypto对象，此对象允许网页可以访问一些加密相关的服务
        5. window.customElements：只读属性，返回一个CustomElementRegistry对象的引用，可以用于注册一个新的自定义元素，也可用于获取之前定义过的自定义元素的信息
        6. window.devicePixelRatio：此属性返回当前显示设备的物理像素分辨率与CSS像素分辨率的比值。即一个CSS像素的大小相对于一个物理像素的大小的比值，且可以重写此属性的值
        7. window.frameElement：返回嵌入当前window对象的元素，如：<iframe>或者<Object>,如果当前window已经是顶层窗口，则返回null
        8. window.frames：返回当前窗口，一个类数组对象，列出了当前窗口的所有直接子窗口
           1. window.frames === window计算结果为true
           2. 类数组的每一项都代表了窗口对应给定的对象的<frame>或<iframe>的内容，而不是DOM元素，即与document.getElementByTagName(‘iframe’)[0] .contentWindow是相同的
        9. window.history：是一个只读属性，用来获取History对象的引用，History对象提供了操作浏览器会话历史的接口；
           1. 向后跳转页面：window.history.back()
           2. 向前跳转页面：window.history.forward()
           3. 跳转到history中指定的一个页面：window.history.go(num),参数num的值是以当前页面相对位置来标志的，且当前页面的相对位置标志为0
           4. 可以通过window.history.length来获取当前历史纪录条目的个数
           5. 添加历史纪录中的条目：window.history.pushState(stateObj, pageName, pageUrl)方法

参数stateObj是一个javascript对象，大小为640k,会在用户导航到对应页面是传递给popstate监听事件的state参数

参数pageName即对应页面的名称

参数pageUrl即新页面的地址，且必须与当前URL同源，且是可选参数，不传默认为当前URL，可为绝对路径也可为相对路径，若为相对路径则相对于当前URL处理

调用此方法后浏览器并不会立即加载这个地址，可能稍后加载或重新打开浏览器加载

* + - * 1. 修改历史纪录中的条目：window.history.replaceSate(stateObj, pageName, pageUrl)方法,用法和pushState相同，只是为了相应用户操作修改当前的历史纪录项的状态对象或者URL地址
        2. 在重新加载浏览器时可以通过window.history.state来获取当前记录的状态对象
      1. 获取浏览器窗口可视区域（即窗口的内层）的高度和宽度：
         1. window.innerHeight: 获取浏览器窗口可视区域的高度,若有水平滚动条则包含滚动条的高度
         2. window.innerWidth：获取浏览器窗口可视区域的宽度，若有垂直滚动条则包含垂直滚动条的宽度
      2. 获取整个浏览器（即窗口的外层）的高度和宽度：
         1. window.outerHeight：返回整个浏览器（即窗口的外层高度）的高度，单位像素，包括侧边栏、窗口镶边、窗口调正边框，为只读属性没有默认值
         2. window.outerWidth：返回整个浏览器（即窗口的外层高度）的高度，单位像素，包括侧边栏、窗口镶边、窗口调正边框，为只读属性没有默认值
      3. 设置窗口的大小：（注意火狐7以上的限制）
         1. window.resizeBy(xDelta, yDelta)：窗口水平方向xDelta和垂直方向yDelta变化的像素值
         2. window.resizeTo(aWidth,aHeight)：动态调整窗口大小，aWidth，一个整数，包括滚动条、窗口边框；aHeight，一个整数，包括滚动条、标题栏、窗口边框
      4. window.length：该窗口中框架的数量（框架包括frame和iframe两种元素）
      5. window.location：只读属性，返回一个Location对象，其中包含有关文档当前位置的信息
         1. window.location.assign(url)：会触发窗口加载并显示指定的URL的内容
         2. window.location.reload(booelan)：用来刷新页面，该方法只有一个布尔类型的参数，为true时将强制浏览器从服务器加载页面资源，否则浏览器可能从缓存中读取页面，默认为false
         3. window.location.replace(url)：以给定的URL来替换当前的资源，与window.location.assign(url)不同的是调用replace()方法后，当前页面不会保存的历史记录中，用户点击回退将不会再跳转到该页面
      6. window.opener:返回打开当前窗口的那个窗口的引用，若当前窗口不是由其他窗口打开的，则返回null
      7. window.origin：WindowOrWorkerGlobalScope接口的origin只读属性返回全局范围的origin,序列化为一个字符串，即当前窗口的由协议、主机名、端口号组成的URL地址，如果没有这些则返回null
      8. window.pageYOffset：只读属性，scrollY的别名
      9. window.parent：返回当前窗口的父窗口对象
         1. 若当前窗口没有父窗口，则它的parent属性为自身的引用
         2. 若当前窗口是一个<iframe>、<object>或者<frame>，则它的父窗口是嵌入它的窗口
      10. window.top：返回当前窗口最顶层的父窗口
      11. window.screen: 返回当前window的screen对象，该对象实现了Screen接口，返回当前渲染窗口中和屏幕有关的属性
      12. window.screenLeft: 只读属性，返回浏览器左边框到左边屏幕边缘的距离，单位像素，且是旧属性window.screenX的别名
      13. window.screenTop：只读属性，返回浏览器的上边界到屏幕顶部边缘的距离，单位像素，且是旧属性window.screenY的别名
      14. window.scrollbars：返回可以检查滚动条可见性的滚动条对象
      15. window.scrollMaxX：只读属性，返回可以水平滚动的最大值
      16. window.scrollMaxY：只读属性，返回可以垂直滚动的最大值
      17. window.scrollX：只读属性，返回当前水平滚动的距离
      18. window.scrollY：只读属性，返回当前垂直滚动的距离
      19. window.self：返回一个指向当前window对象的引用
      20. window.top：返回最顶层window对象的一个实例
      21. window.windowState：只读属性，返回window对象的当前状态，且有4个状态
          1. 状态：STATE\_MAXIMIZED，对应的值是1
          2. 状态：STATE\_MINIMIZED，对应的值是2
          3. 状态：STATE\_NORMAL，对应的值是3
          4. 状态：STATE\_FULLSCREEN，对应的值是4
  1. window对象的方法：
     + 1. atob：此方法是继承自WindowOrWorkerGlobalScope对象，对经过base64编码的字符串进行解码，若传入的字符串不是有效的base64字符串则抛出DOMException
       2. btoa：此方法是继承自WindowOrWorkerGlobalScope对象，从String对象中创建一个base64编码的ASCII字符串，其中每个字符都被视为一个二进制数据字节
       3. createImageBitmap(image[,sx,sy,sw,sh[,options]])：此方法存在于windows和workers中，接受各种图像来源，并返回一个Promise，resolve为ImageBitmap，可选参数可以将图像裁剪成自(sx,sy)且宽为sw，高为sh的像素的矩形
          1. 参数options：设置返回图像的配置

imageOrientation：指示图像是按原样呈现还是垂直翻转，默认是none不翻转，或flipY

premultiplyAlpha：指示位图颜色通道由Alpha通道预乘，选择其一：none，premultiply，default（默认）

colorSpaceConversion：指示图像是否使用色彩空间转换进行解码。none或default(默认)

resizeWidth：指示新宽度的长整数

resizeHeight：指示新高度的长整数

resizeQuality：指定图像缩放算法。选择其一pixelated，low(默认)，medium或high;

* + - 1. fetch(input[,init])：此方法位于WorkerOrGlobalScope这一个mixin中的fetch()方法用于发起获取资源的请求。返回一个promise，这个promise在请求响应后被resolve,并传回Response对象
         1. 当遇到网络错误时，返回promise会被reject，并传回TypeError
         2. 此方法是由Content Security Policy的connect-src指令控制，而不是它请求的资源
         3. fetch()方法的参数和Request()构造器的参数是一样的
         4. 参数input：定义要获取的资源，可能是一个USVString字符串，包含要获取资源的URL。一些浏览器或接受blob:和data:作为schemes，或者是一个Request对象
         5. 参数init：一个配置项对象，包括所有对请求的设置，可选参数有：

method：请求使用的方法，如GET、POST

headers：请求的头信息，形式为Headers的对象或包含ByteString值的对象字面量

body：请求的body信息：可能是一个Blob、BufferSource、FormData、URLSearchParams或者USVString对象。注意GET或HEAD方法的请求不能包含body信息

mode：请求的模式，如cors、no-cors或same-origin

credentials：请求的credentials，如omit、same-origin或者include。为了在当前域名内自动发送cookie，必须提供这个选项

cache：请求的cache模式：default、no-store、reload、no-cache、force-cache或者only-if-cached.

redirect：可用的redirect模式:follow(自动重定向)，error(如果产生重定向将自动终止并且抛出一个错误)，或者manual(手动重定向)

referrer：一个USVString可以是no-referrer、client或者一个URL。默认是client

* + - 1. window.getComputedStyle(ele[,pseudoElt])：返回一个实时的CSSStyleDeclaration对象，该对象在应用活动样式表并解析这些值可能包含的任何基本计算后报告元素的所有CSS属性的值，且当元素的样式更改时，它会自动更新自身
         1. 参数ele：用于获取计算样式的元素
         2. 参数pseudoElt：指定要匹配的伪元素的字符串，必须对普通元素省略或者null
         3. 可以通过document.defaultview对象来调用getComputedStyle方法
         4. getComputedStyle的返回值是resolved values,跟在CSS2.1中的computed values是相同的值。但对一些旧的值比如width、height、padding它们的值又为used values。在CSS2.0中computed values就是属性的最终值，但在CSS2.1中computed values为布局前的值， used values为布局后的值，区别在于百分比可以代表的宽度，在布局后会被替换为像素值
      2. window.getSelection()：返回一个Selection对象，表示用户选择的文本范围或光标的当前位置；
         1. 返回的对象是一个[Selection](https://developer.mozilla.org/zh-CN/docs/Web/API/Selection)对象，可通过连接一个空字符串或者使用toString()方法转换为字符串（被选中的文本）
         2. 还可使用Document.getSelection()方法
      3. Window.matchMedia(mediaQueryString)：返回一个新的MediaQueryList对象，表示指定的媒体查询字符串解析后的结果
         1. 语法结构如：window.matchMedia(‘(min-width: 500px)’)
         2. 可根据媒体查询的结果做网站页面排版的调整
      4. window.minimize: 让当前窗口最小化，可通过window.moveTo()方法让窗口恢复正常显示
      5. window.moveBy(deltaX, deltaY)：根据指定的值，移动当前窗口
         1. 参数deltaX：表示窗口在水平方向移动的像素值
         2. 参数deltaY：表述窗口在垂直方向移动的像素值
         3. 可以使用负值作为该函数的参数
         4. 不能移动非window.open打开的窗口或tab，当一个窗口里有多于一个Tab时，不能移动该窗口
      6. window.moveTo(x, y)：将当前窗口移动到指定的坐标位置
         1. 参数x：要移动到的位置横坐标
         2. 参数y：要移动到的位置纵坐标
      7. window.requestAnimationFrame(callback)：注册一个执行动画的函数，告诉浏览器执行一个动画，并且要求浏览器在下一次重绘之前调用指定的回调函数更新动画
         1. 运行在后台标签页或者隐藏的<iframe>里时，requestAnimationFrame()方法会被暂停调用以提高性能和电池寿命
         2. 参数callback回调函数：会被传入DOMHighResTimeStamp参数，指示由requestAnimationFrame()排队的回调开始触发的时间，是一个十进制毫秒数，最小精度1ms
         3. requestAnimationFrame()函数返回一个非零正整数，最为此次动画的请求ID，可以将此ID传递给window.cancelAnimationFrame()方法已取消对应的动画回调函数
      8. window.requestIdleCallback(callback[,options])：会在浏览器空闲时期依次调用函数，返回一个无符号长整数，可以把这个整数传递给window.cancelIdleCallback()方法来结束回调；
         1. 按照先进先出原则执行回调
         2. 不会对如动画和用户交互这样延迟敏感的事件产生影响
         3. 如果定义了超时时间，则为了能在超时前执行函数而打乱执行顺序
         4. 参数callback：一个在事件循环空闲时即将被调用的函数的引用。函数会接受一个名为IdleDeadline的参数，这个参数可以获取当前空闲时间以及回调是否在超时时间前已经执行的状态
         5. 参数options：可选的配置参数，

timeout：值为正整数时，做为调用callback的最后期限，当指定的时间过去后回调仍未被执行，那么回调会在下一次空闲时期强制执行，可能会对性能造成影响

* 1. window对象下的事件：Events,使用window.addEventListener()方法添加事件
     + 1. beforeunload：在浏览器窗口关闭或者刷新时触发，同时当前页面不会直接关闭，可以点击弹窗内的确定按钮来关闭或刷新，也可以取消关闭或刷新
          1. 若事件处理函数为Event对象的returnValue属性赋值非空字符串，浏览器会弹出一个对话框，来询问用户操作，，没有赋值时该事件不做任何相应；
       2. blur：当元素失去焦点时触发该事件，相对应的聚焦事件是focus
       3. focus：当元素获得焦点时触发，相对应的失焦事件是blur
       4. DOMContentLoaded：当初始的HTML文档被完全加载和解析完成之后，该事件被触发，而无需等待样式表、图像和子框架的完成加载。
          1. DOMContentLoaded事件必须等待其所属script标签之前的样式表加载并解析完成才会触发
          2. IE8中使用readystatechange事件来检测DOM文档是否加载完毕。更早版本的IE中可通过每隔一段时间执行一次document.documentelement.doScroll(‘left’)来检测此状态，因为在DOM加载完毕之前执行这段代码会抛出错误
       5. error：当资源加载失败或无法使用时，会在Window对象触发error事件。

1. DOM操作
2. 跨域
   1. 本质上是浏览器的同源策略，即一种安全策略，所谓同源就是：域名、协议、端口号完全相同，却不会对script标签的src,img标签的src,a标签的href做限制，这些本质上也是一种get请求，且JSONP只支持get请求
   2. 方法一：通过在客户端定义好获取数据的方法，然后通过script标签的src属性将方法通过地址参数的形式即（&callback=getdata）拼接到地址后面来调用对应的文件，在后台执行这个回调来获取数据
   3. 方法二：通过window.name来进行数据跨域传递
   4. 方法三：通过CORS（跨域资源共享），需要浏览器和服务端同时支持
      * 1. CORS请求设置的响应头字段，都以Access-Control-开头：
           1. Access-Control-Allow-Origin:必选，值要么是请求时Origin的值，要么是一个\*，表示接受任意域名的请求
           2. Access-Control-Allow-Credentials:可选，布尔值，默认情况下Cookie不包含在CORS请求中，为true时表示服务器明确许可，Cookie可以包含在请求中一起发个服务器,若服务器不要浏览器发送Cookie，删除该字段即可
           3. Access-Control-Expose-Headers:可选，指定XHR对象的getResponseHeader()方法除了能拿到6个基本字段（Catch-Control、Content-Language、Content-Type、Expires、Last-Modified、Pragma）以外的一些字段
   5. 方法四：使用nginx代理跨域：于CORS原理一样，通过配置文件设置请求相应头，

通过Nginx配置一个代理服务器(域名与当前域名相同，但端口号不同) 做跳板机，反向代理访问目标地址的接口，并可以顺便修改cookie中domain的信息，方便当前域cookie写入，实现跨域访问

* 1. 方法五：通过nodejs中间件跨域（http-proxy-middleware）,通过启动一个代理服务器，实现数据的转发，也可以通过设置cookieDomainRewrite参数修改响应头中cookie中域名，实现当前域的cookie写入，方便接口登陆认证
  2. 方法六：document.domain+iframe跨域，仅限于主域名相同，子域名不同的跨域场景。两个页面都通过js强制设置document.domain为基础主域，就实现了跨域
  3. 方法七：location.hash + iframe跨域，a页面要与b页面通信，则需要通过中间页c来实现，三个页面，不同域之间利用ifram的location.hash传值，相同域之间直接通过js访问来通信，（c页面与a页面或者b页面同域）
  4. 方法八：window.name + iframe跨域，window的name值在不同的页面（甚至不同的域名）加载后依旧存在，并且可以支持非常大的值，约为2M
  5. 方法九：window.postMessage跨域，可以解决一下几种场景：
     + 1. 页面和其打开的新窗口的数据传递
       2. 多窗口之间消息传递
       3. 页面与嵌套的ifram消息传递
       4. 以上三种跨域数据传递
     1. 用法：window.postMessage(data, origin)：
        1. data：要传递的数据
        2. origin：协议+主机+端口号，也可以设置为“\*”，表示可以传递给任意窗口，如果要指定和当前窗口同源的话设置为“/”
  6. 方法十：WebSocket协议跨域，实现了浏览器和服务器全双工通信

1. 服务端cookie、session
   1. http是一种无状态协议，即http是基于TCP的，从http1.1开始，默认使用持久连接，这个过程中，客户端可以向服务端发送多次连接，但是各个请求之间没有任何联系
      1. 无连接：每次连接只处理一个请求，服务器处理完客户端一次请求，等到客户端做出回应之后便断开连接
      2. 无状态：即指服务器对于客户端的每一次请求，都认为它是一个新的请求，且每一个会话之间没有联系
      3. 无连接的维度是连接，无状态的维度是请求
      4. 持久连接：本质上就是客户端和服务器通信时，会建立一个持久化的TCP连接，这个连接不会随着请求结束而关闭，通常会保持连接一段时间，现有的持久连接有两种：
         1. HTTP/1.0+的keep-alive,即每次发起http请求的时候会在请求头（Request header）中添加一个connection: keep-alive的参数，这表示声明一个持久连接
         2. HTTP/1.1的persistent,默认情况下是开启的，只有请求头中有connection: close时，才会在事务结束之后关闭持久连接，客户端和服务器也可以随时关闭持久连接，其次要传递正确的content-length
         3. 根据HTTP/1.1的特性，最好不要和HTTP/1.0建立持久连接，最后要做好重发的准备
      5. 无状态指的是信息，而这些信息是由服务端所维护的，与客户端交互的信息（状态信息），因为HTTP本身不保存任何用户的状态信息，所以HTTP是无状态的协议
   2. 如何HTTP本身不实现状态维护？
      1. HTTP本身维护状态信息的代价很高，如果维护的话后面的一些行为也会受到状态信息的影响
      2. HTTP最初是用来浏览静态文件的，无状态协议已经足够
   3. 如何保持状态信息？
      1. cookie和session体系，即用户第一次访问服务器时，服务器响应报头通常会出现一个Set-Cookie响应头，本质就是在本地设置一个cookie,当用户再次访问服务器时，http会附带这个cookie过去，由服务器来确认是否是同一次会话
   4. Cookie，即服务器发送给客户端的小量信息，以（key-value）的形式存在，保存在客户端
      1. 参数domain：可以访问该Cookie的域名，且第一个字符必须为‘.’
      2. 参数maxAge：Cookie的失效时间，单位秒
         1. 正数是超过多少秒后失效
         2. 负数，则该Cookie是临时Cookie，关闭浏览器即失效，浏览器也不会以任何形式保存该Cookie
         3. 0，表示删除该Cookie
      3. 参数path：该Cookie的使用路径，必须以‘/’结尾
         1. 为‘/’时，本域名下的contextPath都可以访问
      4. 参数secure：指该Cookie是否仅被用在安全协议传输，安全协议包括HTTPS,SSL等，默认为false
      5. 参数isHttpOnly：用来限制非HTTP协议的程序接口对客户端的Cookie访问，有效防止XSS攻击
   5. Cookie是不可跨域的，且Cookie只关注域名，在没有经过任何处理的情况下，二级域名不同也无法访问
   6. 不要超过数量和大小限制（在4kb左右），超过长度会被直接截取丢弃，超过数量则会随机删除旧Cookie或者删除最老的Cookie，以便保存新的Cookie
   7. Session，是以服务器保存状态的
      1. 机制原理：当客户端请求创建seesion时，服务器会先检查这个请求中是否包含一个session的标识-----sessionId
         1. 如果包含一个sessionId，则服务器会按照这个sessionId找出这个session来使用
         2. 如果没有包含sessionId,则会为此客户端创建一个session并生成于此对应的sessionId
         3. 大多数情况下会使用cookie保存这个sessionId
2. Canvas、SVG
   1. 初始化canvas：使用canvas元素以及定义宽度和高度；canvas是基于状态的绘图，先设置绘图的状态，然后调用具体的绘图函数
      1. <canvas id=’canvasId’ width=”500” height=”500”>这是不兼容的浏览器显示的内容</canvas>
   2. Canvas定义一个2d的绘图环境：使用getContext(‘2d’)来创建一个2d的画图环境，此方法会返回一个CanvasRenderingContext2D对象的实例，用来操作canvas上下文
   3. Canvas描边和填充：
      1. 描边：stroke就是只在图形边缘画线
      2. 填充：fill就是用指定的样式（颜色、渐变、图像）填充图像
   4. Canvas绘制矩形：矩形是唯一可以在2d上下文中直接绘制的形状
      1. 绘制相关方法：fillRect()填充矩形、strokeRect()描边矩形、clearRect()清除矩形区域,它们接收四个参数：矩形开始绘制的x轴坐标、y轴坐标、矩形的宽度和高度，单位是像素
   5. Canvas绘制路径：通过路径可以创造出复杂的形状和线条
      1. 注意事项：
         1. 绘制路径必须以方法beginPath()开始；
         2. 可以使用moveTo(x,y)方法来移动画笔的位置；
         3. 可选的以closePath()方法结束，如果使用此方法则会闭合路径图，否则会留有一个缺口；
         4. 绘制结束后使用stroke()描边或者fill()填充，来修饰绘制的图形，且两个方法可以同时使用
         5. 当需要重新绘制一条路径时，需要调用beginPath()方法重新定义
         6. 在绘制路径时，使用beginPath()加上lineTo(x, y)相当于moveTo(x, y)
         7. 绘制带有边框的填充图案时，要先填充在描边，否则会将内边框一并填充掉，导致边框宽度减小
         8. 当线条宽度过大时，线条连接处会出现缺口，这时一定要用closePath()方法关闭路径
      2. 路径绘制方法
         1. arc(x, y, radius, startAngle, endAngle, counterclockwise):用于绘制一条弧线或者扇形，圆心（x,y）, 半径radius，起始角度startAngle，结束角度endAngle，是否是逆时针画图counterclockwise，默认为false
            1. 在使用arc()函数之前使用moveTo()可以绘制一个扇形，若不使用moveTo()函数则会绘制出一个弧线或者圆的一部分
         2. arcTo(x1, y1, x2, y2, radius)：从锚点即（前一个moveTo()或者lineTo()等函数的结束点）开始，当前锚点与（x1, y1）连接的直线，以及（x1, y1）与（x2, y2）连接的直线，作为指定半径（radius）的圆的切线，依此画出的两条切线之间的弧线路径；
         3. bezierCurveTo(c1x, c1y, c2x, c2y, x, y)：用于绘制三次贝塞尔曲线路径的方法，点（c1x, c1y）和点（c2x, c2y）是控制点，（x, y）是曲线的结束点，起始点是当前路径的最后一个点，可以通过moveTo()进行修改；
         4. lineTo(x, y): 绘制一条直线，起始点是当前路径的最后一个点，可通过moveTo(x, y)进行修改
         5. quadraticCurveTo(cx, cy, x, y):绘制二次贝塞尔曲线，点（cx, cy）为控制点， 点（x, y）为结束点，起始点是当前路径最新的点，可以使用moveTo(x, y)进行修改
         6. rect(x, y, width, height)：绘制的是矩形路径而不是独立的形状，因此需要使用stroke()或者fill()方法去描边或者填充，此方法不需要beginPath()方法定义开始
         7. ellipse(x,y,radius,radiusy,rotation,startAngle,endAngle,anticlockwise):
            1. 添加椭圆路径方法
            2. 圆心在(x,y)
            3. 半径分别是radiusx,radiusy
            4. 按照anticlockwise即是否逆时针方向（默认false顺时针）指定的方向，从startAngle开始绘制，到endAngle结束，
            5. rotation即椭圆的旋转角度，以弧度表示（非角度度数）
   6. Canvas状态的保存和恢复：它们是以堆栈（先进后出）的形式保存Canvas上下文的设置和状态
      1. 保存：save()，保存Canvas的全部状态，包括：
         1. 当前变换的矩阵
         2. 当前的剪切区域
         3. 当前的虚线列表
         4. 以及属性的当前值
      2. 恢复：restore()，通过在绘图状态栈中弹出顶端的状态，将Canvas恢复到最近一次的保存状态，如果没有保存状态则不做任何改变
         1. **注**：只是恢复Canvas的状态，并不会清除掉之前绘制的图形
   7. Canvas图形变换：
      1. 平移：translate(x, y),将坐标原点移动到（x, y）,先设置平移在绘制图形
      2. 旋转：rotate(angle)，围绕原点顺时针旋转角度angle弧度，并且用弧度表示
      3. 缩放：scale(scaleX, scaleY)在x方向乘以scaleX, 在y方向乘以scaleY,默认值1
         1. 缩放会将对应元素的所有属性的值进行缩放处理
      4. 组合变换：transform(a, b, c, d, e, f)与setTransform(),使用矩阵多次叠加当前变换的方法，矩阵由方法的参数进行描述，分别可以缩放、倾斜、旋转、平移。
         1. a：水平水平缩放
         2. b：水平倾斜
         3. c：垂直倾斜
         4. d：垂直缩放
         5. e：水平移动
         6. f：垂直移动
   8. canvas渐变：
      1. 线性渐变：使用canvas上下文调用createLinearGradient(x1,y1,x2,y2)创建一个线性CanvasGradient对象，然后赋值给fillStyle或者strokeStyle属性
         1. (x1,y1)开始点坐标
         2. (x2,y2)结束点坐标
         3. 沿参数坐标指定的直线的渐变
      2. 放射性渐变：使用canvas上下文调用createRadialGradient(x1,y1,r1,x2,y2,r2)创建一个由两个圆初始化的放射性CanvasGradient对象，然后赋值给fillStyle或者strokeStyle属性
         1. (x1,y1)第一个圆开始的位置，r1第一个圆的半径
         2. (x2,y2)第二个圆开始的位置，r2第二个圆的半径
      3. 可以使用CanvasGradient对象的addColorStop(offset, color)方法设置多阶段的颜色值
         1. offset：0到1之间的值，超出范围包INDEX\_SIZE\_ERR错误
         2. color：颜色值，不能被解析则抛出SYNTAX\_ERR错误
   9. [CanvasRenderingContext2D.filter](https://developer.mozilla.org/zh-CN/docs/Web/API/CanvasRenderingContext2D/filter)属性：提供模糊、灰度等过滤效果的属性，类似CSS中的filter属性，并且接受相同的函数
      1. 接受DOMString字符串，可以包含一个或多个函数
   10. canvas.drawImage(image, sx,sy,sWidth,sHeight,dx,dy,dWidth,dHeight):可以进行图片绘制或者裁切图片
       1. 当只有前三个参数时，只将源图像image绘制到从点(sx,sy)开始宽高为图片的宽高
       2. 当只有前五个参数时，只将源图像image绘制到从点(sx,sy)开始宽为sWidth,高为sHeight
       3. 当全部参数都传时，将源图像image按照前五个参数进行绘制，然后从点(dx,dy)开始裁剪宽度为dWidth，高为dHeight
   11. 根据角度和半径计算坐标点：
       1. X:centerX + r \* cos(angle \* 3.14/180)
       2. Y:center + r \* sin(angle \* 3.15/180)
   12. 非零环绕原则：即两条路径的绘制方向相反时，相交汇的部分就会出现这种情况，可以利用这种原则绘制圆环等镂空图形
   13. isPointInPath: 此方法用于检测一个点(x, y)是否在canvas的路径内，且这个点是相对于canvas画布而言的，而不是相对于整个窗口，返回布尔值
3. Webgl、webworker
4. 本地存储和离线存储
   1. WebStorage：大小限制在5M左右，超出会抛出错误，仅仅可以存储字符串
      1. LocalStorage：
         1. 存储在本地，可设置过期时间，亦可永久存储
         2. 自行维护存储空间
      2. sessionStorage：
         1. 存储大小随着浏览器的不同而不同
         2. 浏览器网页会话级存储，关闭网页数据随之清除
         3. 同一域名不同标签页不共享
   2. indexedDB：是一种事务性数据库，NoSQL数据库，使用js对象存储数据
      * 1. 通过window.indexedDB.open(‘数据库名称’, ‘数据库版本号’);来创建或者访问数据，并返回一个IDBOpenDBRequest对象实例
        2. 添加数据时需要先创建数据对象仓库,通过createObjectStore(‘仓库名称’, ‘仓库的主键以及是否递增，是一个对象’)创建数据对象仓库
        3. 创建仓库时必须是在数据库更新的情况下才能创建，因为indexedDB API中不允许数据库中的数据仓库在同一个版本中发生变化
        4. 创建事务，进行数据对象仓库的读写操作，通过transaction(‘数据对象仓库名’, ‘创建的事务模式’)来创建事务
        5. 操作对应的数据对象仓库，通过事务的objectStore(‘仓库名’)方法来获取数据对象仓库并返回一个IDBObjectStore对象实例，进行数据的的增(add())、删(delete())、改(put())、查(get())
      1. 数据库事务：包含了一系列的对数据库的读/写操作,使用transaction(‘数据对象仓库名’, ‘创建的事务模式’)，并返回一个IDBTransaction对象实例
         1. 目的：
            1. 为数据库操作序列提供了一系列从失败中恢复到正常状态的方法，同时提供了数据库即使是在异常情况下仍能保持一致性的方法
            2. 当多个程序在并发访问数据库时，可以在这些程序之间提供一个隔离方法，以防止彼此的操作互相干扰
         2. 特点：数据库事务有四个特性，简称ACID
            1. 原子性(Atomicity)：事务作为一个整体被执行，包含在其中的对数据库的操作要么全部执行，要么都不执行
            2. 一致性(Consistency)：事务应该确保数据库的状态从一个一致状态转变为另一个一致状态，一致状态的含义是数据库中的数据应满足完整性约束
            3. 隔离性(Isolation)：多个事务并发执行时，一个事务的执行不应影响其他事务的执行
            4. 持久性(Durability)：已被提交的事务对数据库的修改应该永久保存在数据库中
         3. 事务模式：readonly、readwrite、versionchange，默认模式是readonly
      2. 使用游标：通过store.openCursor(),获取一个主键值区间内的数据,并返回一个IDBRequest对象实例;
         1. openCursor()：使用数据对象仓库store的openCursor(range?, direction?)来创建并打开游标,不传参数会取出所有的数据
            1. range：即定义的主键值的一个范围，有四种方式可以定义范围,通过IDBKeyRange对象的一下方法:

upperBound()：接受两个参数，第一个参数是最大边界值；第二个是一个布尔值，定义这个范围是否包含最大边界值，默认false(包含)

lowerBound(): 接受两个参数，第一个参数是最小边界值；第二个是一个布尔值，定义这个范围是否包含最小边界值，默认false(包含)

bound()：接受四个参数，第一个参数是最小边界值；第二个参数是最大边界值；第三个值是一个布尔值，定义这个范围是否包含最小边界值，默认false(包含)；第四个参数也是一个布尔值，定义这个范围是否包含最大边界值，默认false(包含)

only()：接受一个参数，即不是一个范围，此方法只查询给定主键值的数据一条数据

* + - * 1. direction：即获取数据的方向，有以下几种值：

next：默认值，按主键值升序排列，读取出主键值相等的数据

nextunique：按主键值升序排列，只读取主键值相等的第一条数据

prev：按主键值降序排列，读取主键值相等的数据

prevunique：按主键值降序排列，只读取主键值相等的第一条数据

* + - * 1. 在取出数据时可以使用如下方法对数据进行操作：

value:获取当前取出的数据的值

update():更新当前取出的数据的内容

delete():删除当前取出的数据

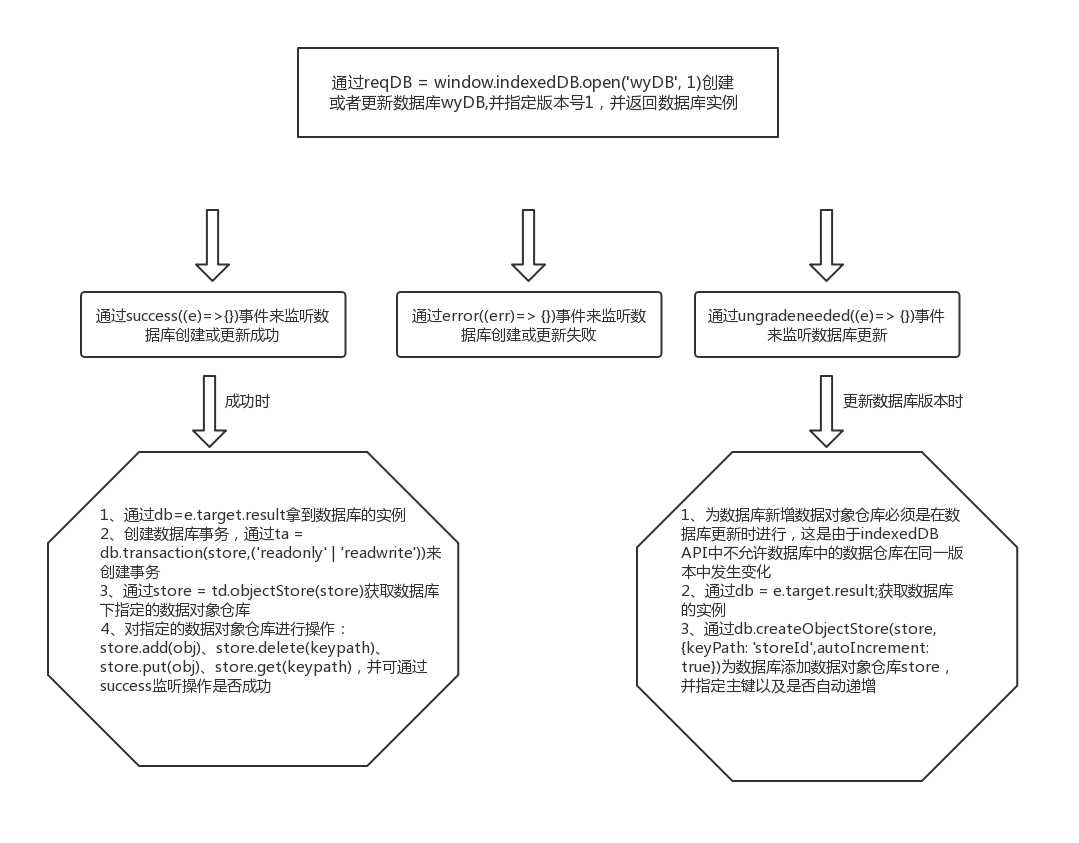
continue():继续获取下一条符合条件的数据

* + 1. 使用索引有条件的查询数据：索引必须是在数据库更新时创建即在upgradeneeded监听事件中创建
       1. 通过数据仓库store.createIndex(indexName, keyPath, optionalParameters)来创建索引
          1. indexName：索引名，不能重复
          2. keyPath：需要建立索引的数据对象仓库的属性，可以是单个的key值，也可以是一个包含key值的集合
          3. optionalParameters：是一个对象，有两个属性:

unique：规定索引值是否是唯一，true表示索引值不可以重复，false表示可以重复

multiEntry：当方法的第二个参数keyPath是一个集合时，true表示为每一个索引值建立一条索引，false表示将整个索引集合作为一个索引，默认为true

* + - 1. 使用创建好的索引查询数据：
         1. 使用数据对象仓库var index = store.index(indexName)来获取已经创建好的索引，并返回IDBIndex对象
         2. 使用获取到的index索引对象调用index.openCursor(rang, direction)进行数据的获取
    1. IndexedDB执行流程图：



* 1. Cookie：
     + 1. 存储大小限制在4K左右
       2. 但域名数量限制在50个左右
       3. 污染请求头，浪费流量

## Javascript高级

1. call、apply、bind、new等原理解析
2. 原型链深入
3. 闭包深入
4. 执行上下文和作用域链
5. 作用域链
6. 理解面向对象编程
7. 创建对象
8. 对象继承
9. Class以及构造函数之间的关系、区别、用法
10. 函数操作
11. Session、cookie、token进行接口验证的方式、区别
    1. 出现顺序：Session > cookie > token
    2. 区别：session和cookie是有状态的，由服务端提供，并且在服务器和客户端各保存一份，token是无状态的，由服务端的密钥签名发送到客户端，只在客户端保存，在数据请求时，将token随着请求头发送到后端，后端根据自己的密钥进行has256解码编译，进行验证，且有时效性

## ES6、7、8、9、10深入学习

1. 新增的基础数据类型
2. 新增的引用类型
3. 理解Promise以及机制、原理
4. 阅读Promise源码深入理解
5. Array、Math、Object、String、Regesp中新增的方法、属性以及变更
6. 理解Async&Await以及原理、机制、
   1. 是Generator的语法糖，即在内部包装了Generator和自动执行器，Async函数返回一个promise；
   2. 内部执行器：async function a () {} => function a (args) {return spawn(function\* () {})}

function spawn(genF) {

return new Promise(function (resolve, reject) {

var gen = genF();

function step(nextF){

try{

var next = nextF();

} catch(err) {

return reject(e);

}

if (next.done) {

return resolve(nex.value);

}

Promise.resolve(next.value).then(function (v) {

step(function () { return gen.next(v); });

}, function (err) {

step(function () { return gen.throw(err)});

});

}

step(function () { return gen.next()})

})

}

* 1. 优化：
     1. 错误捕捉一般使用try/catch进行，由于await后的表达式返回的promise，可以使用promise的catch方法捕捉错误
     2. 执行性能，若需要执行a\b\c\d四个表达式，同时a和c之间不存在依赖关系，可以并行执行时，可以使用如下方法：
        1. var afn = a(); var cfn = c(); await afn; await cfn; 此时两个方法是并行执行的，只是c依赖a 的值返回
        2. 使用Promise.all()来进行并行执行

1. 查看Async&Awai源码实现深入理解
2. 生成器generator和迭代器iterator
3. Proxy

## V8内存管理

1. JavaScript中的垃圾收集
2. JavaScrpt中的内存管理
3. V8垃圾回收机制
4. 引用记数
5. 标记清除
6. 标记整理和增量标记

# 阶段二、React全家桶

## 基础篇

1. 模块化和组件化
2. React的属性、生命周期
3. 校验和参数传递
4. React的状态和双向数据绑定
5. React中的单向数据流
6. 受控组件和非受控组件
7. React表单处理
8. React以及react-dom源码阅读
9. Fiber

## 进阶篇

1. create-react-app原理分析
2. JSX原理和虚拟DOM原理
3. setState原理实现
4. React中的事务处理
5. 使用Immutablejs优化项目性能
6. React生命周期和动画原理
7. React错误处理和Hooks
8. React性能优化(react\_perf、pureComponent)
9. React新的Context Api
10. fragments React中的高阶组件
11. React新一代的Fiber架构
12. 手写一款React框架

## 路由和Redux篇

1. 路由配置
2. 路由懒加载
3. 路由重定向
4. 路由之权限管理
5. 受保护的路由
6. React-router-dom
7. 手写一个完整的React-router4路由库
8. Redux核心概念
9. Action/Reducer/Store
10. Redux数据流
11. react-redux连接React组件
12. redux仓库
13. redux-logger 打印日志
14. redux-promise 中间件执行异步任务
15. redux-thunk 中间件执行异步任务
16. redux-saga 中间件执行副作用
17. 手写Redux、react-redux、redux-logger、redux-promise、redux-thunk、redux-saga类库
18. 浏览器刷新redux中保存的数据丢失的问题：redux-persist

## SSR和mobx篇

1. 客户端渲染VS服务端渲染
2. React中的服务端渲染
3. 同构的原理和意义
4. SSR中使用路由
5. SSR中使用Redux
6. observable | computed | autorun | when | reaction
7. mobx实战
8. 手写一个mobx类库
9. Axios原理 以及源码深入理解

## TypeScript（https://www.tslang.cn/docs/handbook/tsconfig-json.html）

1. Typescript能做什么
2. Typescript基本语法
3. Typescript如何配置声明文件
4. Typescript如何做项目配置

## Rxjs

1. Rxjs能做什么
2. Rxjs的基本语法
3. Rxjs的原理、机制以及设计模式
4. Rxjs中的可观察者Observable
5. Rxjs中的观察者
6. Rxjs中的Operators
7. Rxjs中的Subscription
8. Rxjs中的Subjects
9. 阅读Rxjs深入理解

## 组件库相关篇

1. Ant-Design和AntDesignPro篇
2. 布局组件、表单和复杂表格组件
3. 让create-react-app可配的命令行工具roadhog
4. 可插拔的企业级react应用框架umi
5. 基于redux和redux-saga的数据流方案dva
6. 开箱即用的中台前端/设计解决方案AntDesignPro
7. 手写dva

# 阶段三、微信小程序&微信公众号&ReactNative

## 微信小程序

1. 微信小程序
2. 页面结构构建
3. 页面样式定义
4. 基础组件
5. 后端开发接口
6. 支付功能
7. 小程序版珠峰课堂

## 微信公众号

1. 微信API
2. 消息中间件
3. 微信公众号内的图片、音视频、图文、Ticket管理及三方接口调用
4. 客户端
5. 服务端
6. 微信三方的请求转发交互

## ReactNative

1. UIExplorer项目
2. css盒子模型和样式
3. flexbox布局
4. ReactNative长度单位
5. css元素的浮动
6. RN事件
7. React动画原理
8. 实现一个Navigator
9. App架构之目录结构、路由和组件
10. App架构之网络和Container
11. App架构之命名空间
12. ReactNative第三方插件
13. 珠峰课堂项目实战

# 阶段四、NODE.JS

## 核心模块

1. EventLoop和事件队列
2. process全局对象
3. events事件处理模块
4. commonjs原理解析
5. 深入字符编码
6. Buffer对象
7. fs文件模块
8. 压缩与解压缩
9. 加密和签名算法
10. stream流的原理和应用
11. 多进程与集群
12. tcp和http服务
13. 多语言、防盗链、正向和反向代理服务器

## Express&koa

1. 路由配置
2. 处理参数
3. 使用中间件
4. 模板
5. 静态文件服务器
6. 重定向
7. cookie和session
8. 手写express框架
9. 手写koa框架

## Egg.js

1. 内置对象
2. 中间件
3. 路由
4. 控制器
5. 服务 单元测试
6. 应用部署
7. 多进程模型和进程间通信
8. 安全和国际化

## 单元测试&集群和负载均衡

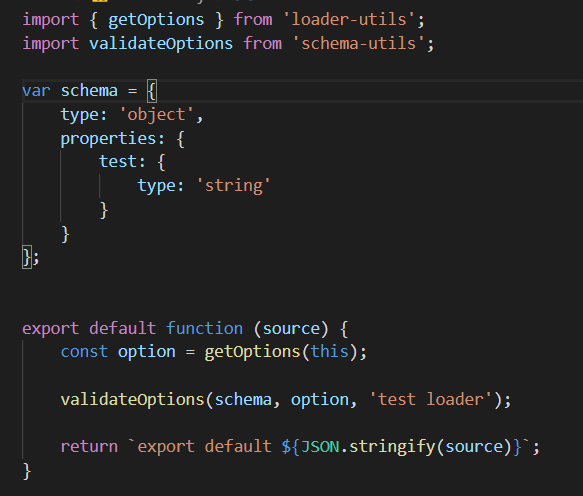
1. 测试用例和需求分析
2. 单元测试框架mocha
3. 爬虫利器
4. Puppeteer 实战
5. Jest+Enzyme实现React单元测试
6. cluster的工作原理和负载均衡算法
7. pm2使用

## 服务器部署

1. 购买自己的域名
2. 域名备案
3. 购买服务器配置
4. 服务器应用环境安装
5. 配置服务器
6. 项目远程部署
7. 项目发布与更新

# 阶段五、前端工程化和性能优化

## Webpack

1. webpack4实战
2. webpack4优化(dll、resolve、模块热替换、压缩、代码分割、可视化工具)
3. Webpack4源码分析
4. 懒加载原理、热更新原理
5. 编写自定义Loader(style-loader css-loader less-loader等)
   1. loader可以链式调用，调用顺序是从右至左，从下至上，返回的是处理后的源内容字符串或者Buffer
   2. 最后一个调用的loader返回的是资源处理后的javascript module和一个optional source map
   3. 示例：
      1. 
6. 编写自定义Plugin(html-webpack-plugin)
   1. 创建步骤：
      1. 声明一个javascript function 或者 一个 javascript class
      2. 在function 或者 class的prototype上定义一个apply方法
      3. 指定一个event hook 介入
      4. 操作webpack内部的实例细化数据
      5. 在以上操作彻底完成之后，调用webpack provided callback
      6. 注意：appl()方法只在webpack 编译器加载这个plugin时执行一次
      7. 示例：
         1. 
   2. Async event hook(异步事件钩子)：
      1. tapAsync(pluginname, (compilation, callback) => {})：调用此事件钩子时我们需要为此函数的第二个参数传入callback参数，并在执行完时调用此callback
      2. tapPromise(pluginname, (compilation) => {})：调用此事件钩子时我们需要返回一个resolve 的promise
   3. 异步钩子（Asyncchronous hooks）:
      1. Async Series hook:
         1. 调用AsyncSeriesHook([params])
         2. 使用tap/tapAsync/tapPromise方法介入
         3. 使用callAsync(...params)方法
   4. 同步钩子（Synchronous Hooks）：
      1. Bail Hook：
         1. SyncBailHook([parms])
         2. tap()
         3. call(...params)
7. Webpack的事件机制tapable和AST抽象语法树
8. 如何编写Babel插件
9. 手写一款自己的Webpack
10. 在webapck中手动通过node添加全局环境变量时需要使用cross-env这个插件，在webpack配置中结合webpack.DefinePlugin()这个插件来进行全局变量赋值。

如：在package.json中cross-env NODE\_ENV=dev webpack-dev-server

在webpack配置中new webpack.DefinePlugin({

\_\_DEV\_\_:JSON.stringify(process.env.NODE\_ENV)

})

注意：url-loader是file-loader的扩展，在url-loader的options配置参数对象中指定fallback:’file-loader’之后，可以在options配置对象中添加file-loader的配置参数

## 服务器部署

## 前端性能优化

1. webpack优化方案
2. 浏览器缓存原理和最佳设置策略
   1. 强缓存，即本地缓存
   2. 协商缓存，即304缓存
3. CDN缓存优化
4. EventLoop异步更新
5. 避免回流和重绘
6. 节流和防抖
7. 通过Performance控制性能
8. TCP网络调优

## 浏览器渲染原理

1. 浏览器内核
2. DOM树
3. CSS规则树
4. 渲染树
5. 回流和重绘

## 网络安全

1. Web漏洞的分析和防御
2. XSS（跨站脚本攻击）
3. CSRF（跨站请求伪造）
4. 防SQL注入
5. HTTPS加密

前端模块化

1. Commonjs
   1. Nodejs使用的模块化规范
   2. 规定一个文件为一个模块
   3. module.exports/exports 导出模块，require()方法来引用模块
   4. 同步执行
2. AMD(Async Module Definition)
   1. RequireJS使用的模块化规范
   2. 使用define(模块名(可省略名称)，[依赖的模块]，function(依赖模块参数){/\*执行的代码\*/})方法定义模块
   3. 使用require()引用模块
   4. 依赖前置，提前执行
   5. 模块异步加载
3. CMD(Common Module Definition)
   1. SeaJS使用的模块化规范
   2. 规定一个文件为一个模块
   3. 使用difine(function(require, exports, module) {/\*执行代码\*/})方法定义模块，并在回调内部使用exports.属性导出部分模块接口或使用module.exports导出全部模块接口
   4. 使用require()方法加载模块
   5. 尽可能懒执行，会提前加载模块代码，但只在代码运行到对应位置才开始执行对应的模块代码
4. UMD(Universal Module Definition)
   1. 通用解决方案
   2. 三步操作，首先判断是不是AMD规范，其次判断是不是CommonJS规范，最后都不是则直接定义为全局变量来使用
5. ES6 Module(ECMA Script2015 Module)
   1. 规定一个文件为一个模块
   2. 使用export/export default导出模块，使用import引用模块
   3. export (\* | {}) from ‘’，即导入并导出对应的模块或接口

# 阶段六、计算机网络&服务器和运维

## 七层网络协议

1. TCP/IP协议族的网络模型
2. IP协议以及简单路由机制
3. TCP连接的本质
4. TCP三次握手和四次挥手
5. TCP协议之滑动窗口和Nagle算法
6. HTTP、HTTPS、HTTP2的区别和联系
7. 常用网络命令
8. 网络调优实战

## Linux

## Nginx

1. nginx的安装和使用
2. 模块和基本配置
3. 正向反向代理等应用场景
4. CDN
5. 浏览器缓存
6. 跨域
7. 防盗链

## Docker

1. 虚拟机
2. Linux容器
3. Docker核心概念
4. Docker架构
5. Docker镜像
6. Docker容器
7. Dockerfile
8. Docker数据盘
9. 网络配置
10. docker-compose

# 阶段七、数据库

## Mongodb

1. Mongodb安装和使用
2. Mongodb的系统架构
3. Mongodb高级查询
4. Mongodb索引
5. Mongodb安全与权限
6. Mongodb模块之Scheme
7. Mongodb模块之Model

## mysql

1. MYSQL安装和使用
2. MYSQL系统架构
3. 数据处理之增删改查
4. 数据类型和约束分页
5. 索引和慢查询性能分析
6. 数据库安全之防止SQL注入
7. 数据库设计ER图设计
8. 数据库事务
9. 数据库设计之三大范式
10. 基于角色的权限访问控制（Role-Based Access Control）

## redis

1. redis安装和使用
2. 5种数据结构及使用场景
3. API的理解和使用
4. Redis客户端

# 阶段八、项目实践

## Egg.js+AntDesignPro+MySQL 开发企业级CMS系

## React+Mongodb+websocket 开发多人聊天室

## cheerio+mysql+nodemailer 开发个性化新闻爬虫

## React+Redux开发珠峰课堂使用canvas开发flappy bird

## 使用three.js开发微信小游戏跳一跳

# 阶段九、前端设计模式

## 创建型设计模式

1. 工厂模式
2. 建造者模式
3. 原型模式
4. 单例模式

## 结构型设计模式

1. 外观模式
2. 适配器模式
3. 代理模式
4. 装饰者模式
5. 组合模式

## 行为型设计模式

1. 观察者模式
2. 状态模式
3. 策略模式
4. 命令模式
5. 访问者模式
6. 中介者模式
7. 迭代器模式

# 阶段十、前端数据结构和算法

## 算法的基础知识

1. 输入、输出、和数量级
2. 计算能力的变革
3. CPU、寄存器和内存
4. 二分查找
5. 插入排序
6. 冒泡排序

## 算法的衡量和优化

1. 时间复杂度和空间复杂度
2. 复杂度的本质
3. 合并排序（分治策略）
4. 递归函数复杂度分析（树状图）
5. 递归表达式分析法
6. 递归数学归纳法分析
7. 主定理 \*选学

## 排序算法

1. 排序算法介绍
2. 基于比较的排序算法
3. 快速排序复杂度和优化
4. 合并排序的优化
5. 基数排序
6. 计数排序
7. 桶排序
8. 外部排序

## 递归

1. 递归的基本概念
2. 递归图形的绘制
3. 递归和穷举问题
4. 组合问题
5. 递归空间优化
6. 回溯算法
7. 重复子问题优化
8. 尾递归
9. 搜索问题
10. 深度优先搜索和广度优先搜索

## 数据结构

1. 数组
2. 双向链表
3. 反转单向链表
4. 堆
5. 栈
6. 队列

# 阶段十一、Python

## 基础知识

1、语法基础

2、字符串解析

3、时间、日期和日历

4、文件操作

6、面向对象

7、设计模式

8、排序算法

9、异常处理

10、内置模块

任务：进行小游戏开发实战训练（飞机大战、）

## 高级语言知识

1、python高级编程

2、正则表达式

3、数据结构和算法

4、linux系统编程

5、linux网络编程

## 爬虫

1、工作流程

2、http请求及响应

3、fiddler的使用

4、urllib库的使用

5、requests库的使用

6、选择器的使用（BeautifulSoup、lxml）

7、scrapy框架

8、IP代理池的实现