前端面试问题

1、一个 tcp 连接能发几个 http 请求？

如果是HTTP 1.0 版本协议，一般情况下，不支持长连接，因此在每次请求发送完毕之后，TCP连接即会断开，因此一个TCP发送一个HTTP请求，但是有一种情况可以将一条TCP连接保持在活跃状态，那就是通过`Connection`和`Keep-Alive`首部，在请求头带上`Connection: Keep-Alive` ，并且可以通过`Keep-Alive` 通用首部中指定的，用逗号分隔的选项调节keep-alive的行为，如果客户端和服务端都支持，那么其实也可以发送多条，不过此方式也有限制，可以关注《HTTP 权威指南》4.5.5 节对于Keep-Alive连接的限制和规则；

而如果是HTTP 1.1 版本协议，支持了长连接，因此只要TCP连接不断开，便可以一直发送HTTP请求，持续不断，没有上限；

同样，如果是HTTP 2.0 版本协议，支持多用复用，一个TCP连接是可以并发多个HTTP请求的，同样也是支持长连接，因此只要不断开TCP的连接，HTTP请求数也是可以没有上限地持续发送，

简单来说：

HTTP/1.0 :一个tcp连接只能发一个http请求。每次服务端返回响应后TCP连接会断开。

HTTP/1.1: 默认开启Connection: keep-alive，一个TCP连接可以发多个http请求，但是多个请求是串行执行。（注意区别TCP的keep-alive）

HTTP/2: 引入了多路复用技术 和二进制分帧，同个域名下的请求只需要占用一个 TCP 连接，请求和响应是并行执行。

2、Virtual Dom 的优势在哪里？

虚拟Dom的优势在哪里？

Virtual Dom 的优势」其实这道题目面试官更想听到的答案不是上来就说「直接操作/频繁操作 DOM 的性能差」，如果 DOM 操作的性能如此不堪，那么 jQuery 也不至于活到今天。所以面试官更想听到 VDOM 想解决的问题以及为什么频繁的 DOM 操作会性能差。

首先我们需要知道：

DOM 引擎、JS 引擎 相互独立，但又工作在同一线程（主线程）JS 代码调用 DOM API 必须 挂起 JS 引擎、转换传入参数数据、激活 DOM 引擎，DOM 重绘后再转换可能有的返回值，最后激活 JS 引擎并继续执行若有频繁的 DOM API 调用，且浏览器厂商不做“批量处理”优化， 引擎间切换的单位代价将迅速积累若其中有强制重绘的 DOM API 调用，重新计算布局、重新绘制图像会引起更大的性能消耗。

其次是 VDOM 和真实 DOM 的区别和优化：

1. 虚拟 DOM 不会立马进行排版与重绘操作
2. 虚拟 DOM 进行频繁修改，然后一次性比较并修改真实 DOM 中需要改的部分，最后在真实 DOM 中进行排版与重绘，减少过多DOM节点排版与重绘损耗
3. 虚拟 DOM 有效降低大面积真实 DOM 的重绘与排版，因为最终与真实 DOM 比较差异，可以只渲染局部

3、函数形参与实参

```
function test (a, b) {
  arguments[2] = 3;
  console.log(arguments.length); // -> 2
  console.log(arguments[2]); // -> 3
}

test(1, 2);
console.log(test.length); // -> 2
```

答案是 2 3 2 --关键知识点 argumesnts --类数组

原因：argumesnts是是一个类数组，并不是一个数组，但除了length属性和索引元素之外没有任何`Array`属性，实际上arguments是一个对象来的

因此`arguments[2]=3`该操作实际上是给`arguments`添加了一个键值对，键名是2，键值是3。并不会为`length`属性加一

```
Arguments(2) [1, 2, 2: 3, callee: ƒ, Symbol(Symbol.iterator): ƒ]js
//实际上的内容
```

所以第一次打印的是2 。第二次打印是直接获取的实参列表键名为2的值，所以直接打印3。第三次打印是在函数外部打印函数的`length`属性，函数的`length`属性就是函数的形参个数，所以打印2

4、['1', '2', '3'].map(parseInt)的结果是什么？

```
先说结果：
['1', NaN, NaN]
为什么不是['1', '2', '3']呢，下面开始分析
复制代码
```

- map() 方法返回一个新数组，数组中的元素为原始数组元素调用函数处理后的值。
- map() 方法按照原始数组元素顺序依次处理元素。

map(parseInt)其实是：

```
map(function(item, index){
    return parseInt(item, index);
})
```

即依次运行的是：

```
parseInt('1', 0);
parseInt('2', 1);
parseInt('3', 2);
```

parseInt的用法

- parseInt(string, radix) 函数可解析一个字符串，并返回一个整数。
- 当参数 radix 的值为 0，或没有设置该参数时，parseInt() 会根据 string 来判断数字的基数。
- radix 可选。表示要解析的数字的基数。该值介于 2 ~ 36 之间。

所以： parseInt('1', 0);//'1' parseInt('2', 1);//NaN parseInt('3', 2);//NaN，由于2进制中没有3

5、对象引用的问题

```
function changeObjProperty(o) {
    o.siteUrl = "http://www.baidu.com";
    o = new Object();
    o.siteUrl = "http://www.google.com";
}
let webSite = new Object();
changeObjProperty(webSite);
console.log(webSite.siteUrl);
```

此题咋看小问题，其实暗藏玄机。 先说答案：

```
console.log(webSite.siteUrl);//"http://www.baidu.com"
```

复盘如下：

```
function changeObjProperty(o) {
    //o是形参，对象的引用，依旧指向原地址，相当于 var o = webSite;赋值改变对象的属性
    o.siteUrl = "http://www.baidu.com";
    //变量o指向新的地址 以后的变动和旧地址无关，题目打印的是外部webSite.siteUrl
    o = new Object();
    o.siteUrl = "http://www.google.com";
}
复制代码
```

将题目改成如下：

```
function changeObjProperty(o) {
    o.siteUrl = "http://www.baidu.com";
    o = new Object();
    o.siteUrl = "http://www.google.com";
    return o;
}
let webSite = new Object();
changeObjProperty(webSite);
console.log(webSite.siteUrl);
let newSite = changeObjProperty(webSite);
console.log(newSite.siteUrl);
```

此时打印结果如下：

```
console.log(webSite.siteUrl);//"http://www.baidu.com"
console.log(newSite.siteUrl);//"http://www.google.com"
```

6、一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么？

1、浏览器会开启一个线程来处理这个请求，对 URL 分析判断如果是 http 协议就按照 Web 方式来处理;

2、调用浏览器内核中的对应方法，比如 WebView 中的 loadUrl 方法;

3、通过DNS解析获取网址的IP地址，设置 UA 等信息发出第二个GET请求;

4、进行HTTP协议会话，客户端发送报头(请求报头);

5、进入到web服务器上的 Web Server，如 Apache、Tomcat、Node.JS 等服务器;

6、进入部署好的后端应用，如 PHP、Java、JavaScript、Python 等，找到对应的请求处理;

7、处理结束回馈报头，此处如果浏览器访问过，缓存上有对应资源，会与服务器最后修改时间对比，一致则返回304;

8、浏览器开始下载html文档(响应报头，状态码200)，同时使用缓存;

9、文档树建立，根据标记请求所需指定MIME类型的文件（比如css、js）,同时设置了cookie;

10、页面开始渲染DOM，JS根据DOM API操作DOM,执行事件绑定等，页面显示完成。

7、JavaScript中的数据类型

（1）基本类型

- Number

  数值分整数类型和浮点数类型

  整数类型为十进制，也可以设置为八进制、十六进制

  浮点类型必须喊小数，可用科学计数法表示

  特殊数值：`NaN` -- 表示不是数值

- String

  字符串可用使用双引号（"）、单引号（'）或反引号（`）标示

  不可改变---若修改，只能先销毁再创建

- Boolean

  布尔值类型--只有两个值 `true` 和 `false`

- Undefined

  唯一值 `undefined`

  声明变量未赋值时相当于给变量赋予 `undefined`

- Null

  唯一值-- `null`

理论上代表空对象指针

- Symbol

  Symbol（符号）是原始值，且符号实例是唯一、不可变的。符号的用途是确保对象属性使用唯一标识符，不会发生属性冲突的危险

（2）引用类型

复杂类型统称为`Object`，主要讲述下面三种：

- Object

  创建`object`常用方式为对象字面量表示法，属性名可以是字符串或数值

- Array

  `JavaScript`数组是一组有序的数据，但跟其他语言不同的是，数组中每个槽位可以存储任意类型的数据。并且，数组也是动态大小的，会随着数据添加而自动增长

- Function

  函数实际上是对象，每个函数都是`Function`类型的实例，而`Function`也有属性和方法，跟其他引用类型一样

  三种常见的表达方式：

  - 函数声明

    ```
    // 函数声明
    function sum (num1, num2) {
        return num1 + num2;
    }
    ```

  - 函数表达式

    ```
    let sum = function(num1, num2) {
        return num1 + num2;
    };
    ```

  - 箭头函数

    ```
    let sum = (num1, num2) => {
        return num1 + num2;
    };
    ```

（3）存储区别

- 基本数据类型存储在栈中
- 引用类型的对象存储于堆中

小结：

- 声明变量时不同的内存地址分配：
    - 简单类型的值存放在栈中，在栈中存放的是对应的值
    - 引用类型对应的值存储在堆中，在栈中存放的是指向堆内存的地址
- 不同的类型数据导致赋值变量时的不同：
    - 简单类型赋值，是生成相同的值，两个对象对应不同的地址
    - 复杂类型赋值，是将保存对象的内存地址赋值给另一个变量。也就是两个变量指向堆内存中同一个对象

8、typeof 与 instanceof 区别

（1）typeof

`typeof` 操作符返回一个字符串，表示未经计算的操作数的类型

无法判别null和引用类型（除`function`）

（2）instanceof

`instanceof` 运算符用于检测构造函数的 `prototype` 属性是否出现在某个实例对象的原型链上

```
/ 定义构建函数
let Car = function() {}
let benz = new Car()
benz instanceof Car // true
let car = new String('xxx')
car instanceof String // true
let str = 'xxx'
str instanceof String // false
```

（3）区别

`typeof`与`instanceof`都是判断数据类型的方法，区别如下：

- `typeof`会返回一个变量的基本类型，`instanceof`返回的是一个布尔值
- `instanceof` 可以准确地判断复杂引用数据类型，但是不能正确判断基础数据类型
- 而`typeof` 也存在弊端，它虽然可以判断基础数据类型（`null` 除外），但是引用数据类型中，除了`function` 类型以外，其他的也无法判断

> 通用检测数据类型，可以采用`Object.prototype.toString`，调用该方法，统一返回格式“`[object Xxx]`”的字符串

```
Object.prototype.toString.call({})
Object.prototype.toString.call(1)    // "[object Number]"
Object.prototype.toString.call('1')  // "[object String]"
Object.prototype.toString.call(true)  // "[object Boolean]"
Object.prototype.toString.call(function(){})  // "[object Function]"
```

```
function getType(obj){
  let type  = typeof obj;
  if (type !== "object") {    // 先进行typeof判断，如果是基础数据类型，直接返回
    return type;
  }
  // 对于typeof返回结果是object的，再进行如下的判断，正则返回结果
  return Object.prototype.toString.call(obj).replace(/^\[object (\S+)\]$/,
'$1');
}
```

9、什么是事件代理？应用场景？

（1）事件代理

事件代理，俗地来讲，就是把一个元素响应事件（`click`、`keydown`.....）的函数委托到另一个元素

事件流的都会经过三个阶段： 捕获阶段 -> 目标阶段 -> 冒泡阶段，而事件委托就是在冒泡阶段完成

事件委托，会把一个或者一组元素的事件委托到它的父层或者更外层元素上，真正绑定事件的是外层元素，而不是目标元素

（2）应用场景

```
<ul id="list">
  <li>item 1</li>
  <li>item 2</li>
  <li>item 3</li>
  ......
  <li>item n</li>
</ul>
```

```
// 获取目标元素 ---内存消耗大
const lis = document.getElementsByTagName("li")
// 循环遍历绑定事件
for (let i = 0; i < lis.length; i++) {
    lis[i].onclick = function(e){
        console.log(e.target.innerHTML)
    }
}


// 给父层元素绑定事件
document.getElementById('list').addEventListener('click', function (e) {
    // 兼容性处理
    var event = e || window.event;
    var target = event.target || event.srcElement;
    // 判断是否匹配目标元素
    if (target.nodeName.toLocaleLowerCase === 'li') {
        console.log('the content is: ', target.innerHTML);
    }
});
```

适合事件委托的事件有：`click`，`mousedown`，`mouseup`，`keydown`，`keyup`，`keypress`

从上面应用场景中，我们就可以看到使用事件委托存在两大优点：

- 减少整个页面所需的内存，提升整体性能
- 动态绑定，减少重复工作

但是使用事件委托也是存在局限性：

- `focus`、`blur`这些事件没有事件冒泡机制，所以无法进行委托绑定事件
- `mousemove`、`mouseout`这样的事件，虽然有事件冒泡，但是只能不断通过位置去计算定位，对性能消耗高，因此也是不适合于事件委托的

如果把所有事件都用事件代理，可能会出现事件误判，即本不该被触发的事件被绑定上了事件

10.全局函数有哪些？

全局函数它不属于任何一个内置对象。」

常用的功能：escape( )、eval( )、isFinite( )、isNaN( )、parseFloat( )、parseInt( )、unescape( )。

1) 、decodeURI()

参数:string

功能描述:对 encodeURI() 函数编码过的 URI 进行解码。

实例:

可把 http://www.cnblogs.com/My%20first/ 解码为 http://www.cnblogs.com/My first/

2) 、decodeURIComponent()

参数:string

功能描述:函数可对 encodeURIComponent() 函数编码的 URI 进行解码。

3) 、encodeURI()

参数:string

功能描述:可把字符串作为 URI 进行编码。

提示：如果 URI 组件中含有分隔符，比如 ? 和 #，则应当使用 encodeURIComponent() 方法分别对各组件进行编码。

4) 、encodeURIComponent()

功能描述:可把字符串作为 URI 组件进行编码。

encodeURIComponent() 与 encodeURI() 区别：

encodeURIComponent()假定它的参数是 URI 的一部分（比如协议、主机名、路径或查询字符串），因此将转义用于分隔 URI 各个部分的标点符号。

5）、escape()

参数:string

功能描述:可对字符串进行编码，这样就可以在所有的计算机上读取该字符串。

不编译参数：ASCII 字母和数字以及 ASCII 标点符号`- _ . ! ~ * ' ( )`

温馨提示：ECMAScript v3 反对使用该方法，应用使用 decodeURI() 和 decodeURIComponent() 替代它。

6）、unescape()

参数:string

功能描述:可对通过 escape() 编码的字符串进行解码。

工作原理：通过找到形式为 %xx 和 %uxxxx 的字符序列（x 表示十六进制的数字），用 Unicode 字符 \u00xx 和 \uxxxx 替换这样的字符序列进行解码。

温馨提示:ECMAScript v3 已从标准中删除了 unescape() 函数，并反对使用它，因此应该用 decodeURI() 和 decodeURIComponent() 取而代之。

7）、eval()

参数:string

功能描述:可计算某个字符串，并执行其中的的 JavaScript 代码。

实例：

document.write(eval("12+2")) 将输出 14

注意：参数必需是string类型的,否则该方法将不作任何改变地返回.

8）、isFinite()

参数: number

功能描述:用于检查其参数是否是有穷大的。如果 number 是有限数字（或可转换为有限数字），

那么返回 true。否则，如果 number 是 NaN（非数字），或者是正、负无穷大的数，则返回 false。

实例:

isFinite(-125)和isFinite(1.2)返回true,

而isFinite('易水寒')和isFinite('2011-3-11')返回false.

9）、isNaN()

参数:无限制

功能描述:函数用于检查其参数是否是非数字值。

10）、Number()

参数:无限制

功能描述:把对象的值转换为数字。

如果参数是 Date 对象，Number() 返回从 1970 年 1 月 1 日至今的毫秒数。

如果对象的值无法转换为数字，那么 Number() 函数返回 NaN。