**Web前端1**

1. 怎么理解 web 语义化；

根据内容的结构化（内容语义化），选择合适的标签（代码语义化）便于开发者阅读和写出更优雅的代码的同时让浏览器的爬虫和机器很好地解析。

* 为了在没有CSS的情况下，页面也能呈现出很好地内容结构、代码结构:为了裸奔时好看；
* 有利于[SEO](http://baike.baidu.com/view/1047.htm" \t "https://www.cnblogs.com/freeyiyi1993/p/_blank)：和搜索引擎建立良好沟通，有助于爬虫抓取更多的有效信息：[爬虫](http://baike.baidu.com/view/998403.htm" \t "https://www.cnblogs.com/freeyiyi1993/p/_blank)依赖于标签来确定上下文和各个关键字的权重；
* 方便其他设备解析（如屏幕阅读器、盲人阅读器、移动设备）以意义的方式来渲染网页；
* 便于团队开发和维护，语义化更具可读性，是下一步吧网页的重要动向，遵循W3C标准的团队都遵循这个标准，可以减少差异化。

<header><main><footer><article><section>

1. 如果去优化 html 怎么做；

**减少HTTP请求。**

**缓存。**

**压缩HTML、CSS、JavaScript**。

**优化首屏加载**。

**预加载**。

**按需加载**。

**压缩图片**。

#### 预先解析DNS

非常简单，效果立竿见影，加快页面加载时间，多用于预解析CDN的地址的DNS

<!--在head标签中，越早越好--><link rel="dns-prefetch" href="//example.com">

#### Preload

浏览器会在遇到如下link标签时，立刻开始下载main.js(不阻塞parser)，并放在内存中，但不会执行其中的JS语句。  
只有当遇到script标签加载的也是main.js的时候，浏览器才会直接将预先加载的JS执行掉。

<link rel="preload" href="/main.js" **as**="script">

#### Prefetch

浏览器会在空闲的时候，下载main.js, 并缓存到disk。当有页面使用的时候，直接从disk缓存中读取。其实就是把决定是否和什么时间加载这个资源的决定权交给浏览器。

如果prefetch还没下载完之前，浏览器发现script标签也引用了同样的资源，浏览器会再次发起请求，这样会严重影响性能的，加载了两次，，所以不要在当前页面马上就要用的资源上用prefetch，要用preload。

<link href="main.js" rel="prefetch">

3. css 三列布局的实现方式；

flex布局：flex:1，浮动: left、right、center，绝对定位：left:0, right:0，calc: calc(100% - left-right)

双飞翼：main先渲染，marin-left:-100% margin-rigit: width, main被包裹起来margin:with

圣杯：不需要包裹main，由外层的padding取代，其余一致，先渲染main

1. 代码质量如何保障；

变量名函数名语义化，代码逻辑清晰可读性强，关键代码注释，函数注释，格式规范用、eslint检测，库文件的changelog，commit的规范解决了什么问题、引入原因、解决方案、代码影响范围 ，提交的代码需要进行codereivew

1. html 模板的工作原理是什么样的；

编译时，利用正则找出模板内容，根据模板引擎处理模板内容，返回处理结果并替换原来的模板内容，最后生成处理后的html

1. 手写一个 flattern 函数；

递归展开

1. JS 中的作用域、原型、构造函数、this 怎么理解，new 操作符干了什么；

全局、函数、块级

词法(编写时确定)、动态（this）（调用时确定）

this代表函数执行时的环境，指向调用函数的对象

1. JS 函数上下文；

javaScript执行代码时创建的上下文，并压栈，创建作用域链，变量对象，this

9. 算法：怎么找到一个数组中的相同元素，空间复杂度和时间复杂度是怎么样的。

Olog(n)2分逼近法

**Web前端2**

1. 介绍一下 TypeScript

js的超集，具备静态类型和基于类的面向对象编程语法。

1. EventLoop Promise Async 题目

事件池，宏队列、微队列，所有微队列在下一个宏执行前需全部执行完





3. ES6继承与ES5继承的对比，优缺点

4. 如何实现私有变量

5. 算法题 二分查找

6. HTTPS HTTP2.0

7. 浏览器渲染流程

**WEB前端3**

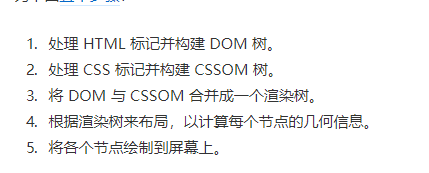
1. TCP/IP协议，HTTPS，HTTP2.0

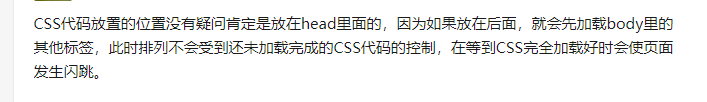
TCI/IP建立时三次握手，关闭四次握手

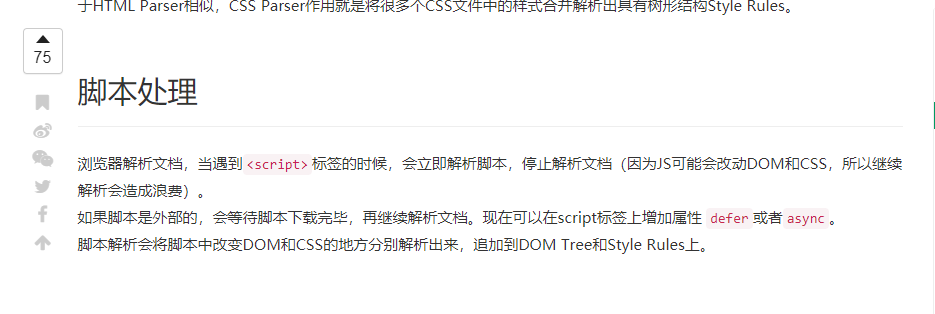
Content-length判断是否传输完毕，keep-alive：time\_out

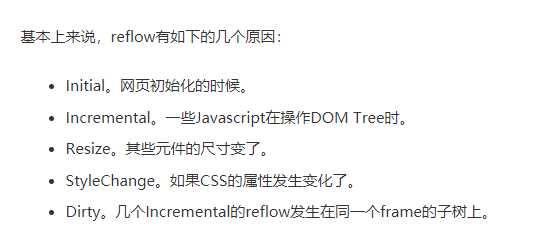
https， 服务器ca申请配置证书，客户端下载（公钥）验证通过后，使用公钥加密对称·密钥，发送使用对称密钥加密的信息，服务器利用私钥解密密钥利用密钥解密和加密信息

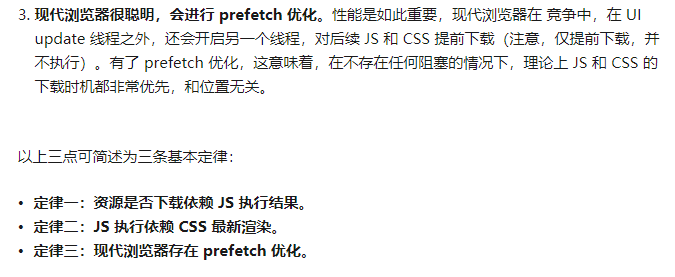
1. 为什么link标签要放在head里面，script标签会要放在body底部，为什么script标签的下载要阻塞ui渲染











link标签中链接的样式表会影响浏览器的绘制和布局，dom 解析到script后中断后面的解析

1. css解析规则，为什么要从右往左解析

效率更高，往往左边的选择符更泛化，右边更具化，从右往左减少大量查询次数

1. css选择器优化策略

（1）.ID选择器

（2）类选择器

（3）元素选择器

（4）兄弟选择器

（5）子选择器

（6）后代选择器

（7）属性选择器

（8）伪类/伪元素选择器

上面是常见的CSS选择器，性能从上到下越来越低。

1. .尽可能不使用通配符选择器：

关键选择器越具体越好，通配符选择器实在是最不具体的一个。

**（2）.合理避免使用id选择器：**

id选择器的性能最好，难道我们将每一个元素都添加一个id属性，肯定不现实。不过规范还是建议尽可能少的使用id选择器，这就是最佳实践与最佳性能之间的一个平衡或者取舍。

使用id确定元素在网页中的位置，应该始终考虑使用class，而不是id，除非只使用一次。

**（3）.减少后代选择器的使用：**

尽可能避免使用后代选择器，最好使用子选择器替代。

div>a的性能肯定要好于div a。

首先使用a选择器匹配页面所有链接<a>元素，后代选择器要一层层查找最终确定当前a是否具有div父元素，但是子元素选择器只要查找一层就可以了，性能可以优化很多。

**（4）.尽可能使用继承：**

#nav {}

#nav > .span { **font-size**:24px; }

#nav > .a { **font-size**:24px; }

上述代码可以优化如下：

#nav {**font-size**:24px;}

使用继承，而不是每一个选择器都设置一次。

1. 实现多列布局的方式
2. js基本数据类型，null和undefined的意义

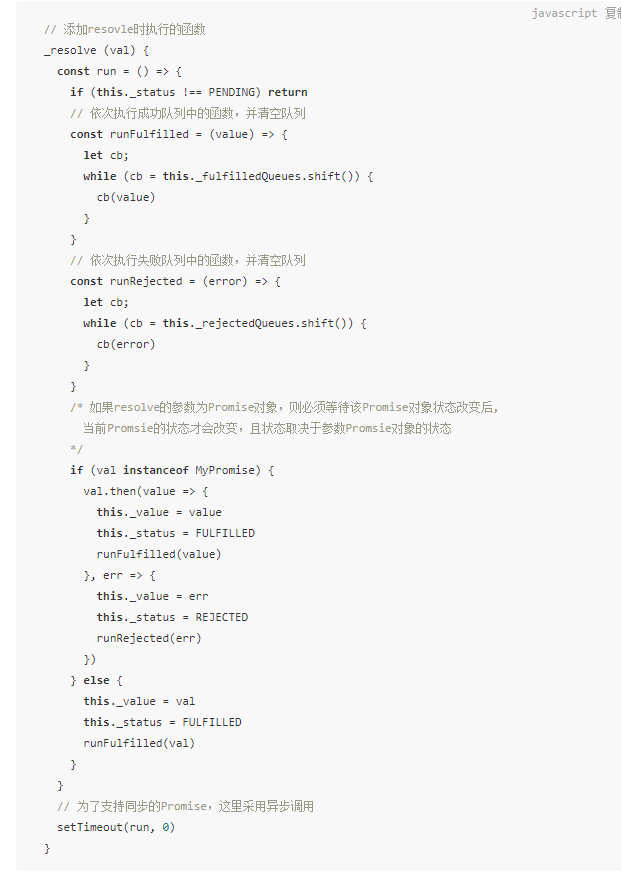
null表示变量赋为空值，undefined表示未赋值

1. js的异步有哪些实现方案，然后给了代码题考察promise、async|await

promise是一条热流，创建时就会执行内部代码，状态分为：pending、fullfilled、rejected，属性有reslove、reject、all、then、catch，

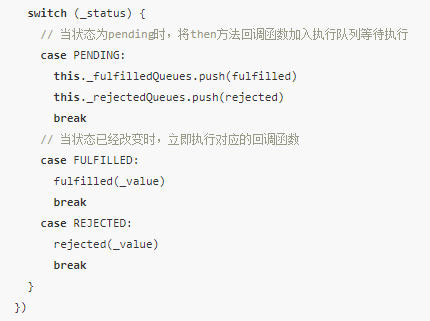
Then、all会返回一个新的promise并使其符合链式调用,then会传入onfullfiiled和onrejected，pending时进行注册回调函数队列，onfullfiiled或时onrejected直接执行，promise的参数有reslove和reject，resolve执行时将Promise对象的状态从 Pending(进行中) 变为 Fulfilled(已成功),并从注册的onfullfiiled队列中取出队尾函数执行，reject将Promise对象的状态从 Pending(进行中) 变为 Rejected(已失败)并从注册的onrejected队列中取出队尾函数执行，代码实现







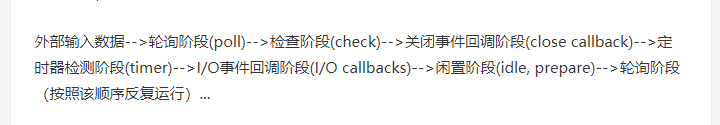




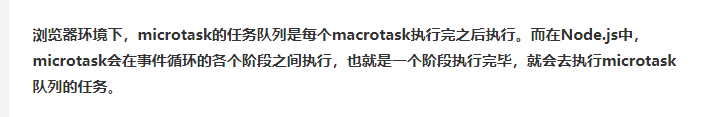
Aysnc\await yeild加promise语法糖

1. 事件循环的理解，浏览器跟node的事件循环差别

Node:

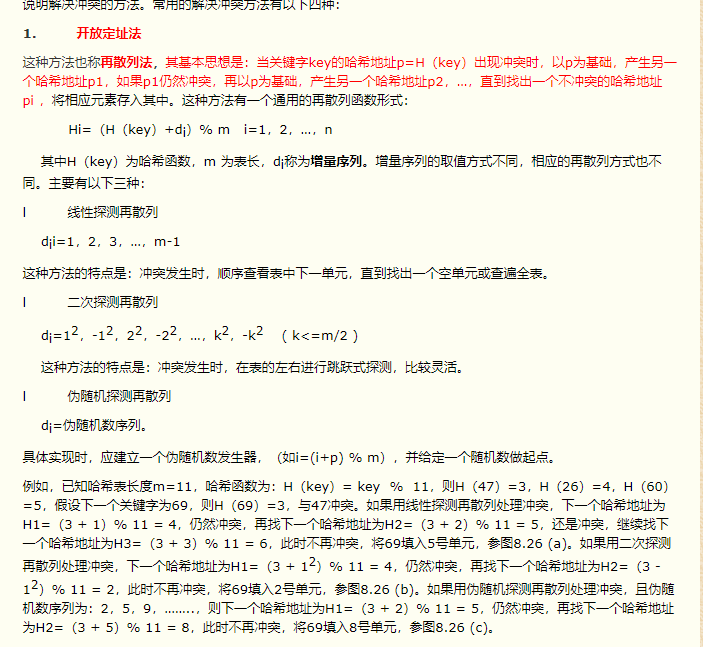


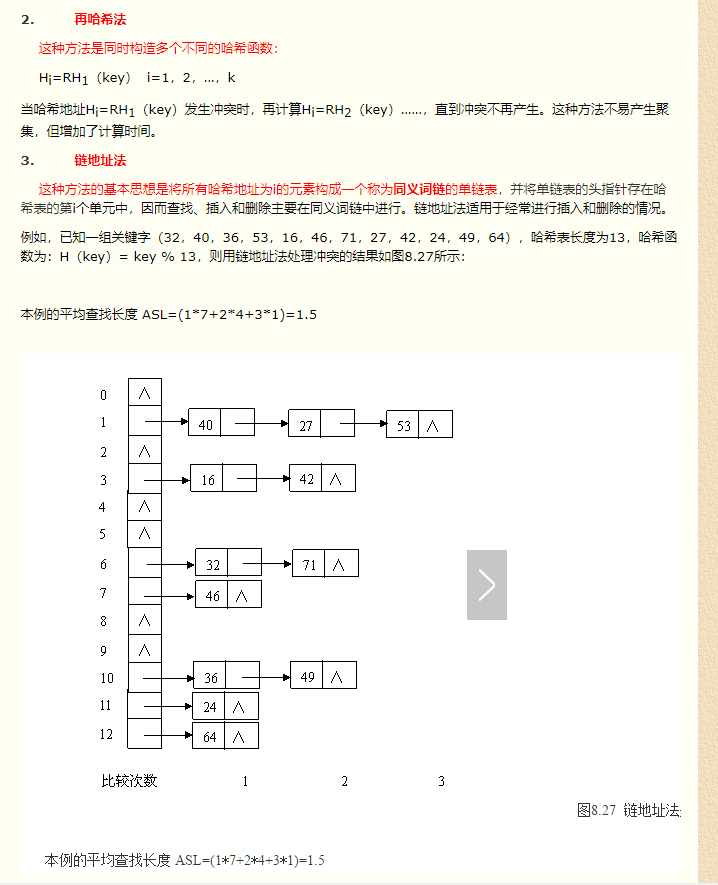




1. 数据结构与算法，哈希表怎么解决冲突

Hash：取余法，平方取中，**伪随机数法 解决冲突：开放定址：**线性探测再散列， 二次探测再散列了，随机数序列。





1. 常用的排序算法和时间复杂度，冒泡、归并、快排，为什么平时快排用得多一些，快排的优化方式
2. 函数执行上下文理解+代码题
3. 常用的设计模式
4. js面向对象的理解，es6、es5继承的方式
5. 写代码：二分查找
6. 写代码：正则匹配电话号码

**WEB前端4**

1. https如何保证数据不被篡改，密匙采用什么算法；MAC算法、SHA算法

如果证书（公钥、权威机构的信息、服务器域名和经过CA私钥签名之后的证书摘要,证书的计算方法以及证书对应的域名）被拦截并伪造，客户端在收到证书时会用ca公钥解密摘要,再根据证书上描述的计算证书的方法计算一下当前证书的信息摘要，对比两者是否一致，若一致则为服务器所发。

2.使用udp如何保证数据完整性，不丢包，如何控制传输速率；

3.vue中的key作用，vue-hooks， react-hooks；

key在v-for里面作为列的唯一标识减少数据变化重渲成本，同时列表节点变换算法也需要key提高性能，Vue-hooks，重用逻辑，不按生命周期周期

1. get，post之外还有哪些请求；强缓存如何实现；

put、delete、option，exprie:0，利用hash更新。

5.leetcode峰值问题；

6.async／await问题；

7.事件循环／setTimeout不精确的原因;

setTimeout自己内部有执行时间，被其他异步任务延后，间隔时间减去自己的执行时间

**WEB前端5**

1.代码题：zhongbiao给定24小时内任意两个时间，例如：开始时间00:00:00，结束时间：18:30:30，，时针，分针，秒针在钟表上每走一圈的角度是360度，计算两个时间段内时针，分针，秒针分别在钟表上走过的角度

2.代码题：字符串中含有左右括号， 写出函数判断字符串是否合法，合法的条件是左右括号都匹配，合法返回true否则返回false。例如： ：输入 ((a+b)\*(a+b))\*(a+b) 返回true，输入 ((a+b))))返回false。

3.回答以下输出内容，并说明原因：

(function(){

var a = b = 5;

})();

console.log(a);

console.log(b);

4.描述HTTP三次握手过程，HTTPS握手过程。

5.HTTP请求方式除了GET，POST以外还有哪些，分别是什么意思？。

6.具体说明display属性值与表现，positon:fixed在什么情况会失效以及为什么？

7.垂直居中实现方法有哪些？

8.vue双向绑定原理，简述具体源码实现

defineProperty(data, {

Get() {},

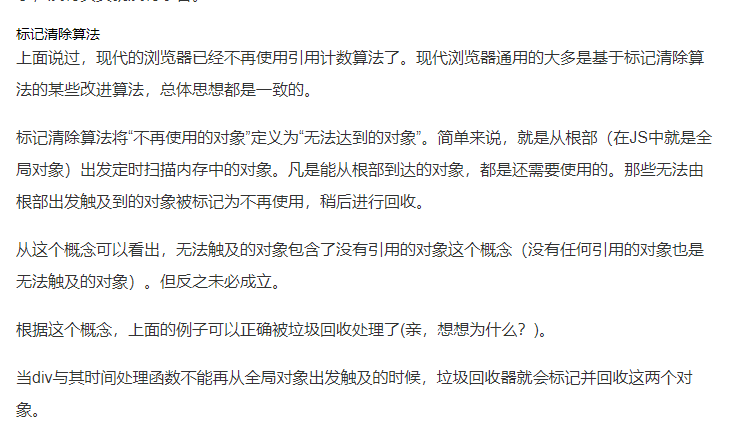
Set() {}

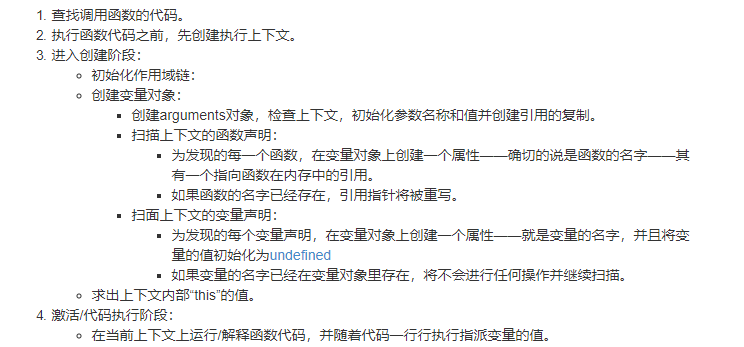
})

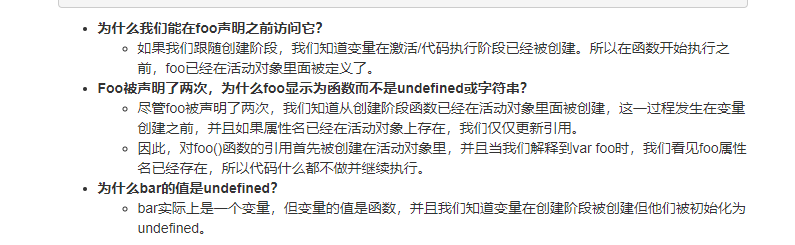
1. JS继承实现

js内存机制：

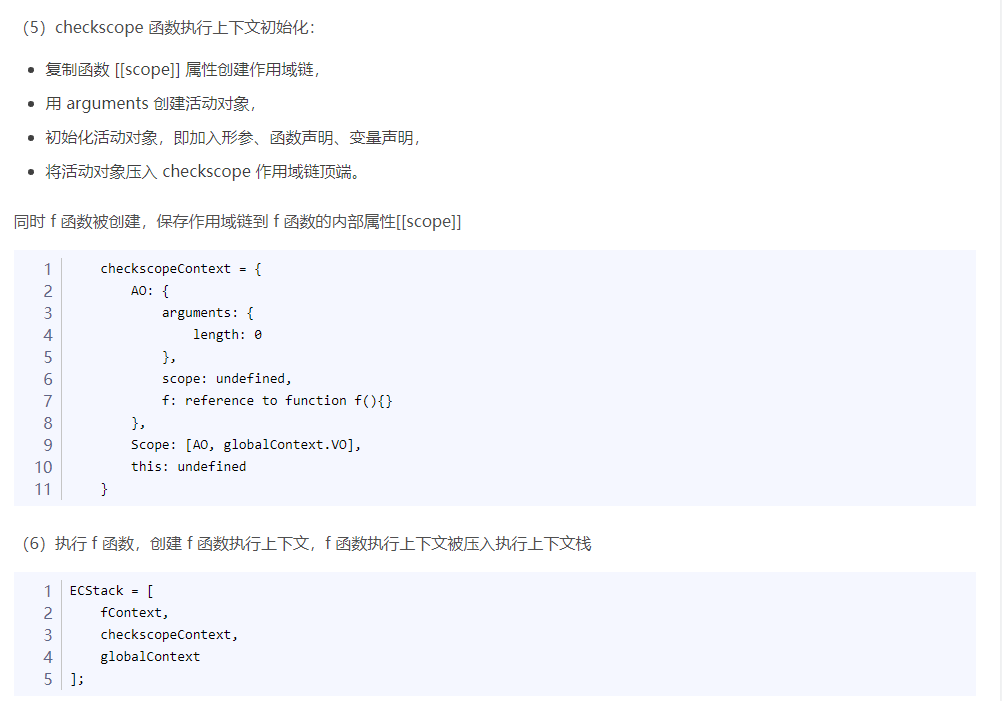






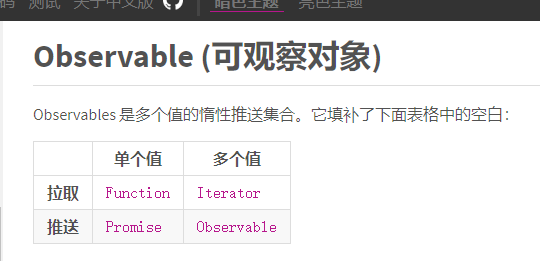






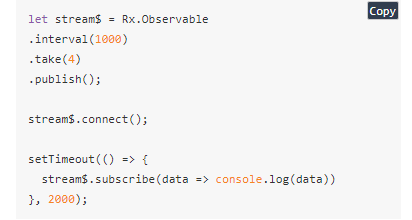
rxjs相关：





Observable 有冷热两种类型。我们先来看看什么是冷的 observable 。如果是冷的 observable 的话，那么两个订阅者得到值是两份完全相同的副本,如果是热的 observable 的话，订阅者只能收到当它开始订阅后的值，这很像是足球比赛的实况直播，如果你在开场5分钟后才开始观看，你会错失开场前5分钟的一切，从观看的这一刻起你才开始接收数据：

需要两个部件来将冷的 observable 转变成热的， publish() 和 connect() 。



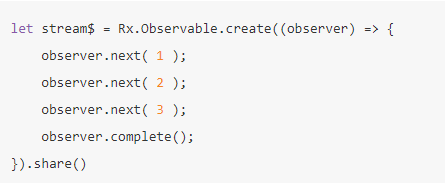
## **暖的 Observables**

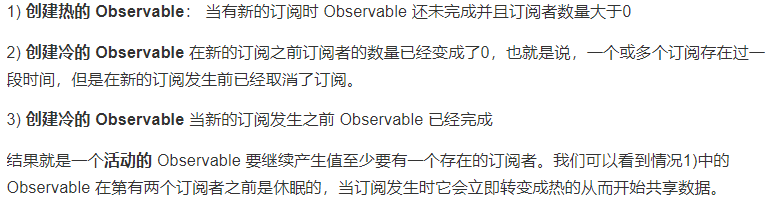
这是 observalbes 的另外一种类型，它的表现很像热的 observable ，但它在某种程度上是惰性的。我想表达的是从本质上来说，在有订阅发生之前它们不会发出任何值。



## **天生的热 observables**

使用 share() 操作符将冷的 observables 转变成热的结果。





mergeAll

