回到顶部



## Generator 函数

Generator函数是ES6提供的一种异步编程解决方案，语法行为与传统函数完全不同

Generator函数有多种理解角度： 语法上，Generator函数是一个状态机，封装了多个内部状态。执行Generator函数会返回一个遍历器对象，可以依次遍历Generator函数内部的每一个状态。

形式上，Generator函数是一个普通函数，但是有两个特征。一是，function关键字与函数名之间有一个星号；二是，函数体内部使用yield表达式，定义不同的内部状态。

#### yield语句

Generator函数返回的遍历器对象，yield语句暂停，调用next方法恢复执行，如果没遇到新的yeild，一直运行到return语句为止，return 后面表达式的值作为返回对象的value值，如果没有return语句，一直运行到结束，返回对象的value为undefined.


- 回到顶部


## async与await

async函数就是Generator函数的语法糖。 async函数就是将Generator函数的星号（*）替换成async，将yield替换成await，仅此而已。 进一步说，async函数完全可以看作多个异步操作，包装成的一个Promise对象，而await命令就是内部then命令的语法糖。

async函数返回一个Promise对象，可以使用then方法添加回调函数。当函数执行的时候，一旦遇到await就会先返回，等到异步操作完成，再接着执行函数体内后面的语句。


- 回到顶部


## js执行机制/eventLoop

js是单线程，同一个时间只能做一件事，执行顺序：自上之下执行。所以，实现异步通过eventLoop。

**机制eventLoop**

事件循环机制。js引擎在一次事件循环中，会先执行js线程的主任务，然后会去查找是否有微任务 microtask（promise），如果有那就优先执行微任务，如果没有，在去查找宏任务 macrotask（setTimeout、setInterval）进行执行

**浏览器 eventLoop 和 node eventLoop**
js 是单线程的，但是 ajax 和 setTimeout 在浏览器里面会多开一个县城

**宏任务：**setTimeout setInterval setImmediate(ie 下 生效) MessageChannel(消息通道)
**微任务：**Promise MutationObserver (监听 dom 节点更新完毕) process.nextTick()

小结:代码从上到下执行，会先执行同步的代码，再执行微任务，等到宏任务有没有到时间，时间到了的宏任务放到宏任务队列，微任务执行完毕之后，会从宏任务队列中取出一个宏任务会放到当前的浏览器的执行环境中执行，当前执行环境都执行完毕后，会先去清空微任务。

**node.js 中对于事件环的解释**

- timers： 定时器 setTimeout 和 seInterval 的执行，将 callback 加入队列中。
- pending callbacks： 一些 I/O 的 callback，推迟到下一次循环中执行。idle, prepare： 内部的一些事件。
- poll： 轮循，i/o，回调，fs.readFile()。
- check： setImmediate 的 callback 执行。
- close callbacks： 一，些关闭的回调函数，如 socket。

> 其实我们需要关心的就是`timers`、`poll`、`check`这三个阶段。下面我们来说一下它们的执行顺序。默认，会从上到下依次执行，如果代码执行到`poll`后，发现`check`阶段没有，那就在`poll`在等待，等待`times`时间到达后，再清空代码。只到队列发生切换时，就会执行微任务。`poll`的下一个阶段就是`check`，如果`check`队列中有东西的，会先执行`check`。


- 回到顶部


## 浏览器渲染机制

解析 HTML 和 css 将其转成 js 语言进行渲染 dom 树，使用流式布局渲染成浏览器文档。

- 构建 DOM 树（parse）：渲染引擎解析 HTML 文档，首先将标签转换成 DOM 树中的 DOM node
- 构建渲染树（construct）：解析对应的 CSS 样式文件信息
- 布局渲染树（reflow/layout）：从根节点递归调用，计算每一个元素的大小、位置等，给出每个节点所应该在屏幕上出现的精确坐标；
- 绘制渲染树（paint/repaint）：遍历渲染树，使用 UI 后端层来绘制每个节点。
- 回到顶部

## 重绘和回流(重排)的区别

回流(重排)：重新渲染页面
重绘：改变页面样式的时候，不会影响其他元素在页面的配置。
触发回流(重排)：

- 页面渲染初始化(无法避免)
- 添加或删除可见的 DOM 元素
- 元素位置的改变，或者使用动画
- 元素尺寸的改变——大小，外边距，边框
- 浏览器窗口尺寸的变化
- 填充内容的改变，比如文本的改变或图片大小改变而引起的计算值宽度和高度的改变

重排必定会引发重绘，但重绘不一定会引发重排。

- 回到顶部

## 缓存的理解/http缓存机制

缓存分为强缓存和协商缓存。强缓存不过服务器，协商缓存需要过服务器，协商缓存返回的状态码是304。两类缓存机制可以同时存在，强缓存的优先级高于协商缓存。当执行强缓存时，如若缓存命中，则直接使用缓存数据库中的数据，不再进行缓存协商.

- 强缓存 所请求的数据在缓存数据库中尚未过期时，不与服务器进行交互，直接使用缓存数据库中的数据。 Expires
  cache-control
- 协商缓存
  　　当强缓存过期未命中或者响应报文 Cache-Control 中有 must-revalidate 标识必须每次请求验证资源的状态时，便使用协商缓存的方式去处理缓存文件.
  　　从缓存数据库中取出缓存的标识，然后向浏览器发送请求验证请求的数据是否已经更新，如果已更新则返回新的数据，若未更新则使用缓存数据库中的缓存数据 Last-Modified 和 If-Modified-Since Etag 和 If-None-Match

#### http缓存机制

浏览器第一次获取到资源后，然后根据返回的信息来告诉如何缓存资源，可能采用的是强缓存，也可能告诉客户端浏览器是协商缓存，这都需要根据响应的header内容来决定的。同时当相应header里没返回cache-control和expires的时候，浏览器可以根据LM-Factor算法计算出一个试探性最大使用期。

#### 有关缓存的响应头字段

304 协商缓存 200 不缓存
200 携带size字段为：from disk cache 不请求网络资源，资源缓存在硬盘中，一般是js、css等较大资源 强缓存或试探缓存
200 携带size字段为：from memory cache 不请求网络资源，资源缓存在内存中，一般是图片等较小资源，浏览器关闭后，数据也将不存在 强缓存或试探缓存

**请求头**：浏览器向服务器发送请求的数据，资源。

```
Cache-Control：max-age=300缓存的最长时间 300s
Cookie：浏览器暂存服务器发送的信息
Allow:GET 请求的方法 GET 常见的还有POST
Accept: text/html,image/*浏览器可以接收的类型
Accept-Charset: ISO-8859-1浏览器可以接收的编码类型
Accept-Encoding: gzip,compress 浏览器可以接收压缩编码类型
Accept-Language: en-us,zh-cn  浏览器可以接收的语言和国家类型
Host: www.lks.cn:80浏览器请求的主机和端口
```

**响应头**：服务器向浏览器响应数据，告诉浏览器具体操作。

```
Location: http://www.lks.cn/index.html                    控制浏览器显示哪个页面
Server:apache nginx                                       服务器的类型
Content-Encoding: gzip                                    服务器发送的压缩编码方式
Content-Length: 80                                        服务器发送显示的字节码长度
Content-Language: zh-cn                                   服务器发送内容的语言和国家名
```

```
Content-Type: image/jpeg; charset=UTF-8                    服务器发送内容的类型和编码类型
Last-Modified: Tue, 11 Jul 2000 18:23:51GMT                服务器最后一次修改的时间
Refresh: 1;url=http://www.lks.cn                           控制浏览器1秒钟后转发URL所指向的页面
Content-Disposition: attachment; filename=lks.jpg          服务器控制浏览器发下载方式打开文件
Transfer-Encoding: chunked                                 服务器分块传递数据到客户端
Set-Cookie:SS=Q0=5Lb_nQ; path=/search                      服务器发送Cookie相关的信息
Expires: -1                                                资源的过期时间，提供给浏览器缓存数据,-1永远过期
Cache-Control: no-cache                                    告诉浏览器，一定要回服务器校验，不管有没有缓存数据。
Pragma: no-cache                                           服务器控制浏览器不要缓存网页
Connection: close/Keep-AliveHTTP                           请求的版本的特点
Date: Tue, 11 Jul 2000 18:23:51 GMT                        响应网站的时间
ETag: “ihfdgkdgnp98hdfg”                                   资源实体的标识(唯一标识，类似md5值，文件有修改md5就不一样)
```

- 回到顶部

## 几种请求方法

GET、POST、HEAD、PUT、DELETE、CONNECT、OPTIONS、TRACE

### get和post请求的区别

- get 一般用于获取数据
- get请求如果需要传递参数，那么会默认将参数拼接到url的后面；然后发送给服务器；
- get请求传递参数大小是有限制的；是浏览器的地址栏有大小限制；
- get安全性较低
- get 一般会走缓存，为了防止走缓存，给url后面每次拼的参数不同；放在?后面，一般用个时间戳
- post 一般用于发送数据
- post传递参数，需要把参数放进请求体中，发送给服务器；
- post请求参数放进了请求体中，对大小没有要求；
- post安全性比较高；
- post请求不会走缓存；
- 回到顶部

## HTTP与HTTPS的区别

- HTTP 的 URL 由 http://起始且默认使用端口 80，而 HTTPS 的 URL 由 https://起始且默认使用端口 443
- HTTP 是超文本传输协议，信息是明文传输，HTTPS 则是具有安全性的 SSL 加密传输协议
- HTTP 的连接很简单，是无状态的，HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 http 协议安全
- 回到顶部

## 从输入url地址到页面相应都发生了什么?

1、浏览器的地址栏输入 URL 并按下回车.
2、浏览器查找当前 URL 是否存在缓存，并比较缓存是否过期。
3、DNS 解析 URL 对应的 IP.
4、根据 IP 建立 TCP 连接（三次握手）。

5、HTTP 发起请求。
6、服务器处理请求，浏览器接收 HTTP 响应。
7、渲染页面，构建 DOM 树。
8、关闭 TCP 连接（四次挥手）

- 回到顶部

## TCP的三次握手和四次挥手

### 三次握手

- 第一次握手：客户端发送一个 SYN 码给服务器，要求建立数据连接；
- 第二次握手： 服务器 SYN 和自己处理一个 SYN（标志）；叫 SYN+ACK（确认包）；发送给客户端，可以建立连接
- 第三次握手： 客户端再次发送 ACK 向服务器，服务器验证 ACK 没有问题，则建立起连接；

### 四次挥手

- 第一次挥手： 客户端发送 FIN(结束)报文，通知服务器数据已经传输完毕；
- 第二次挥手: 服务器接收到之后，通知客户端我收到了 SYN,发送 ACK(确认)给客户端，数据还没有传输完成
- 第三次挥手： 服务器已经传输完毕，再次发送 FIN 通知客户端，数据已经传输完毕
- 第四次挥手： 客户端再次发送 ACK,进入 TIME_WAIT 状态；服务器和客户端关闭连接；
- 回到顶部

## 为什么建立连接是三次握手，而断开连接是四次挥手呢?

建立连接的时候， 服务器在 LISTEN 状态下，收到建立连接请求的 SYN 报文后，把 ACK 和 SYN 放在一个报文里发送给客户端。 而关闭连接时，服务器收到对方的 FIN 报文时，仅仅表示对方不再发送数据了但是还能接收数据，而自己也未必全部数据都发送给对方了，所以己方可以立即关闭，也可以发送一些数据给对方后，再发送 FIN 报文给对方来表示同意现在关闭连接，因此，己方 ACK 和 FIN 一般都会分开发送，从而导致多了一次。

- 回到顶部

## 前端有哪些页面优化方法?

- 减少 HTTP 请求数
- 从设计实现层面简化页面
- 合理设置 HTTP 缓存
- 资源合并与压缩
- 合并 CSS 图片，减少请求数的又一个好办法。
- 将外部脚本置底（将脚本内容在* 页面信息内容加载后再加载）
- 多图片网页使用图片懒加载。
- 在 js 中尽量减少闭包的使用
- 尽量合并 css 和 js 文件
- 尽量使用字体图标或者 SVG 图标，来代替传统的 PNG 等格式的图片
- 减少对 DOM 的操作
- 在 JS 中避免“嵌套循环”和 “死循环”
- 尽可能使用事件委托（事件代理）来处理事件绑定的操作
- 回到顶部

## 瀑布流

https://www.jianshu.com/p/cea62b6868ce

- 设置图片宽度一致

- 根据浏览器宽度以及每列宽度计算出列表个数，列表默认0
- 当图片加载完成，所有图片依次放置在最小的列数下面
- 父容器高度取列表数组的最大值
- 回到顶部

## 项目上线流程

- 打包上线(html css js)
- 提交文件到 git 仓库
- 服务器同步拉取 git 仓库代码
- 部署服务器(重新启动当前服务，jenkins 自动部署)

项目从开发到上线一般是，开发环境、测试环境、预发环境、生产环境。

**开发环境**：开发调试阶段，每次新的开发，要从最新的生产代码合并过来开始开发。

**测试环境**：一般指的是开发的模块具备测试条件，开始让测试人员进行测试。

**预发布环境**：把测试环境代码合并到预发环境，这时，访问线上数据，进行测试没有问题的话可以直接合到生产环境。

**生产环境（线上）**：合并预发环境的代码

项目在搭架子的时候，一定要把环境变量配置好，避免每个分支手动修改环境变量配置，通过命令去打包对应环境的代码，其实主要是接口 ip 地址的不同。

以上是 2C 的项目

2B 的项目就简单多了，但是要考虑的是前端依赖包，尽量避免全局安装，因政府部门驻场开发是没有网络的，各种插件及依赖包都不能通过 npm 等下载。

至于部署前端代码这块，可以自己配置 nginx 或者让后端小伙伴配置，前端可以不用管部署这块了，再者就是用 node 起个服务部署。

- 回到顶部

## 项目如何搭建

1、全局安装vue-cli
2、进入你的项目目录，创建一个基于 webpack 模板的新项目: vue init webpack 项目名

```
Vue build ==> 打包方式，回车即可；
Install vue-router ==> 是否要安装 vue-router，项目中肯定要使用到 所以Y 回车；
Use ESLint to lint your code ==> 是否需要 js 语法检测 目前我们不需要 所以 n 回车；
Set up unit tests ==> 是否安装 单元测试工具 目前我们不需要 所以 n 回车；
Setup e2e tests with Nightwatch ==> 是否需要 端到端测试工具 目前我们不需要 所以 n 回车；
```

3、进入项目：cd vue-demo，安装依赖
4、npm run dev，启动项目

```
1、build：构建脚本目录
  1）build.js    ==>  生产环境构建脚本；
  2）check-versions.js    ==>  检查npm，node.js版本；
  3）utils.js    ==>  构建相关工具方法；
  4）vue-loader.conf.js    ==>  配置了css加载器以及编译css之后自动添加前缀；
  5）webpack.base.conf.js    ==>  webpack基本配置；
  6）webpack.dev.conf.js    ==>  webpack开发环境配置；
  7）webpack.prod.conf.js    ==>  webpack生产环境配置；
2、config：项目配置
```

```
  1）dev.env.js    ==>   开发环境变量；
  2）index.js    ==>   项目配置文件；
  3）prod.env.js    ==>   生产环境变量；
3、node_modules：npm 加载的项目依赖模块
4、src：这里是我们要开发的目录，基本上要做的事情都在这个目录里。里面包含了几个目录及文件：
  1）assets：资源目录，放置一些图片或者公共js、公共css。这里的资源会被webpack构建；
  2）components：组件目录，我们写的组件就放在这个目录里面；
  3）router：前端路由，我们需要配置的路由路径写在index.js里面；
  4）App.vue：根组件；
  5）main.js：入口js文件；
5、static：静态资源目录，如图片、字体等。不会被webpack构建
6、index.html：首页入口文件，可以添加一些 meta 信息等
7、package.json：npm包配置文件，定义了项目的npm脚本，依赖包等信息
8、README.md：项目的说明文档，markdown 格式
9、.xxxx文件：这些是一些配置文件，包括语法配置，git配置等
```

- 回到顶部

## key的作用

让 vue 精准的追踪到每一个元素，高效的更新虚拟 DOM。 触发过渡

```
<transition>
  <span :key="text">{{ text }}</span>
</transition>
```

复制代码当 text 改变时，这个元素的 key 属性就发生了改变，在渲染更新时，Vue 会认为这里新产生了一个元素，而老的元素由于 key 不存在了，所以会被删除，从而触发了过渡。

- 回到顶部

## scoped属性作用/如何使用deep

在 Vue 文件中的 style 标签上有一个特殊的属性，scoped。当一个 style 标签拥有 scoped 属性时候，它的 css 样式只能用于当前的 Vue 组件，可以使组件的样式不相互污染。如果一个项目的所有 style 标签都加上了 scoped 属性，相当于实现了样式的模块化。

scoped 属性的实现原理是给每一个 dom 元素添加了一个独一无二的动态属性，给 css 选择器额外添加一个对应的属性选择器，来选择组件中的 dom。

```
<template>
    <div class="box">dom</div>
</template>
<style lang="scss" scoped>
.box{
    background:red;
}
</style>
复制代码vue将代码转译成如下：
.box[data-v-11c6864c]{
    background:red;
}
<template>
    <div class="box" data-v-11c6864c>dom</div>
</template>
```

### scoped 样式穿透

scoped 虽然避免了组件间样式污染，但是很多时候我们需要修改组件中的某个样式，但是又不想去除 scoped 属性。

**使用/deep/**

```
//Parent
<template>
<div class="wrap">
    <Child />
</div>
</template>

<style lang="scss" scoped>
.wrap /deep/ .box{
    background: red;
}
</style>

//Child
<template>
    <div class="box"></div>
</template>
```

**使用两个style标签**

```
//Parent
<template>
<div class="wrap">
    <Child />
</div>
</template>

<style lang="scss" scoped>
//其他样式
</style>
<style lang="scss">
.wrap .box{
    background: red;
}
</style>

//Child
<template>
    <div class="box"></div>
</template>
```

- 回到顶部

## ref的作用

- 获取dom元素this.$refs.box
- 获取子组件中的datathis.$refs.box.msg
- 调用子组件中的方法this.$refs.box.open()
- 回到顶部

## $nextTick / $set

- Vue 的 nextTick 涉及到 Vue 中 Dom 的异步更新

```
在Vue生命周期的created()钩子函数进行的DOM操作一定要放在Vue.nextTick()的回调函数中
在created()钩子函数执行的时候DOM 其实并未进行任何渲染，
而此时进行DOM操作无异于徒劳，
所以此处一定要将DOM操作的js代码放进Vue.nextTick()的回调函数中。与之对应的就是mounted()钩子函数，
因为该钩子函数执行时所有的DOM挂载和渲染都已完成，此时在该钩子函数中进行任何DOM操作都不会有问题。
```

- 在数据变化后要执行的某个操作，而这个操作需要使用随数据改变而改变的 DOM 结构的时候，这个操作都应该放进 Vue.nextTick()的回调函数中。

**nextTick中定义的三个重要变量**

- callbacks 用来存储所有需要执行的回调函数
- pending 用来标志是否正在执行回调函数
- timerFunc 用来触发执行回调函数

**$set**

由于 Vue 会在初始化实例时进行双向数据绑定，使用 Object.defineProperty()对属性遍历添加 getter/setter 方法，所以属性必须在 data 对象上存在时才能进行上述过程，这样才能让它是响应的。如果要给对象添加新的属性，此时新属性没有进行过上述过程，不是响应式的，所以会出想数据变化，页面不变的情况。此时需要用到$set。

**数组：**

```
this.$set(Array, index, newValue)
由于 JavaScript 的限制，Vue 不能检测以下变动的数组：
当你利用索引直接设置一个项时，例如：vm.items[indexOfItem] = newValue
解决：用$set方法
当你修改数组的长度时，例如：vm.items.length = newLength
解决：vm.items.splice(newLength)
```

**对象：**

```
this.$set(Object, key, value)
有时你想向已有对象上添加一些属性，例如使用 Object.assign() 或 _.extend() 方法来添加属性。
但是，添加到对象上的新属性不会触发更新。
在这种情况下可以创建一个新的对象，让它包含原对象的属性和新的属性：

this.someObject = Object.assign({}, this.someObject, { a: 1, b: 2 })
```

- 回到顶部

## XSS和CSRF区别

跨站脚本攻击（Cross Site Scripting），为了不和层叠样式表 CSS 混淆，故将跨站脚本攻击缩写为 XSS。恶意攻击者往 Web 页面里插入恶意 Script 代码，当用户浏览该页之时，嵌入其中 Web 里面的 Script 代码会被执行，从而达到恶意攻击用户的目的。

跨站请求伪造（Cross-site request forgery），是伪造请求，冒充用户在站内的正常操作。我们知道，绝大多数网站是通过 cookie 等方式辨识用户身份，再予以授权的。所以要伪造用户的正常操作，最好的方法是通过 XSS 或链接欺骗等途径，让用户在本机（即拥有身份 cookie 的浏览器端）发起用户所不知道的请求。

区别：

原理不同，CSRF 是利用网站 A 本身的漏洞，去请求网站 A 的 api；XSS 是向目标网站注入 JS 代码，然后执行 JS 里的代码。 CSRF 需要用户先登录目标网站获取 cookie，而 XSS 不需要登录 CSRF 的目标是用户，XSS 的目标是服务器 XSS 是利用合法用户获取其信息，而 CSRF 是伪造成合法用户发起请求

- 回到顶部

## 性能优化

- 使用CDN
- gzip压缩
- 文本压缩
- 合并请求
- 雪碧图
- 图片懒加载
- 缓存资源
- 减少DOM操作

- 回到顶部

## 事件捕获、事件冒泡、事件委托

### js事件执行先捕获后冒泡

**事件捕获:**

从顶级祖先元素开始，直到事件触发元素。两种事件流都会触发DOM中的所有对象，从document对象开始，也从document对象结束。js事件捕获一般通过DOM2级事件模型addEventListener来实现。

```
target.addEventListener(type,listener,useCapture)
第三个参数默认为false，表示在冒泡阶段触发事件，设置为true时表示在捕获阶段触发。
```

```
<script>

//事件捕获
window.onload=function(){
    let box=document.getElementById("box");
    let middle=document.getElementById("middle");
    let inner=document.getElementById("inner");
    box.addEventListener("click",function(){console.log("box")},true);
    middle.addEventListener("click",function(){console.log("middle")},true);
    inner.addEventListener("click",function(){console.log("inner")},true);
}
</script>
当点击inner绑定事件时，控制台会直接输出，box，middle,inner
```

**事件冒泡:**

当事件在一个元素上触发之后，事件会逐级传播给父级元素，直到 document 为止，有的浏览器直到 window 为止，这就是事件冒泡事件。blur 事件，focus 事件，load 事件。

**事件委托:**

事件委托又称事件代理，事件委托就是利用事件冒泡，只指定一个事件处理程序，就可以管理某一类型的所有事件。

==事件委托的好处==：我们都知道，减少 dom 操作可以提高网页性能，当一个页面的父级元素和很多自己元素都需要同一事件的时候，我们不可能每个元素都去给它绑定一个事件。

并不是所有的情况都适用于事件冒泡，当出现父子级之间的注册事件不一致时，就不适用。

**js 阻止事件冒泡**

平时开发过程中，会用到大量的事件冒泡事件，但是可能我们在某个子集标签不需要传递事件给父级，这个时候就需要阻止它事件的冒泡。

一般使用 stopPropagation 来阻止事件的冒泡，IE 中使用 cancelBuble=true，stopPropagation 也是事件对象(Event)的一个方法，作用是阻止目标元素的冒泡事件，但是不会阻止默认行为。

```
let btna = document.getElementById('btn');
btna.onclick=function(e){
    window.event? window.event.cancelBubble = true : e.stopPropagation();
 };
```

**js 阻止浏览器默认行为**

开发过程中，总会出现各种浏览器的默认行为，这时候就需要阻止浏览器的默认行为，一般情况下，使用

preventDefault 阻止浏览器的默认行为，在 IE 浏览器下，使用 returnValue = false;

javascript 的 return false 只会阻止默认行为，而是用 jQuery 的话则既阻止默认行为又防止对象冒泡。

```
//阻止浏览器的默认行为
function stopDefault( e ) {
    //一般情况下
    if ( e && e.preventDefault )
        e.preventDefault();
    //IE中
    else
        window.event.returnValue = false;
    return false;
}
```


- 回到顶部