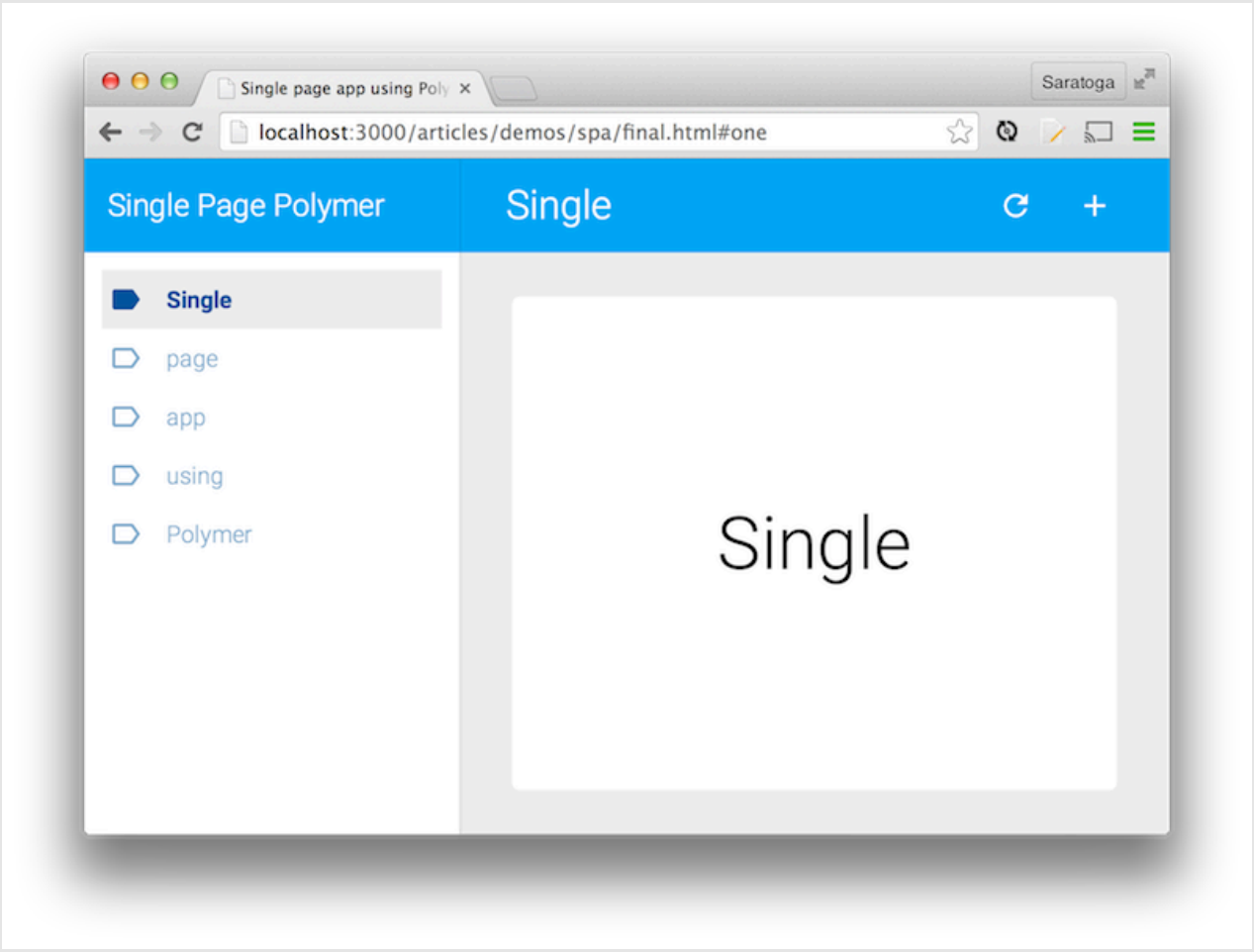
深度剖析 Vue.js 经典面试题

1. 谈一谈你对 SPA 单页面的理解，它的优缺点分别是什么



SPA（ single-page application ）仅在 Web 页面初始化时加载相应的 HTML、JavaScript 和 CSS。一旦页面加载完成，SPA 不会因为用户的操作而进行页面的重新加载或跳转；取而代之的是利用路由机制实现 HTML 内容的变换，UI 与用户的交互，避免页面的重新加载。

优点：

- 用户体验好、快，内容的改变不需要重新加载整个页面，避免了不必要的跳转和重复渲染；
- 基于上面一点，SPA 相对对服务器压力小；
- 前后端职责分离，架构清晰，前端进行交互逻辑，后端负责数据处理；

缺点：

- 首屏（初次）加载慢：为实现单页 Web 应用功能及显示效果，需要在加载页面的时候将 JavaScript、CSS 统一加载，部分页面按需加载；
- 不利于 SEO：由于所有的内容都在一个页面中动态替换显示，所以在 SEO 上其有着天然的弱势。

2. SPA 单页面应用的实现方式有哪些

> 前端路由的实现原理。

- hash 模式
- history 模式

在 `hash` 模式中，在 `window` 上监听 `hashchange` 事件（地址栏中hash变化触发）驱动界面变化；

在 `history` 模式中，在 `window` 上监听 `popstate` 事件（浏览器的前进或后退按钮的点击触发）驱动界面变化，监听 a 链接点击事件用 `history.pushState`、`history.replaceState`方法驱动界面变化；

直接在界面用显示隐藏事件驱动界面变化。

3. 使用过 Vue SSR 吗？说说 SSR?

> Server Side Renderer
>
> Vue.js 是构建客户端应用程序的框架。默认情况下，可以在浏览器中输出 Vue 组件，进行生成 DOM 和操作 DOM。然而，也可以将同一个组件渲染为服务端的 HTML 字符串，将它们直接发送到浏览器，最后将这些静态标记"激活"为客户端上完全可交互的应用程序。
>
> 即：SSR大致的意思就是vue在客户端将标签渲染成的整个 html 片段的工作在服务端完成，服务端形成的html 片段直接返回给客户端这个过程就叫做服务端渲染。

服务端渲染 SSR 的优缺点如下：

（1）服务端渲染的优点：

- 更好的 SEO： 因为 SPA 页面的内容是通过 Ajax 获取，而搜索引擎爬取工具并不会等待 Ajax 异步完成后再抓取页面内容，所以在 SPA 中是抓取不到页面通过 Ajax 获取到的内容；而 SSR 是直接由服务端返回已经渲染好的页面（数据已经包含在页面中），所以搜索引擎爬取工具可以抓取渲染好的页面；
- 更快的内容到达时间（首屏加载更快）： SPA 会等待所有 Vue 编译后的 js 文件都下载完成后，才开始进行页面的渲染，文件下载等需要一定的时间等，所以首屏渲染需要一定的时间；SSR 直接由服务端渲染好页面直接返回显示，无需等待下载 js 文件及再去渲染等，所以 SSR 有更快的内容到达时间；

(2) 服务端渲染的缺点：

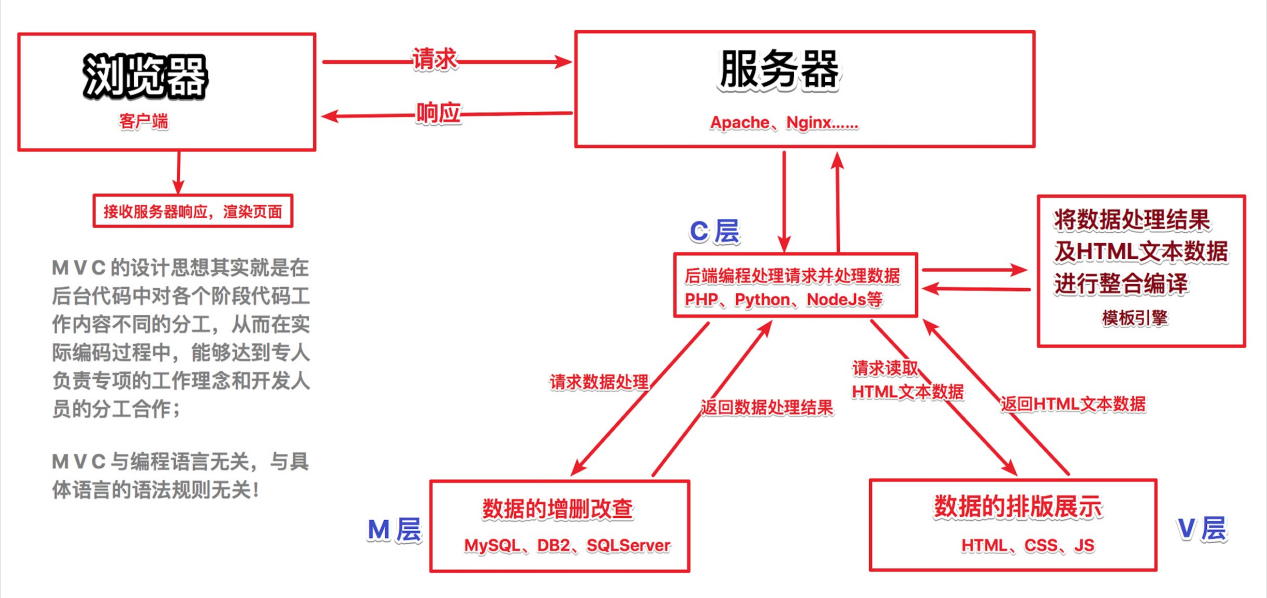
- 更多的开发条件限制： 例如服务端渲染只支持 beforCreate 和 created 两个钩子函数，这会导致一些外部扩展库需要特殊处理，才能在服务端渲染应用程序中运行；并且与可以部署在任何静态文件服务器上的完全静态单页面应用程序 SPA 不同，服务端渲染应用程序，需要处于 Node.js server 运行环境；
- 更多的服务器负载：在 Node.js 中渲染完整的应用程序，显然会比仅仅提供静态文件的 server 更加大量占用CPU 资源 (CPU-intensive - CPU 密集)，因此如果你预料在高流量环境 (high traffic) 下使用，请准备相应的服务器负载，并明智地采用缓存策略。

如果没有 SSR 开发经验的同学，可以参考本文作者的另一篇 SSR 的实践文章《Vue SSR 踩坑之旅》，里面 SSR 项目搭建以及附有项目源码。

4. 谈一谈你对 MVVM 的理解

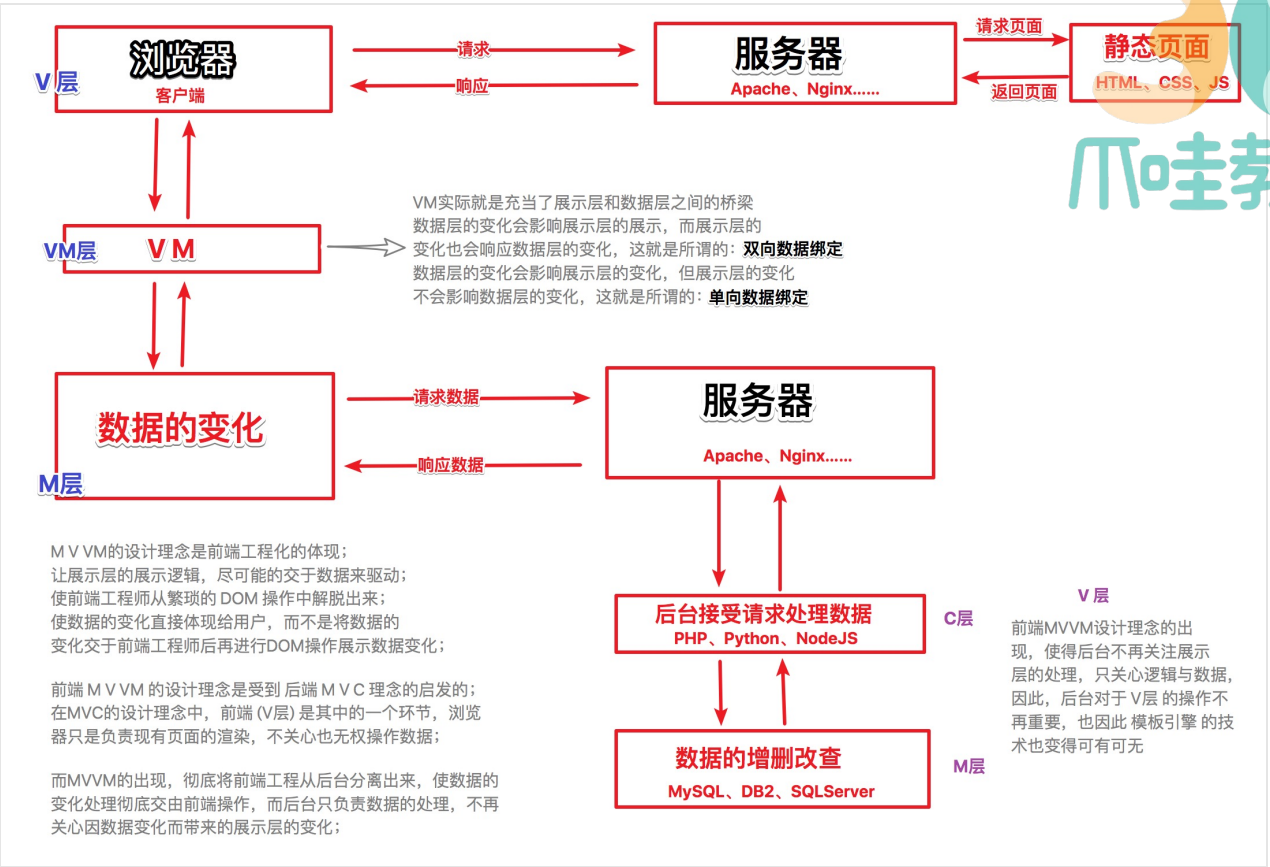
传统的服务端 MVC 架构模型：View

```
models  数据模型，专门提供数据支持的
controllers 控制器模块，处理不同的页面请求的或者处理接口请求
views 视图文件
```



> 核心理念：单一职责，分工协作
>
> 优点：
>
> - 更好的开发效率
> - 更好的可维护性

MVVM 模式常见于用于构建用户界面的客户端应用。



MVVM 模型：

字面意义是这样的：

MVVM 是 Model-View-ViewModel 的缩写，MVVM 是一种设计思想。

- Model 层代表数据模式，也可以在 Model 中定义数据修改和操作的业务逻辑
- View 代表 UI 组件，它负责将数据模型转化为 UI 展现出来
- ViewModel 是一个同步 View 和 Model 的对象。

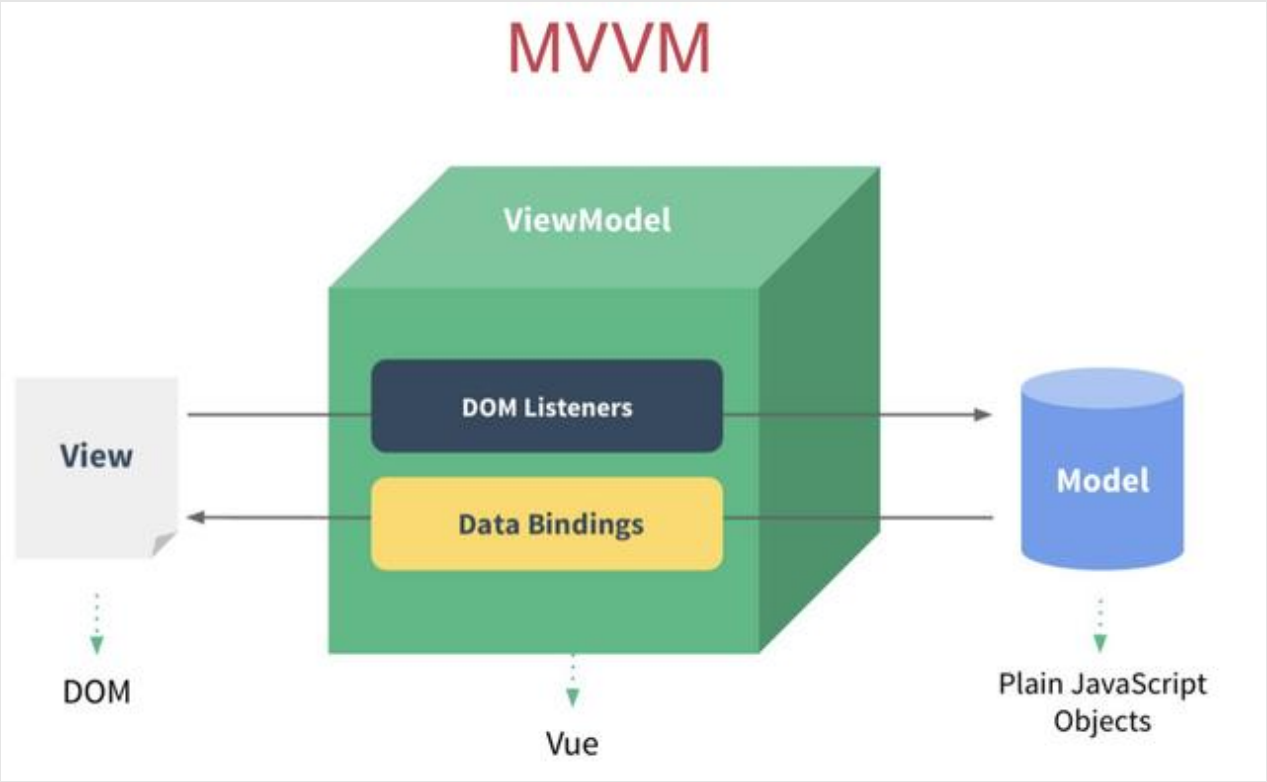
我的理解：

- 前端开发早期的时候都是**操作 DOM**
- 后来使用 jQuery 让我们**提高了操作 DOM 的效率**，但从开发角度**还是在大量的手动操作 DOM**
- MVVM 模式让以往手动操作 DOM 的方式彻底解脱了，它**不要用户自己操作 DOM**，而是**将普通数据绑定到 ViewModel 上，会自动将数据渲染到页面中，视图变化会通知 ViewModel 层更新数据，数据变化也会通过 ViewModel 层更新视图**，因此 MVVM 中最重要的角色就是 ViewModel，真正意义上把视图和数据实现了解耦，提高了开发效率。
- MVVM 模式主要用于构建用户界面的前端应用
  - 微软的 WPF，构建客户端应用的
  - 手机应用，iOS APP、Android App
  - Web 应用
  - ...

核心：

- MVVM 模式让我们从繁琐的 DOM 操作中彻底解放了
- MVVM 也叫数据驱动视图

下面是在 Vue 中的 MVVM：



用 Vue 中组件代码来表示 MVVM 的话就是这样的：

```
View 视图层
<template>
  <div>
    <h1>{{ message }}</h1>
    <ul>
      <li v-for="item in list" :key="item.id">{{ item.title }}</li>
    </ul>
  </div>
</template>

<script>
// Model 层
// ViewModel 层，就是 Vue 本身
export default {
  data () {
    // 普通 JavaScript 数据都会被转化到 ViewModel 层
    return {
      message: 'hello',
      list: []
    }
  },
```

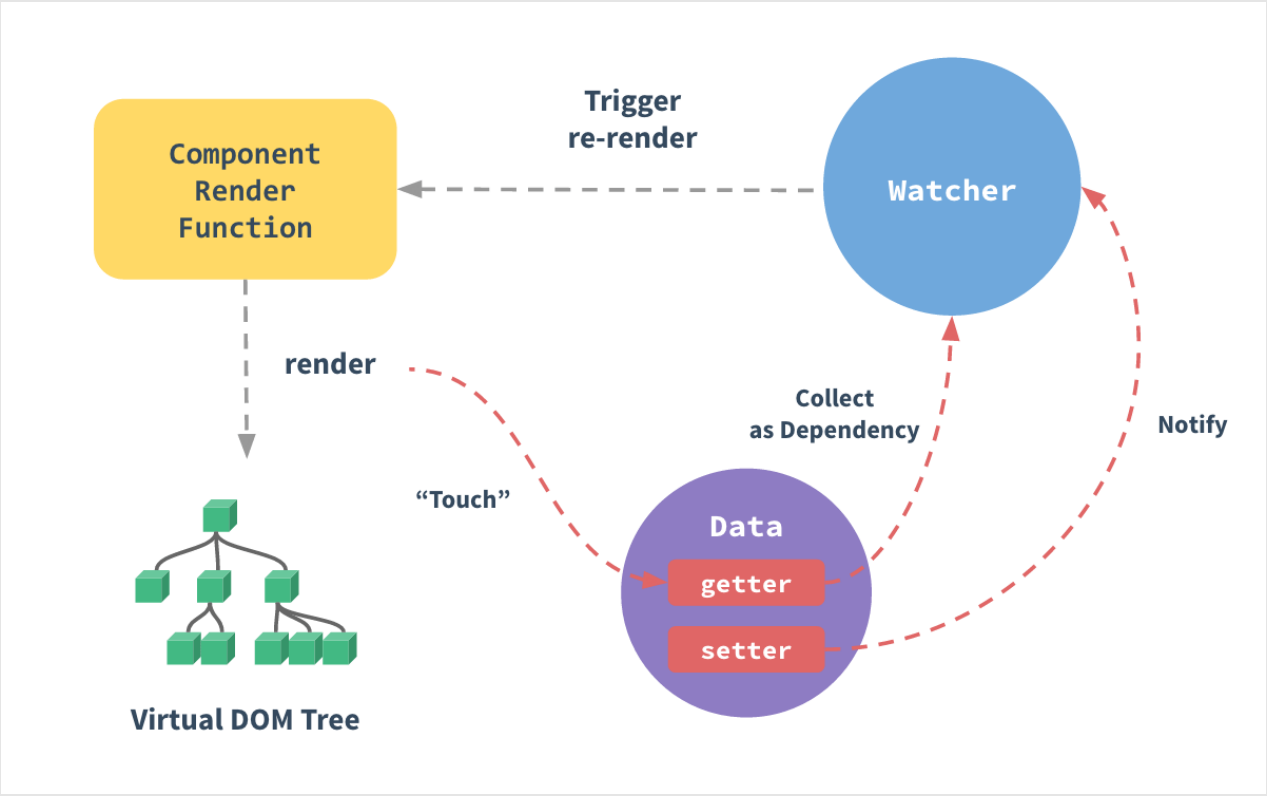
```
  async created () {
    const { data } = await ajax({
      method: 'GET',
      url: 'xxx'
    })
    this.list = data
  }
}
</script>

<style>

</style>
```

5. 谈一谈你对 Vue.js 的响应式数据的理解

核心知识点：

- 在 Vue 2.x 版本中使用的是 Object.defineProperty
- 在 Vue 3.x 版本中使用的是 ECMAScript 6 中新增的 Proxy

先来看一下 Vue.js 官网的解释：https://cn.vuejs.org/v2/guide/reactivity.html。



总结一下：

- 当你把一个普通的 JavaScript 对象传入 Vue 实例作为 `data` 选项，Vue 将遍历此对象所有的 property，并使用 `Object.defineProperty` 把这些 property 全部转为 getter/setter。
  - `Object.defineProperty` 是 ES5 中一个无法 shim 的特性，这也就是 Vue 不支持 IE8 以及更低版本浏览器的原因。
- 当使用这些数据属性时，会进行依赖收集（收集到当前组件的 watcher）
  - 每个组件都对应一个 watcher 实例，它会在组件渲染的过程中把“接触”过的数据记录为依赖
- 之后当依赖项的 setter 触发时，会通知 watcher，从而使它关联的组件重新渲染

下面我们可以通过源码来理解它的原理实现。

核心方法	作用	源码
function Vue () {}	Vue 构造函数	core/instance/index.js:8
Vue.prototype._init	初始化实例成员	core/instance/init.js:16
vm.initState	初始化 props、data、watch、computed	core/instance/state.js:50
vm.initData	初始化用户传入的 data 数据	core/instance/state.js:112
observe		
new Observer	将数据进行监视	core/observer/index.js:124
this.walk	处理对象的监视	core/observer/index.js:55
defineReactive	循环对象属性定义响应式变化	core/observer/index.js:135
Object.defineProperty	核心方法，监视数据（拦截数据的访问和修改）	core/observer/index.js:157
get	收集依赖	
set	通知更新	

简而言之就是两件事儿：

- 拦截属性的获取进行依赖收集
- 拦截属性的修改对相关依赖进行通知更新

6. 浏览器和 Node.js 中的事件循环

JavaScript 单线程：同一时间只能干一件事儿，假如有一个死循环，那你整个程序就崩溃了。

所以 JavaScript 不适合做大量运算的任务。

JavaScript 中有很多的异步任务：发请求、读写文件，这些都是比较耗时的，需要进行 IO 工作。

主线程执行普通的代码，遇到异步任务就把它扔到事件队列中。完了才提取队列中的任务开始执行，异步任务执行多线程（环境提供的，我们也不到）。

异步成功，把 callback 放到队列中。

中间过程，主线程会不断的取提取队列中的任务，如果是回调，就直接执行。