鸿蒙面试题-刘玉静 整理版

1.什么是ability?

> Ability Kit（程序框架服务）提供了应用程序开发和运行的应用模型，提供了应用程序必备的组件和运行机制。
>
> 包含两种模型：
>
> FA模型、Stage模型
>
> 其中Stage模型。对 `Ability` 生命周期、上下文环境等调用管理的能力，包括 `Ability` 创建、销毁、转储客户端信息等, `Ability` 主要分为两种类型： `UIAbility` 和 `ExtensionAbility` 。

2.FA模型、Stage模型有什么区别?

> - FA（模型：从API 7开始支持的模型，已经不再主推。
> - Stage模型：目前主推且会长期演进的模型。由于提供了AbilityStage、WindowStage等类作为应用组件和Window窗口的“舞台”，因此称这种应用模型为Stage模型。
>
> **Stage模型与FA模型最大的区别在于**：Stage模型中，多个应用组件共享同一个ArkTS引擎实例；而FA模型中，每个应用组件独享一个ArkTS引擎实例

3.UIAbility的生命周期

> UIAbility的生命周期包括Create、Foreground、Background、Destroy四个状态

4.UIAbility组件启动模式

> - singleton（单实例模式）
> - multiton（多实例模式）
> - specified（指定实例模式）

5.什么是显示Want隐式Want

> 显式 Want:在创建 Want 对象时会指定目标组件的 *bundleName* 和 *abilityName*
>
> 隐式Want:不知道具体的目标组件，但知道某些动作或数据类型时,例如，只想分享一张图片而不关心哪个应用来处理这个操作

6.页面生命周期

> - **onPageShow**: 当页面显示时触发，用于执行页面显示时的操作。
> - **onPageHide**: 当页面隐藏时触发，用于执行页面隐藏时的操作。
> - **onBackPress**: 当用户点击手机下方的返回按钮时触发。

7.组件生命周期

> - **aboutToAppear**: 当组件即将出现时回调。这个回调在创建自定义组件的新实例后，执行其 `build` 函数之前执行。

- **aboutToDisappear**: 当自定义组件即将销毁时执行。开发者可以在这个回调中执行组件销毁前的清理工作。

8.数据持久化

> 数据持久化是ArkData框架中的API：
>
> 用户首选项（Preferences）
>
> relationalStore 关系型数据库，底层基于SqlLite
>
> distributedKVStore 键值数据库

9.父子组件通信

> 可以使用状态管理器中的prop和link装饰器
>
> Prop 子集不能修改数据 父级改了 子集会响应
> Link 子集和父级都可以修改数据 且双方都会响应

10.navigation 和router 的区别

> Router 和 Navigation 虽都用于页面导航，但存在明显差异。Router 功能基础，侧重于简单的页面切换，支持通过 URL 跳转，页面间数据传递需手动处理，适用于页面结构和导航逻辑简单的场景，因功能轻量，在简单场景下性能开销小、导航速度快，使用时主要调用 Router.pushUrl 方法；而 Navigation 功能更丰富，支持栈管理和便捷的返回数据处理，适合复杂导航及需处理返回数据的场景，不过因管理页面栈等操作性能开销稍大。
>
> 1. **实现机制**
>   - **Navigation**：作为页面的根容器，主要包含主页和内容页。它通过NavPathStack实现页面路由，支持pushPath和pushPathByName两种方法进行页面跳转。
>   - **Router**：通过不同的URL地址实现页面路由，支持pushUrl和replaceUrl两种跳转模式，并且可以传递参数。页面栈的最大容量为32个页面，超过这个限制可以调用clear()方法释放内存空间。
> 2. **跳转模式**
>   - **Navigation**：提供pushPath和pushPathByName两种方法，支持pop、move、clear等多种操作，能够灵活控制页面栈的状态。
>   - **Router**：提供pushUrl和replaceUrl两种跳转模式，决定了目标页面是否会替换当前页面。同时提供Standard和Single两种实例模式，决定目标URL是否对应多个实例。
> 3. **数据传递**
>   - **Navigation**：可以通过params方法向目标页面传递参数，支持对象的传递。
>   - **Router**：在调用跳转方法时添加params属性传递参数，目标页可以通过getParams()方法获取传递过来的参数。
> 4. **导航控制**
>   - **Navigation**：支持更丰富的导航控制，如动态加载、跨包引用等。
>   - **Router**：主要提供基本的页面跳转和返回功能，可以通过router.back()等方式返回上一页。
> 5. **适用场景**
>   - **Navigation**：适用于需要统一页面跳转管理、复杂导航控制的场景。
>   - **Router**：适用于简单的页面间跳转和数据传递的场景。
> 6. **性能考量**
>   - **Navigation**：由于其丰富的导航控制，可能会占用更多的系统资源。
>   - **Router**：提供基本的跳转功能，可能在资源占用上更为经济。

7. **易用性**
    - **Navigation**：学习曲线可能较陡，需要理解多种导航控制和路由转场的概念。
    - **Router**：相对简单直观，特别是对于熟悉Web开发中URL路由的开发者来说。
8. **平台版本**
    - **Navigation**：在不同API版本中有不同的实现方式，如NavPathStack在API Version 9上需要配合NavRouter使用，而在API Version 10中推荐使用NavPathStack配合NavDestination。
    - **Router**：提供的是较为通用的URL路由功能，在不同平台版本中变化不大。

11.ts 跟arkTS的区别

ArkTS基于TypeScript的增强——规范的代码更好的保证正确性和性能，扩展语言特性增强UI声明式范式和并发能力支持。

- ArkTS中更严格的类型检查
- ArkTS不支持在运行时更改对象布局
- ArkTS对象字面量需标注类型
- ArkTS禁止使用对象字面量初始化具有复杂constructor的类，在ArkTS中，如果需要创建这些class的实例，需要使用new操作符创建

12.优化性能

1. **渲染优化**

- 使用懒加载和虚拟列表
- 避免不必要的重渲染
- 优化条件渲染逻辑

2. **状态管理**

- 合理拆分状态
- 使用计算属性
- 实现批量更新

3. **资源管理**

- 及时释放资源
- 实现分片处理
- 优化内存使用

4. **网络优化**

- 实现请求缓存
- 合并重复请求
- 添加错误重试

5. **监控与调试**

- 实现性能监控
- 添加错误追踪
- 优化日志记录

其他：

- 将网络请求提前至AbilityStage/UIAbility生命的onCreate()生命周期回调函数中，将首刷或二刷的时间提前，减少用户等待时间。
- 启动页图标startWindowIcon分辨率建议不超过256px*256px。

- 在AbilityStage、UIAbility和自定义组件的生命周期回调函数中不建议直接执行耗时任务，比如复杂的计算任务、同步文件读写等耗时任务，建议通过异步任务延迟处理或者放到其他线程执行。
- import模块按需加载，移除初始化阶段不需要的模块导入，考虑动态加载耗时的模块。
- 减少使用嵌套export *全量导出和import *全量引用。
- 拆分HAR包导出文件或导入冷启动相关文件时使用全路径，减少应用冷启动中.ets文件执行耗时。
- 避免多个HAP/HSP对相同HAR的引用。
- 单HAP场景下，模块推荐使用多HAR，不推荐使用HSP。
- 通过使用合理的布局结构、使用懒加载等UI优化方法来减少首帧绘制时间。
- 使用本地存储首页数据，减少首帧展示完成时延，减少用户可见白屏或白块时间。
- 减少组件的嵌套层级
- 使用 LazyForEach、cachedCount进行长列表优化
- 控制渲染范围、减少布局节点、优化组件绘制、控制状态刷新、优化动画帧率

13.后台任务

后台任务，包括短时任务、长时任务、延迟任务、代理提醒等

14.Builder和BuilderParams的区别

Builder是当前组件的UI复用结构 BuilderParams是接收父组件传入的UI复用结构， 传过来的类型是UI复用结构类型

Builder传值 想要响应式必须是对象，如果基础数据类型，不具备响应式

15.如何封装公共样式?

- 封装组件
- 提取Builder
- 提取Styles和Extends、AttributeModifier

16.Promise 异步、和TaskPool、Worker有什么区别吗?

Promise：异步仍然是在当前线程中运行任务，结果以异步方式返回

② TaskPool:：对于一些耗时操作，使用Promise占用了主线程资源，可能会导致ANR，所以出现了taskPool来保证可以起子线程处理耗时操作。taskPool存活最长时间为3分钟，运行超过3分钟，那么会被系统杀掉。

③ Worker：存在一个特别耗时的操作，或者可能存在业务需要一直死循环运行。通过worker可以启动一个线程，该线程不存在时长限制，可以一直存在，所以我们可以在里边做一些耗时操作。worker线程最多可以在APP里同时存在8个一起运行，超出数量的不会运行。但是创建可以创建多个，所以当worker不需要的时候要及时停止运行。worker对内存有影响，每个空worker起来后就会占用2MB左右内存

17.HarmonyOS和OpenHarmony的区别?

HarmonyOS 是华为基于 OpenHarmony 等开发的商用版本，OpenHarmony 是开源项目，二者在性质来源、技术实现等方面存在差异

18.V1状态装饰器有什么?

@State、@Prop、@Link、@Provide 和 @Consume

19.V2状态装饰器主要解决了什么问题？都有什么？

> V1的实际使用中，开发人员发现@Observed和@ObjectLink 监听实现**多层级嵌套对象的更新的方案**，太过于臃肿。当需要监听处理更新的多层级对象是七八层，就需要配套创建七八层的ObjectLink，代码太过于冗余。
>
> V2就是为了解决该问题，华为官方才提出的新状态管理装饰器方案。该方案在解决该问题的基础上，也对V1的进行加强。

V1和V2混用注意点

1. 使用@ObservedV2与@Trace装饰的类不能和@State等V1的装饰器混合使用，编译时报错。
2. 继承自@ObservedV2的类无法和@State等V1的装饰器混用，运行时报错。
3. @ObservedV2的类实例目前不支持使用JSON.stringify进行序列化。
4. @ObservedV2、@Trace不能与@Observed、@Track混合使用。
5. @Trace是class中属性的装饰器，不能用在struct中。
6. @Trace不能用在没有被@ObservedV2装饰的class上。
7. @ObservedV2仅能装饰class，无法装饰自定义组件。
8. 非@Trace装饰的成员属性用在UI上无法触发UI刷新。

20.MVVM是什么？

> MVVM（Model - View - ViewModel）是一种前端开发的软件设计模式，它是在 MVC（Model - View - Controller）和 MVP（Model - View - Presenter）模式的基础上发展而来，主要用于分离视图（UI）和业务逻辑，提高代码的可维护性、可测试性和可扩展性。 MVVM模式，应用的UI以及基础表示和业务逻辑被分成三个独立的类：视图，用于封装UI和UI逻辑；视图模型，用于封装表示逻辑和状态；以及模型，用于封装应用的业务逻辑和数据。

21.HAR和HSP分别是什么？区别是什么？

> HAR 是静态资源共享包，用于存放图片、配置文件等不频繁变更的内容；HSP 是用于底层及应用层动态更新文件的包。二者区别在于存储内容和更新频率不同

22.应用切到后台后，如何保证任务继续进行？

> 当应用切到后台后，系统会对应用进行一定的管控，以优化资源利用和用户体验。为保证任务在后台继续进行，可采用以下方式：
>
> 申请后台任务权限，需要配置权限ohos.permission.KEEP_BACKGROUND_RUNNING

23.数据临时和持久存储技术方案是什么？

> 数据临时存储可使用内存变量，在程序运行期间保存数据，应用关闭或进程被杀时数据丢失；还能利用函数内部的局部变量实现临时存储。而数据持久存储方面，可采用 LocalStorage，它能在本地存储数据，即使应用关闭或进程被杀，数据依然保留，存储容量有限，约 5 - 10 兆；APP Storage 也是常用方案，它存储无限制，数据存于沙箱，仅卸载应用时数据才被清除；此外，还能借助数据库，如关系型数据库或轻量级的文件数据库，来高效管理和持久存储结构化数据，满足复杂数据存储和查询需求。

24.webview如何和原生双向通信？

> runJavaScript：异步执行JavaScript脚本，并通过回调方式返回脚本执行的结果。 onConfirm：网页调用confirm()告警时触发此回调。 或在h5注册map与回调，通过调用注册的回调方法向ArkTS侧传递数据

25.鸿蒙如何打包插件和应用

> 打包应用时，首先要确保项目代码开发完成且无错误，在 DevEco Studio 里，通过配置 config.json 文件明确应用的基本信息、权限需求等；接着，选择合适的构建方式，可使用 Studio 自带的构建工具，按提示完成签名配置，包括选择签名类型、创建或导入密钥等；最后执行构建任务，系统会生成 .hap（HarmonyOS Ability Package）或 .app 格式的应用安装包。而打包插件，若为 HSP（HarmonyOS Service Package）插件，同样先完善代码开发，在 config.json 里对插件进行详细配置，明确插件类型、入口文件等信息；之后进行编译操作，完成签名后即可生成 .hsp 格式的插件包。无论打包应用还是插件，都要确保配置准确、签名合规，以保证能正常安装和使用。

26.如何进行全局状态管理?

> **@Provide+@Consume装饰器** 适用场景：适用于整个组件树而言“全局”的状态共享，且该状态改动不频繁的场景。 工作原理：通过在最顶层组件中使用 `@Provide` 装饰器提供状态，其他需要共享状态的组件通过 `@Consume` 装饰器获取该状态 。 优点：减少了状态传递的层级，提升了代码的可维护性和可拓展性。 注意事项：确保状态的生命周期与组件树的生命周期一致，避免不必要的UI刷新。
>
> **AppStorage** 适用场景：适用于整个应用而言“全局”的变量或应用的主线程内多个 `UIAbility` 实例间的状态共享。 工作原理：`AppStorage` 与应用的进程绑定，由UI框架在应用程序启动时创建，当应用进程终止，`AppStorage` 被回收。 优点：适用于需要在整个应用中共享状态的场景。 注意事项：确保状态的生命周期与应用进程一致，避免在应用退出后仍有状态存在。
>
> **LocalStorage** 适用场景：适用于单个Ability而言“全局”的变量，主要用于不同页面间的状态共享。 工作原理：`LocalStorage` 的生命周期由应用程序决定，当应用释放最后一个指向 `LocalStorage` 的引用时，`LocalStorage` 被垃圾回收。 优点：适用于需要在单个UIAbility中不同页面间共享状态的场景。 注意事项：确保状态的生命周期与应用程序的生命周期一致，避免在应用退出后仍有状态存在。

27.LocalStorage在应用重启后数据会消失吗?

> 会,因为LocalStorage 是一种用于页面或组件级别的数据存储方式，它允许开发者在页面或组件的生命周期内存储和检索数据。LocalStorage 的数据存储在内存中，因此它的读写速度相对较快。但是，当应用重启后，LocalStorage 中的数据会丢失。

28.兄弟组件如何通信?

> **通过公共父组件传递** 如果两个组件是同一个父组件的子组件，可以通过父组件来传递数据或事件。父组件可以作为中介，将一个子组件的数据或事件传递给另一个子组件。
>
> **使用全局状态管理** 使用全局状态管理（如 AppStorage、LocalStorage）来存储共享数据。兄弟组件可以独立地读取和更新这个全局状态，从而实现通信。

29. HarmonyOS与Android和iOS有什么区别?

> HarmonyOS 是华为开发的一个开源、分布式的操作系统。它设计用于多种设备，包括智能手机、平板电脑、智能电视和物联网设备。与Android和iOS的主要区别在于：
>
> - 分布式架构：HarmonyOS支持跨设备无缝协作，允许设备之间共享硬件资源。
> - 性能：HarmonyOS优化了任务调度和内存管理，提高了性能和响应速度。
> - 安全性：HarmonyOS采用了多层次的安全策略，包括数据加密和安全启动。
> - 生态系统：HarmonyOS正在构建自己的应用生态系统，鼓励开发者使用Ark Ts和ArkUI框架。

30.跨设备通信的方式有哪些?

- 分布式软总线：一种高性能的通信机制，允许设备之间建立直接连接，进行数据传输。
- 蓝牙：使用标准的蓝牙技术进行设备间的通信。
- WLAN：通过WLAN网络实现设备间的通信。
- 远程服务调用：通过分布式任务调度实现跨设备的服务调用。

31.Ability是如何与用户交互的?

界面显示：Ability可以包含一个或多个AbilitySlice，用于显示UI界面并与用户进行交互。

事件处理：Ability可以处理用户的输入事件，如触摸、按键等。

数据绑定：Ability可以使用数据绑定机制，将UI组件与数据模型绑定，实现数据的自动更新和交互。

通知：Ability可以通过系统通知机制向用户发送通知，即使应用不在前台运行。

32.分布式数据库是如何实现数据同步的?

分布式事务：确保跨设备的数据库操作具有原子性、一致性、隔离性和持久性。

数据版本控制：为数据添加版本号，确保同步时数据的一致性。

冲突解决策略：定义冲突解决策略，处理并发操作导致的数据冲突。

网络状态感知：根据网络状态智能同步数据，优化同步效率和流量使用。

33.如何优化应用的性能?

- 内存管理：合理分配和释放内存，避免内存泄漏。
- 后台优化：合理使用后台服务和定时任务，避免不必要的后台运行。
- UI渲染优化：使用轻量级的UI组件，减少布局复杂度，优化渲染性能。
- 资源优化：压缩图片和媒体资源，减少应用的体积和加载时间。

34.HarmonyOS中的权限管理模型是怎样的?

- 权限声明：应用在config.json中声明所需的权限。
- 权限申请：在应用运行时，根据需要动态申请权限。
- 权限检查：在执行敏感操作前，检查是否已获得相应权限。
- 权限分组：系统将权限分为不同的组，便于管理和申请。

35.LazyForEach是什么?

LazyForEach 是一个用于高效渲染列表的组件或功能，它允许开发者在用户滚动列表时才加载和渲染列表项，而不是一次性渲染整个列表。这种按需渲染的方式可以显著提高应用的性能，特别是在处理大量数据时。

36.LazyForEach的工作原理是什么?

LazyForEach 的工作原理通常是基于用户的滚动位置来动态地创建和销毁列表项的组件实例。当用户滚动到列表的某个部分时，LazyForEach 会加载并渲染那些即将进入视图的列表项，同时可能会卸载那些滚出视图的列表项，以节省内存和计算资源。

37.如何获取屏幕的安全区域?

1. 可以通过设置组件的expandSafeArea属性来获取获取UIWindow：首先，你需要获取到当前页面的UIWindow实例。
2. 调用getSafeArea方法：通过UIWindow实例调用getSafeArea方法来获取安全区域的Rect对象。 示例:

38.@Provider和@Consumer vs @Provide和@Consume的区别?

能力	V2 装饰器@Provider 和@Consumer	V1 装饰器@Provide 和@Consume
本地初始化	允许本地初始化，当找不到@Provider 的时候使用本地默认值。	禁止本地初始化，当找不到对应的@Provide 时候，会抛出异常。
支持类型	支持 function。	不支持 function。
观察能力	仅能观察自身赋值变化，如果要观察嵌套场景，配合@Trace 一起使用。	观察第一层变化，如果要观察嵌套场景，配合@Observed 和@ObjectLink 一起使用。
alias 和属性名	alias 是唯一匹配的 key，如果缺省 alias，则默认属性名为 alias。	alias 和属性名都为 key，优先匹配 alias，匹配不到可以匹配属性名。
从父组件初始化	禁止。	允许。
支持重载	默认开启，即@Provider 可以重名，@Consumer 向上查找最近的@Provider。	默认关闭，即在组件树上不允许有同名@Provide。如果需要重载，则需要配置 allowOverride。

39.@Prop和@ObjectLink装饰器有什么区别?

1.**用途**

- `@Prop`装饰器：主要用于在组件之间传递数据，将父组件的值传递给子组件。它定义了子组件的属性，可以接收来自父组件的赋值。`@ObjectLink`用于建立对象之间的链接，通常用于在组件内部或组件之间共享和同步状态。它可以将一个对象的属性与另一个对象的属性进行链接，当一个对象的属性发生变化时，另一个对象的属性也会自动更新。

1. **数据传递方式**

- `@Prop`：是单向的数据传递，从父组件到子组件。父组件可以设置子组件的`@Pro`p属性值，但子组件不能直接修改这个值。`@ObjectLink`是双向的数据传递，父组件和子组件都可以修改子组件的`@ObjectLink`属性值。

1. **性能**

- `@Prop`会深拷贝数据，具有拷贝的性能开销，性能低于`@ObjectLink`

40.ForEach和LazyForEach的区别?

ForEach和LazyForEach都是用于渲染列表的装饰器，它们的区别在于：

- ForEach：渲染列表时，会将列表中的每一项都渲染一次，适用于列表项数量较少的情况。
- LazyForEach：渲染列表时，只渲染当前可见的列表项，适用于列表项数量较多的情况。

41.hap、har、hsp三者的区别?

> HAP（Harmony Ability Package）是应用安装和运行的基本单元。HAP包是由代码、资源、第三方库、配置文件等打包生成的模块包，其主要分为两种类型：entry和feature。（又称ability）
>
> HAR（Harmony Archive）是静态共享包，可以包含代码、C++库、资源和配置文件。通过HAR可以实现多个模块或多个工程共享ArkUI组件、资源等相关代码。（又称static library, 静态共享包）
>
> HSP（Harmony Shared Package）是动态共享包，可以包含代码、C++库、资源和配置文件，通过HSP可以实现代码和资源的共享。HSP不支持独立发布，而是跟随其宿主应用的APP包一起发布，与宿主应用同进程，具有相同的包名和生命周期。（又称shared library, 动态共享包）

42.鸿蒙常用的装饰器有哪些?

> @State 定义状态，当前组件能使用
>
> @Prop 父子组件通信（特点：子组件数据不能修改）
>
> @Link 父子组件通信（特点：子组件数据可以修改）
>
> @Observed 和 @ObjectLink 父子组件通信（特点：嵌套第二层数据修改可以达到响应式，之前方案不行）
>
> @Provide 和 @Consume 祖孙组件通信
>
> @Builder 和 @BuilderParam 父子组件通信，通信组件数据
>
> @Watch 监视数据的变化（第一次不会触发）

43.如何启动一个 ability?

> 通过startAbility启动

44.三层架构是什么?

> commons（公共能力层）：用于存放公共基础能力集合（如工具库、公共配置等）。commons层可编译成一个或多个HAR包或HSP包，只可以被products和features依赖，不可以反向依赖。
>
> features（基础特性层）：开发页面、组件（HAR包或HSP包）。
>
> products（产品定制层）：定义phone\pad两个ability，引用 features 的包和 commons 的包完成应用功能

45.优化内存有哪些方法?

> 1. 使用onMemoryLevel监听内存变化
> 2. 使用LRUCache优化ArkTS内存 例如：我们搜索租房列表可以无限加载租房数据，这样数据会越来越多，我们使用LRUCacheUtil来管理数据
> 3. 使用生命周期管理优化ArkTS内存 例如：aboutToDisappear中销毁订阅事件，清除定时器等
> 4.使用purgeable优化C++内存

46.蓝牙的基本操作

使用蓝牙需要先申请权限：ohos.permission.ACCESS_BLUETOOTH，使用ConnectivityKit中的对应的API进行开发。

> 传统蓝牙
>
> 传统蓝牙，可以通过connection 模块，进行开发，
>
> 可以扫描周边蓝牙设备、设置本机蓝牙扫描模式、查找已配对设备信息

> 数据可通过串口通信协议（SPP协议）传输。当两个设备间进行SPP通信交互时，依据设备功能的不同，可区分为客户端与服务端。
>
> 客户端通过设备地址、和UUID连接服务端，连接上可以通过socket.sppWrite写入数据到服务端。可以通过socket.on('sppRead')监听发送过来的数据。
>
> 服务端：1.创建服务端套接字，在蓝牙子系统中注册该UUID服务；2.监听客户端连接，socket.sppAccept(serverNumber）3.向客户端发送数据， socket.sppWrite；3.监听客户端发送过来的数据，socket.on('sppRead'

> 低功耗蓝牙：可以通过ConnectivityKit中的ble模块操作。分为广播流程和扫描流程。
>
> 1.广播流程
>
> - 通过ble.on('advertisingStateChange', onReceiveEvent);订阅广播
> - 通过 ble.startAdvertising启动广播
> - 通过 ble.stopAdvertising或者API11以后的ble.stopAdvertising();停止广播
>
> 2.扫描流程
>
> - 通过ble.createBleScanner()创建监听对象，通过bleScanner.on('BLEDeviceFind'）监听设备，api14以上可以通过ble.on('BLEDeviceFind', onReceiveEvent);监听设备。
> - 通过BLE扫描周边其他设备发出的BLE广播，a.通过ble.createBleScanner()创建ble扫描实例；b.过滤符合目标BLE广播报文的设备；c.通过bleScanner.startScan发起扫描，API14以后可以通过 ble.startBLEScan([scanFilter], scanOptions);发起扫描

47.Wlan无线局域网开发

包含三种模式：STA模式（工作站模式）、P2P模式（Wi-Fi直连）、AP模式（热点模式）

> STA模式即工作站模式，可以理解为某网络中的一个工作站即客户端。当某设备具备该功能时，它可以连到另外的一个路由网络中，如家用路由器，通常用于提供网络的数据上行服务
>
> P2P模式也为Wi-Fi Direct；Wi-Fi Direct 是一种点对点连接技术，它可以在两台 STA 之间直接建立 TCP/IP 链接，并不需要AP的参与；其中一台STA会起到传统意义上的AP的作用，称为Group Owner(GO)，另外一台station则称为Group Client(GC)，像连接AP一样连接到GO
>
> AP模式为加入无线局域网的成员设备（即客户端）提供下行数据业务，它提供以无线方式组建无线局域网WLAN，相当于WLAN的中心设备

P2P相关函数：

createGroup()	**创建群组。**
removeGroup()	删除群组。
startDiscoverDevices()	开始发现设备。
getP2pPeerDevices()	获取P2P对端设备列表信息。
p2pConnect()	执行P2P连接。
getP2pLinkedInfo()	获取P2P连接信息。
on(type: 'p2pPersistentGroupChange')	注册P2P永久组状态改变事件。
off(type: 'p2pPersistentGroupChange')	取消注册P2P永久组状态改变事件。
on(type: 'p2pPeerDeviceChange')	注册P2P对端设备状态改变事件。
off(type: 'p2pPeerDeviceChange')	取消注册P2P对端设备状态改变事件。
on(type: 'p2pConnectionChange')	注册P2P连接状态改变事件。
off(type: 'p2pConnectionChange')	取消注册P2P连接状态改变事件。