使用 npm 包和分包

1、使用 npm 包

1.1、小程序对 npm 的支持与限制

目前，小程序中已经支持使用npm 安装第三方包，从而来提高小程序的开发效率。但是，在小程序中使用
npm 包有如下 3 个限制：
① 不支持依赖于 Node.js 内置库的包
② 不支持依赖于浏览器内置对象的包

③ 不支持依赖于 C++ 插件的包
总结：虽然 npm 上的包有千千万，但是能供小程序使用的包却“为数不多”。

1.2、Vant Weapp

1.2.1、什么是 Vant Weapp

Vant Weapp 是有赞前端团队开源的一套小程序 UI 组件库，助力开发者快速搭建小程序应用。它所使用的是
MIT 开源许可协议，对商业使用比较友好。
官方文档地址 https://youzan.github.io/vant-weapp
扫描下方的小程序二维码，体验组件库示例：

1.2.2、安装 Vant 组件库

在小程序项目中，安装Vant 组件库主要分为如下 3 步：
① 通过 npm 安装（建议指定版本为@1.3.3）
② 构建 npm 包
③ 修改 app.json
详细的操作步骤，大家可以参考 Vant 官方提供的快速上手教程：
https://youzan.github.io/vant-weapp/#/quickstart#an-zhuang

1.2.3、使用 Vant 组件

安装完 Vant 组件库之后，可以在app.json 的 usingComponents 节点中引入需要的组件，即可在 wxml 中
直接使用组件。示例代码如下：

```
// app.json
"usingComponents": {
  "van-button": "@vant/weapp/button/index"
}

// 页面的 .wxml 结构
<van-button type="primary">按钮</van-button>
```

1.2.4、定制全局主题样式

Vant Weapp 使用 CSS 变量来实现定制主题。 关于 CSS 变量的基本用法，请参考 MDN 文档:
https://developer.mozilla.org/zh-CN/docs/Web/CSS/Using_CSS_custom_properties

1.2.5、定制全局主题样式

在 app.wxss 中，写入 CSS 变量，即可对全局生效:

```
/* app.wxss */
page {
  /* 定制警告按钮的背景颜色和边框颜色 */
  --button-danger-background-color: #C00000;
  --button-danger-border-color: #D60000;
}
```

所有可用的颜色变量，请参考Vant 官方提供的配置文件:
https://github.com/youzan/vant-weapp/blob/dev/packages/common/style/var.less

1.3、API Promise化

1.3.1、基于回调函数的异步 API 的缺点

默认情况下，小程序官方提供的异步 API 都是基于回调函数实现的，例如，网络请求的 API 需要按照如下的方
式调用:

```
wx.request({
  method: '',
  url: '',
  data: { },
  success: () => { }, // 请求成功的回调函数
  fail: () => { },    // 请求失败的回调函数
  complete: () => { } // 请求完成的回调函数
})
```

缺点：容易造成回调地狱的问题，代码的可读性、维护性差！

1.3.2、什么是 API Promise 化

API Promise化，指的是通过额外的配置，将官方提供的、基于回调函数的异步 API，升级改造为基于 Promise 的异步 API，从而提高代码的可读性、维护性，避免回调地狱的问题。

1.3.3、实现 API Promise 化

在小程序中，实现 API Promise 化主要依赖于 miniprogram-api-promise这个第三方的 npm 包。它的安装
和使用步骤如下：

```
npm install --save miniprogram-api-promise@1.0.4
```

```
// 在小程序入口文件中(app.js)，只需调用一次 promisifyAll() 方法，
// 即可实现异步 API 的 Promise 化
import { promisifyAll } from 'miniprogram-api-promise'

const wxp = wx.p = {}
// promisify all wx's api
promisifyAll(wx, wxp)
```

1.3.4、调用 Promise 化之后的异步 API

```
// 页面的 .wxml 结构
<van-button type="danger" bindtap="getInfo">vant按钮</van-button>

// 在页面的 .js 文件中，定义对应的 tap 事件处理函数
async getInfo() {
  const { data: res } = await wx.p.request({
    method: 'GET',
    url: 'https://www.escook.cn/api/get',
    data: { name: 'zs', age: 20 }
  })

  console.log(res)
},
```

2、 分包

2.1、基础概念

2.1.1、什么是分包

分包指的是把一个完整的小程序项目，按照需求划分为不同的子包，在构建时打包成不同的分包，用户在使用
时按需进行加载。

2.1.2、分包的好处

对小程序进行分包的好处主要有以下两点：

1. 可以优化小程序首次启动的下载时间
2. 在多团队共同开发时可以更好的解耦协作

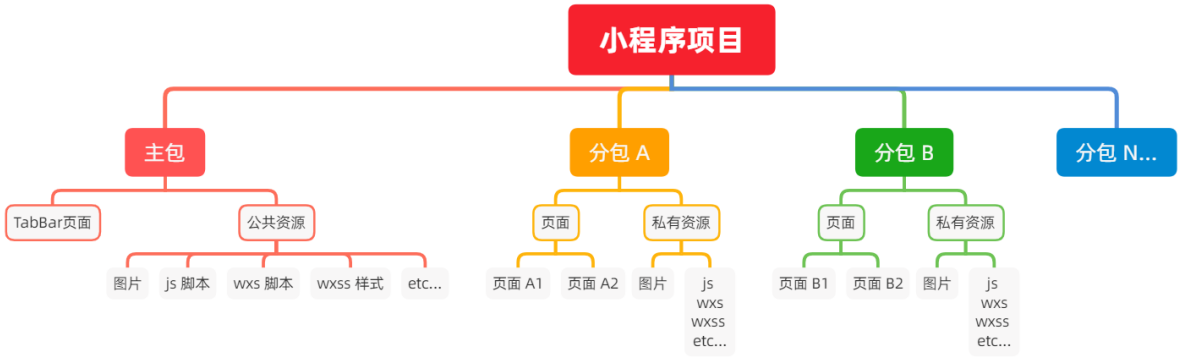
2.1.3、分包前项目的构成

分包前，小程序项目中所有的页面和资源都被打包到了一起，导致整个项目体积过大，影响小程序首次启动的
下载时间。

2.1.4、分包后项目的构成

分包后，小程序项目由1 个主包 + 多个分包组成：

1. 主包：一般只包含项目的启动页面或 TabBar 页面、以及所有分包都需要用到的一些公共资源
2. 分包：只包含和当前分包有关的页面和私有资源



2.1.5、分包的加载规则

① 在小程序启动时，默认会下载主包并启动主包内页面
　tabBar 页面需要放到主包中
② 当用户进入分包内某个页面时，客户端会把对应分包下载下来，下载完成后再进行展示
　非 tabBar 页面可以按照功能的不同，划分为不同的分包之后，进行按需下载

2.1.6、 分包的体积限制

目前，小程序分包的大小有以下两个限制：

1. 整个小程序所有分包大小不超过 16M（主包 + 所有分包）
2. 单个分包/主包大小不能超过 2M

2.2、使用分包

2.2.1、配置方法

小程序的目录结构

```
├── app.js
├── app.json
├── app.wxss
├── pages          // 主包的所有页面
|   ├── index
|   └── logs
├── packageA       // 第一个分包
|   └── pages      // 第一个分包的所有页面
|       ├── cat
|       └── dog
├── packageB       // 第二个分包
|   └── pages      // 第二个分包的所有页面
|       ├── apple
|       └── banana
└── utils
```

在 app.json 的 subpackages 节点中声明分包的结构

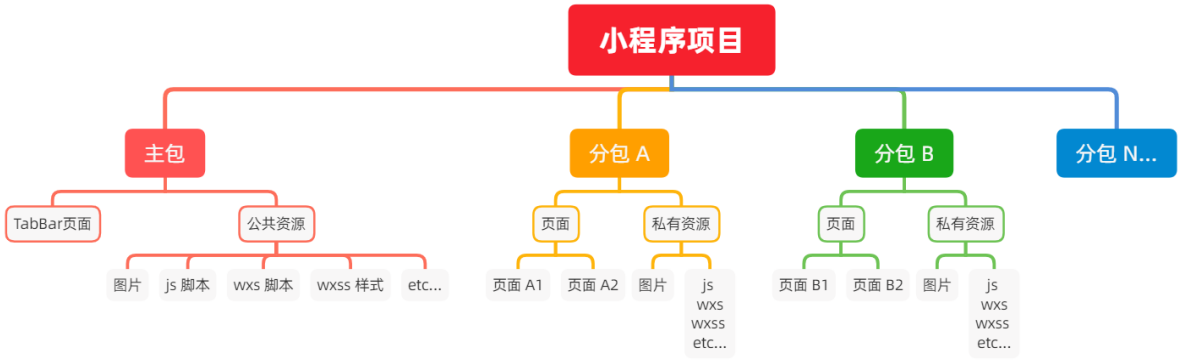
```
{
  "pages":[ // 主包的所有页面
    "pages/index",
    "pages/logs"
  ],
  "subpackages": [ // 通过 subpackages 节点，声明分包的结构
    {
      "root": "packageA", // 第一个分包的根目录
      "pages": [ // 当前分包下，所有页面的相对存放路径
        "pages/cat",
        "pages/dog"
      ]
    },{
      "root": "packageB", // 第二个分包的根目录
      "name": "pack2",    // 分包的别名
      "pages": [ // 当前分包下，所有页面的相对存放路径
        "pages/apple",
        "pages/banana"
      ]
    }
  ]
}
```

2.2.2、打包原则

① 小程序会按 subpackages 的配置进行分包，subpackages 之外的目录将被打包到主包中
② 主包也可以有自己的 pages（即最外层的 pages 字段）
③ tabBar 页面必须在主包内
④ 分包之间不能互相嵌套

2.2.3、引用原则

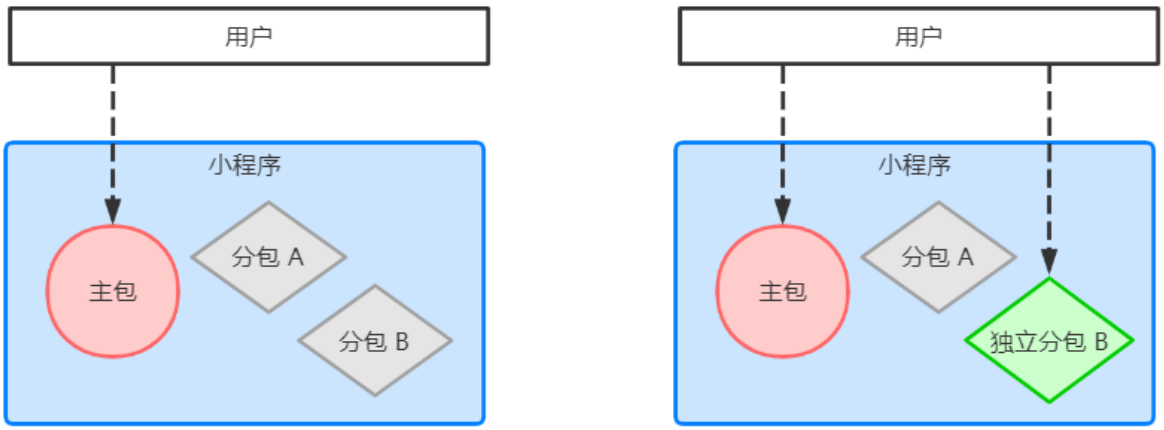
① 主包无法引用分包内的私有资源
② 分包之间不能相互引用私有资源
③ 分包可以引用主包内的公共资源



2.3、独立分包

2.3.1、什么是独立分包

独立分包本质上也是分包，只不过它比较特殊，可以独立于主包和其他分包而单独运行。



2.3.2、独立分包和普通分包的区别

最主要的区别：是否依赖于主包才能运行

1. 普通分包必须依赖于主包才能运行
2. 独立分包可以在不下载主包的情况下，独立运行

2.3.3、独立分包的应用场景

开发者可以按需，将某些具有一定功能独立性的页面配置到独立分包中。原因如下：

1. 当小程序从普通的分包页面启动时，需要首先下载主包
2. 而独立分包不依赖主包即可运行，可以很大程度上提升分包页面的启动速度
   注意：一个小程序中可以有多个独立分包。

2.3.4、独立分包的配置方法


小程序的目录结构
├── app.js
├── app.json
├── app.wxss
├── pages   // 主包的所有页面
| ├── index
| └── logs
├── moduleA // 普通分包
| └── pages
| ├── rabbit
| └── squirrel
├── moduleB // 独立分包
| └── pages
| ├── pear
| └── pineapple
└── utils

通过 independent 声明独立分包

```
{
  "pages": [
    "pages/index",
    "pages/logs"
  ],
  "subpackages": [
    {
      "root": "moduleA", // moduleA 为普通分包
      "pages": [
        "pages/rabbit",
        "pages/squirrel"
      ]
    }, {
      "root": "moduleB",
      "pages": [
        "pages/pear",
        "pages/pineapple"
      ],
      "independent": true // 通过此节点，声明当前 moduleB 分包为“独立分包”
    }
  ]
}
```

2.3.5、引用原则

独立分包和普通分包以及主包之间，是相互隔绝的，不能相互引用彼此的资源！例如：
① 主包无法引用独立分包内的私有资源
② 独立分包之间，不能相互引用私有资源
③ 独立分包和普通分包之间，不能相互引用私有资源
④ 特别注意：独立分包中不能引用主包内的公共资源

2.4、分包预下载

2.4.1、什么是分包预下载

分包预下载指的是：在进入小程序的某个页面时，由框架自动预下载可能需要的分包，从而提升进入后续分包
页面时的启动速度。

2.4.2、配置分包的预下载

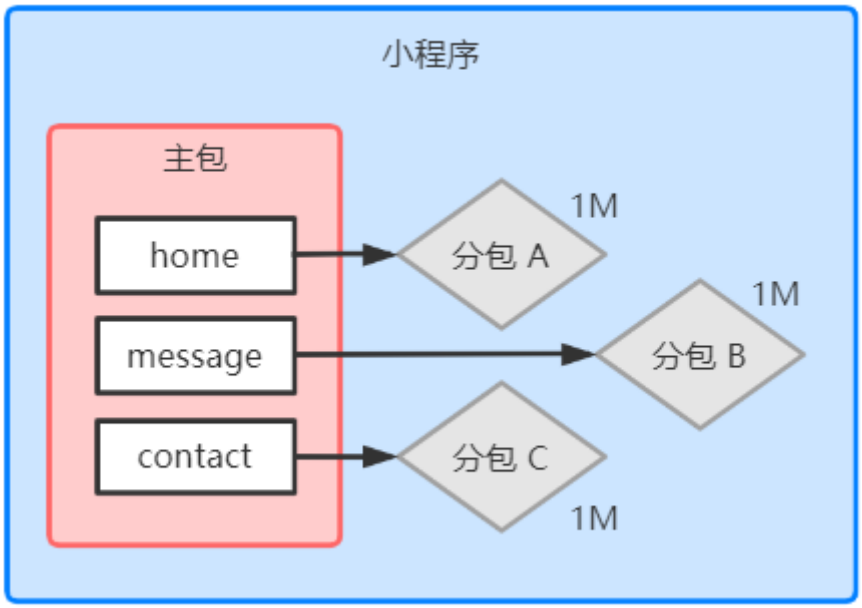
预下载分包的行为，会在进入指定的页面时触发。在 app.json 中，使用 preloadRule 节点定义分包的预下载
规则，示例代码如下：

```
{
  "preloadRule": { // 分包预下载的规则
    "pages/contact/contact": { // 触发分包预下载的页面路径
      // network 表示在指定的网络模式下进行预下载，
      // 可选值为：all（不限网络）和 wifi（仅 wifi 模式下进行预下载）
      // 默认值为：wifi
      "network": "all",
      // packages 表示进入页面后，预下载哪些分包
      // 可以通过 root 或 name 指定预下载哪些分包
      "packages": ["pkgA"]
    }
  }
}
```

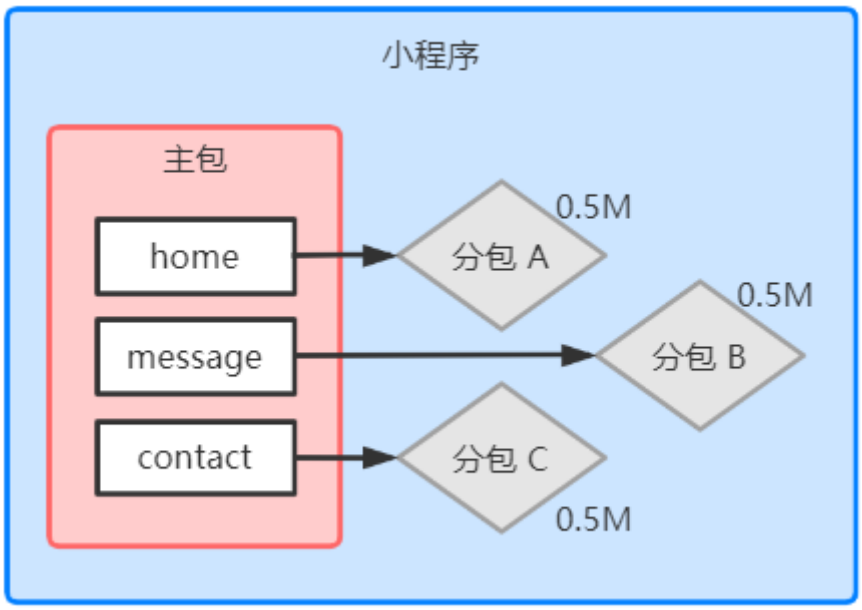
2.4.3、分包预下载的限制

同一个分包中的页面享有共同的预下载大小限额2M，例如：

不允许，分包 A+B+C 体积大于 2M



允许，分包 A+B+C 体积小于 2M



3、总结

① 能够知道如何安装和配置 vant-weapp 组件库
● 参考 Vant 的官方文档
② 能够知道如何对小程序的 API 进行 Promise 化
● 安装包、在 app.js 中进行配置
③ 分包的概念及配置