快应用
Quick App

快应用入门教程

——从认识到开发

快应用
Quick App

大纲

Contents

01 快应用简介

介绍快应用的产生背景、发展历史；技术架构、基本原理；优势、适合场景、优秀案例

02 开发、调试、发布

从环境准备、初始化工程、开发调试、发布上线几个方面介绍快应用开发的全流程

03 代码编写

介绍前端开发基础，快应用工程各部分的代码编写，注意事项

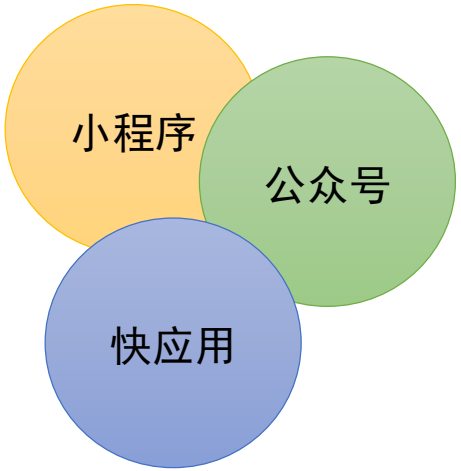
04 作业相关

围绕课程作业，对照 demo 工程，对其涉及的各项技术点进行详细讲解，减少大家的开发阻力

1.1 快应用产生背景

移动互联网发展新形势

- 超级 App 占据大部分流量
- 安装新应用意愿低
- 大量长尾需求存在
- 用户体验要求



1.2 什么是快应用

- 基于手机硬件平台的新型应用形态：无需安装、即点即用、体验良好
- 由主流手机厂商组成的快应用联盟联合制定研发标准、接口、能力等
- 一次开发，跨厂商联合分发
- 覆盖10亿设备
- 多场景入口、智能化场景入口
- ……

1.3 快应用的优点

× 功能有缺失

× 性能和体验较差

× 无留存能力

网页

VS

× 先安装后使用

× 互联互通程度低

× 版本碎片化

Native app

✓ 功能完备

✓ 原生应用体验

✓ 易于留存

✓ 即点即用

✓ 互联互通

✓ 无版本碎片

快应用

1.4 快应用的场景入口


全局搜索


小爱


智能助理

AI键


快应用中心


浏览器搜索

+

- Deeplink 唤起
- 网页跳转
- PUSH 唤起
- 短信
- 应用码
- 电视、音箱、车机等智能设备入口
- ……

1.4 快应用的典型案例

智能助理

菜鸟裹裹快应用

AI按键

微软小冰快应用

智能短信

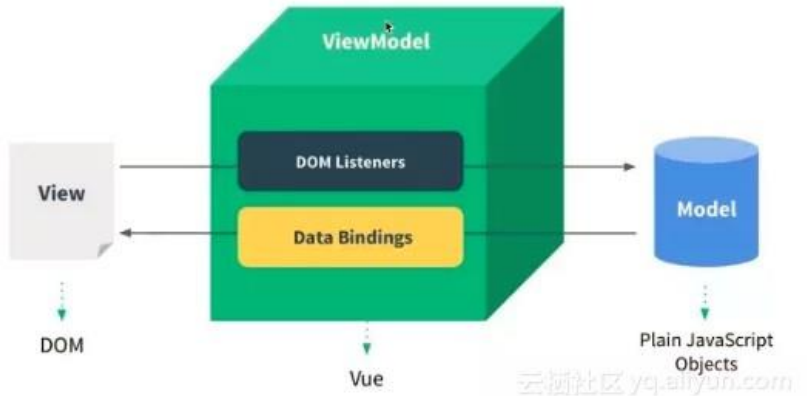
携程旅行快应用

开发技术栈

F(HTML/CSS/JS) = Native Page

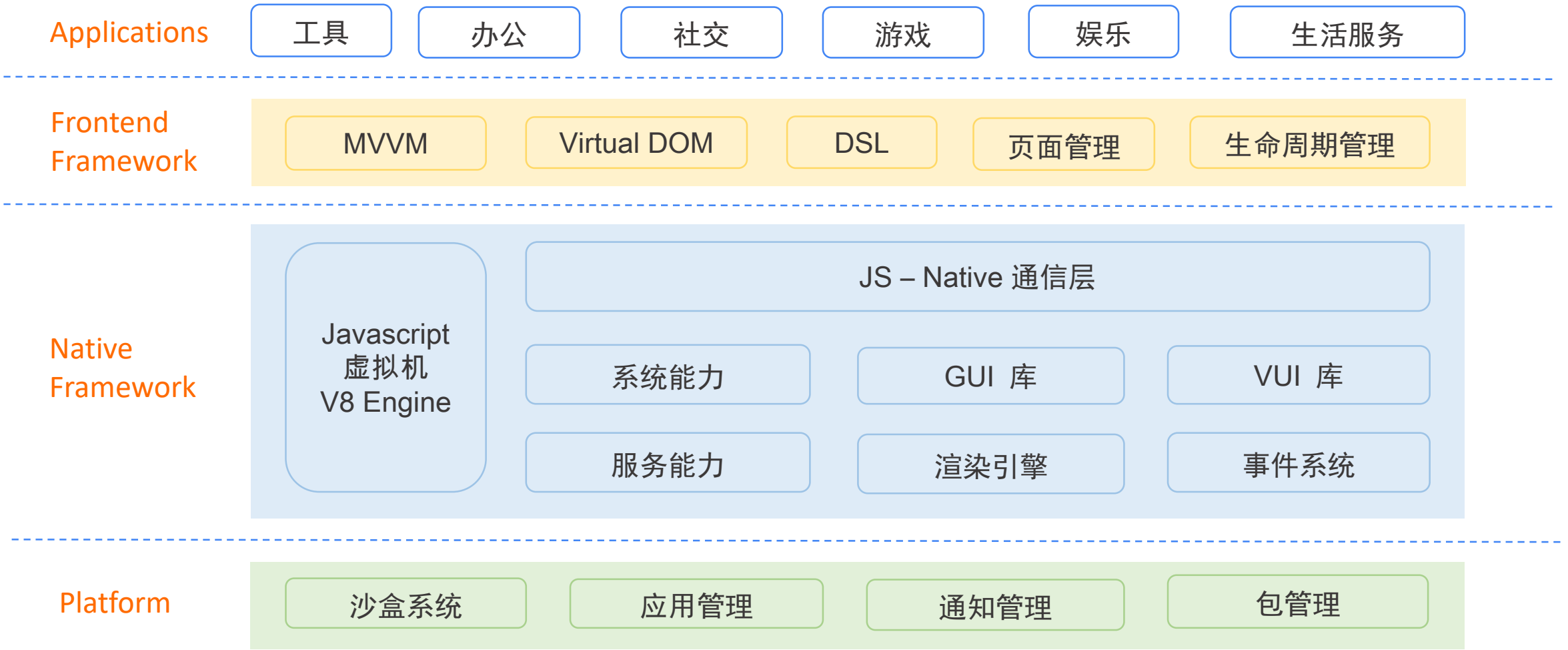
前端技术栈

- HTML5 + Native 标签
- Flexbox 布局
- ES 语法
- MVVM 框架

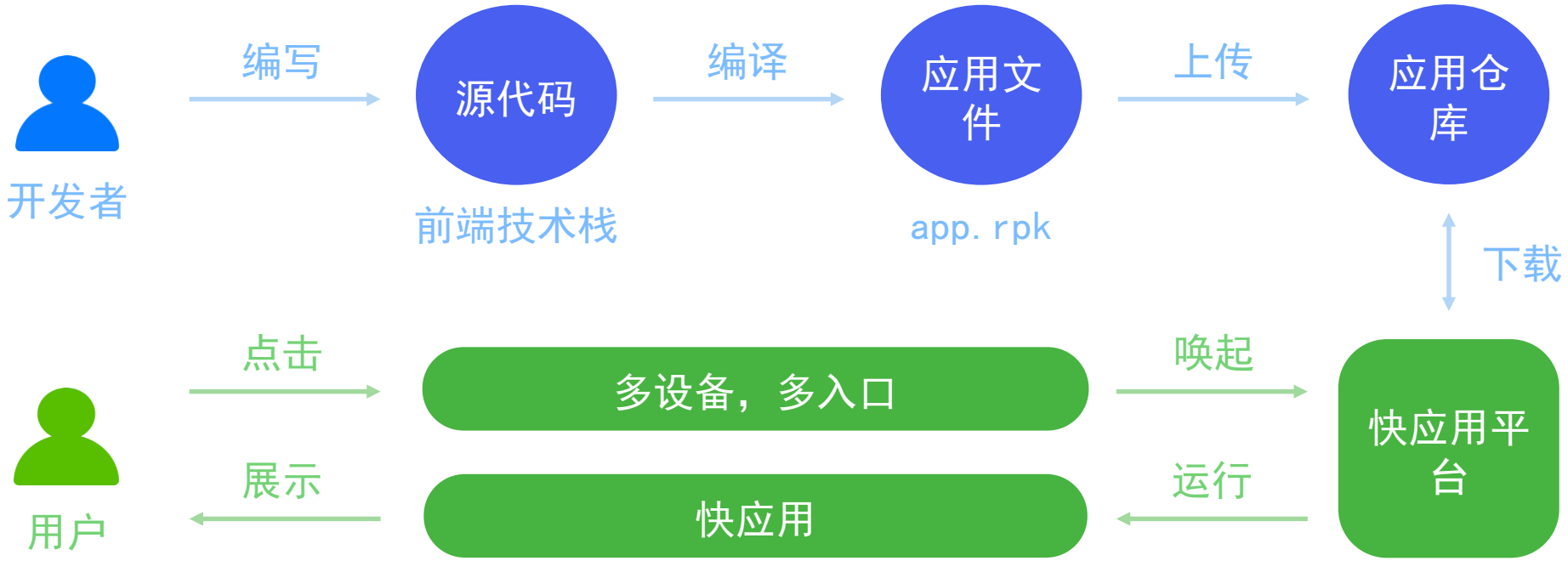


- ✓ 开发、运行效率高
- ✓ 学习成本低
- ✓ 代码可复用

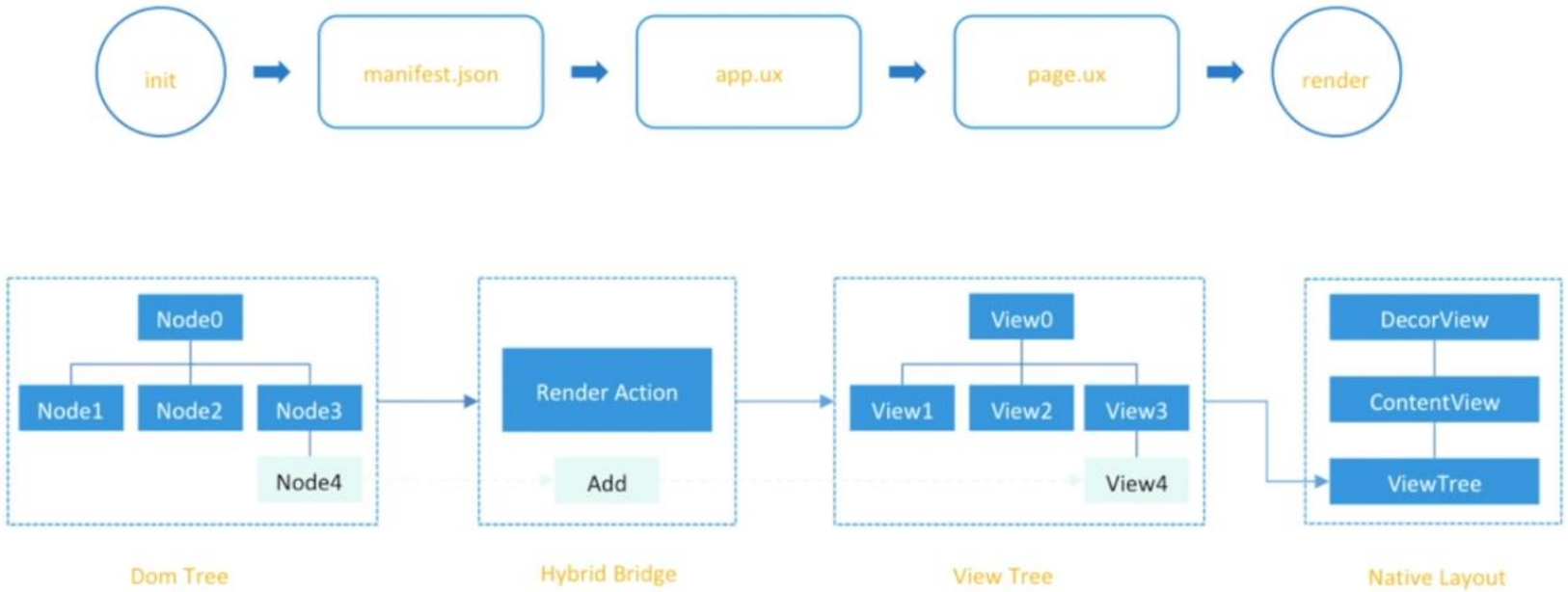
整体架构


Applications
工具
办公
社交
游戏
娱乐
生活服务
Frontend Framework
MVVM
Virtual DOM
DSL
页面管理
生命周期管理
Native Framework
Javascript 虚拟机 V8 Engine
JS – Native 通信层
系统能力
GUI 库
VUI 库
服务能力
渲染引擎
事件系统
Platform
沙盒系统
应用管理
通知管理
包管理

开发和分发模式


开发者
编写
源代码
前端技术栈
编译
应用文件
app. rpk
上传
应用仓库
下载
用户
点击
展示
多设备，多入口
快应用
唤起
运行
快应用平台

加载及渲染



原生渲染

- 功能强大
- 体验流畅
- 资源消耗少

安全机制

沙盒模型

进程、数据、权限隔离

签名机制

防篡改

包名唯一

与 APP 包名不同

权限管理

声明、弹窗确认

快应用
Quick App

大纲

Contents

01 快应用简介

介绍快应用的产生背景、发展历史；技术架构、基本原理；优势、适合场景、优秀案例

02 开发、调试、发布

从环境准备、初始化工程、开发调试、发布上线几个方面介绍快应用开发的全流程

03 代码编写

介绍前端开发基础，快应用工程各部分的代码编写，注意事项，以及常用工具库推荐

04 作业相关

围绕课程作业，对照 demo 工程，对其涉及的各项技术点进行详细讲解，减少大家的开发阻力

电脑

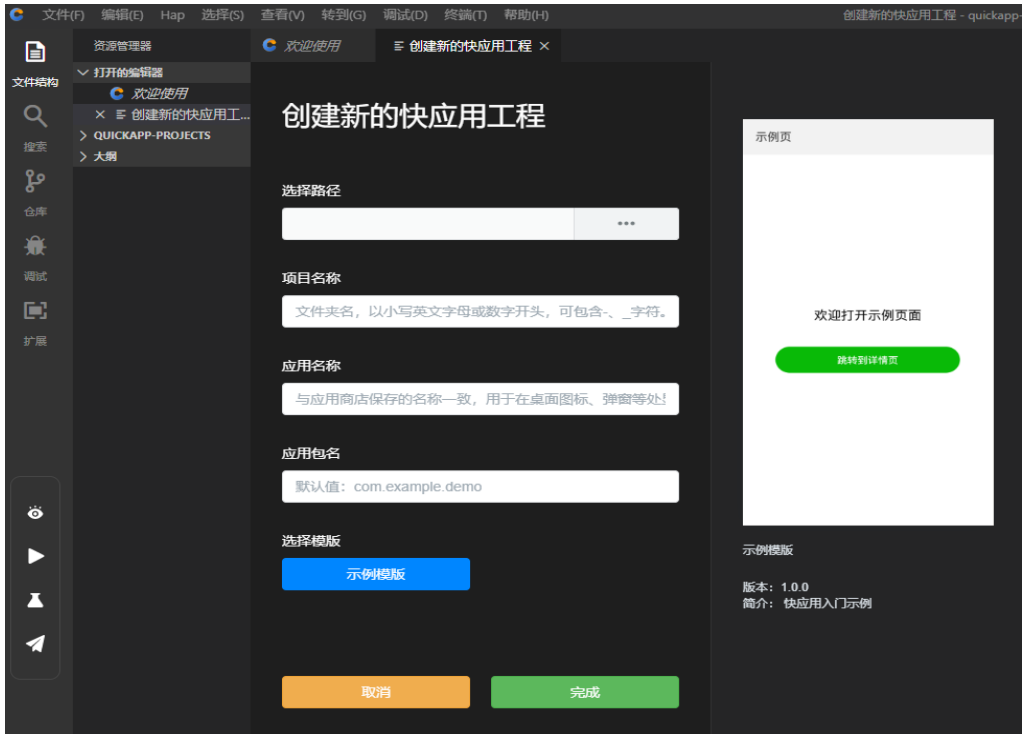
- 安装 Node.js https://nodejs.org/zh-cn/download/
- 全局安装 hap-toolkit npm install –g hap-toolkit
- 安装代码编辑器（vscode\webstorm\sublime ）或者快应用 IDE
- 默认浏览器设置为chrome

Android 手机

- 设置->系统->关于手机->版本号 连击三下 打开开发者模式，返回系统出现开发人员选项，打开USB调试
- 安装调试器 官网下载到手机然后安装 https://www.quickapp.cn/docCenter/post/69
- 安装快应用框架预览版 官网下载到手机然后安装
- 设置->应用->权限管理->快应用预览版->打开麦克风权限

2.2 工程初始化和编译

使用快应用IDE



使用命令行工具

- 在代码编辑器里打开一个空文件夹，然后打开终端执行以下命令
- 初始化：hap init <ProjectName>
- 安装依赖：npm install
- 手动编译：npm run build
- 自动编译：npm run watch

编译打包成功后，项目根目录下会生成文件夹：build、dist

•build：临时产出，包含编译后的页面 js，图片等

•dist：最终产出，包含 rpk 文件。其实是将 build 目录下的资源打包压缩为一个文件，后缀名为 rpk，这个rpk文件就是项目编译后的最终产出

2.4 rpk 调试

非IDE方式

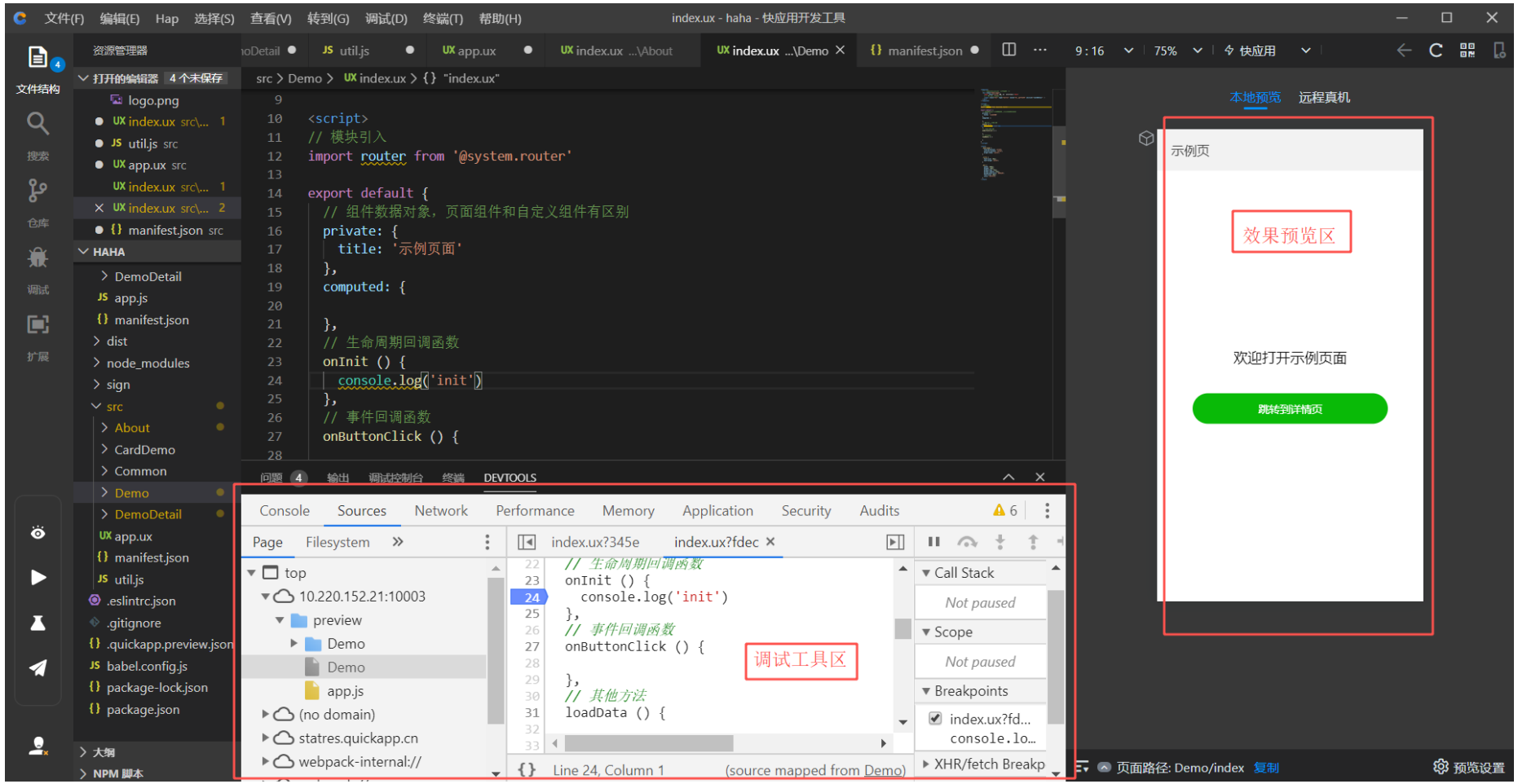
扫码安装

PC 开发环境启动 hap server --watch ⇨ 手机调试器扫码 ⇨ 手机调试器打开该快应用界面 ⇨ 代码改动并保存，界面自动更新

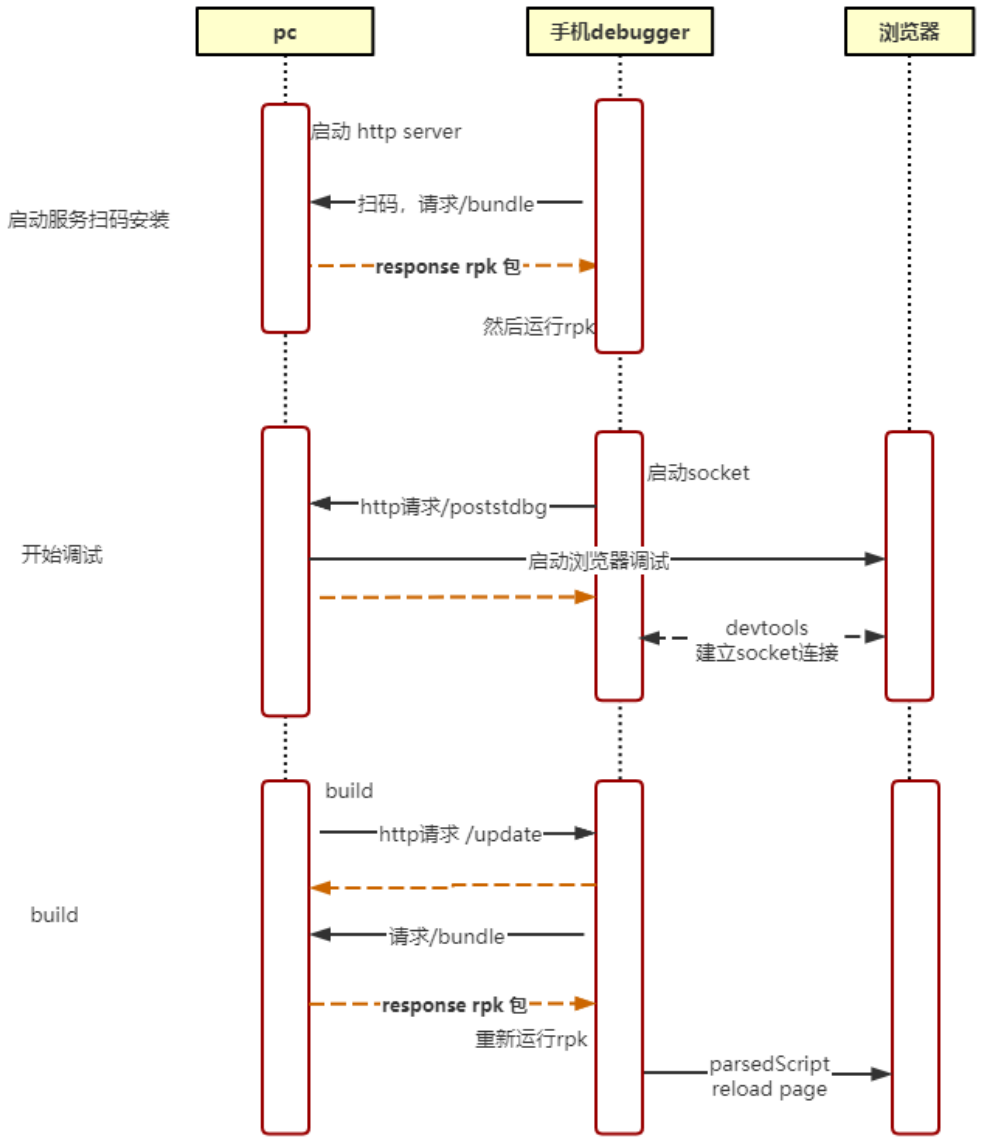
调试

调试器选择运行平台为预览版 ⇨ 调试器点击开始调试 ⇨ PC 浏览器打开模拟界面 ⇨ devtools 中查看代码、样式、log、断点调试

IDE方式



2.5 rpk 调试原理



- pc端开发环境：启动server，负责打包和传送 rpk
- 手机端debugger：接收 rpk 并运行渲染预览
- pc端浏览器： web 模拟显示快应用最终效果，可以像 web 一样在 devtool 里进行 debug

2.6 快应用打包发布

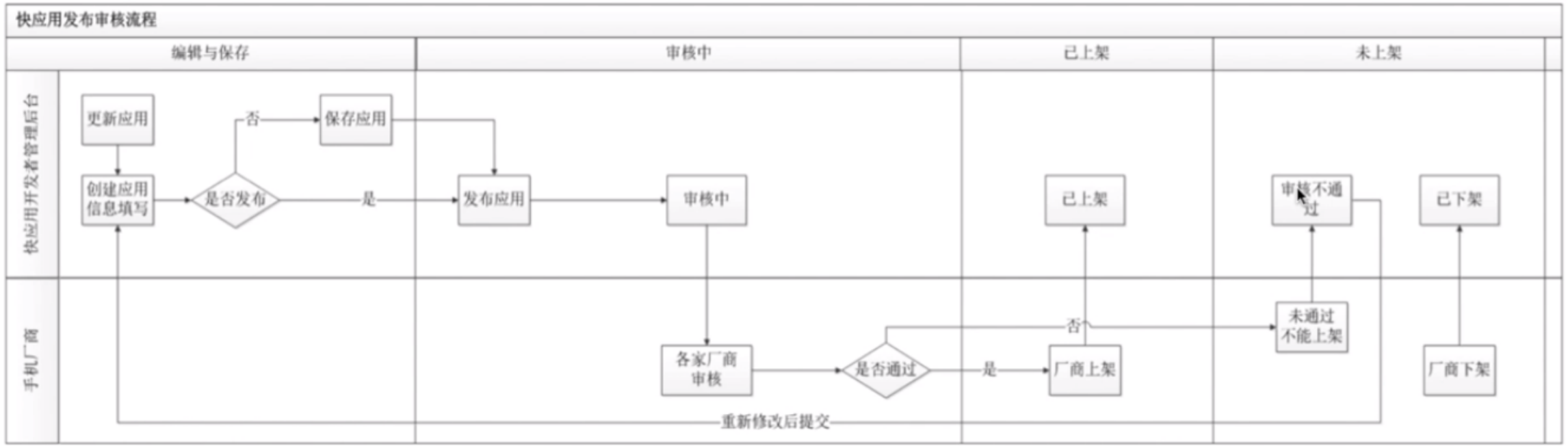
打包-命令行方式

工程根目录 npm run build ⇨ Sign目录下新建release目录并生成签名文件 ⇨ 给包签名 npm run release ⇨ 联盟官网注册并提交

打包-IDE 方式

2.6 快应用打包发布

上传发布



快应用
Quick App

大纲

Contents

01 **快应用简介**

介绍快应用的产生背景、发展历史；技术架构、基本原理；优势、适合场景、优秀案例

02 **开发、调试、发布**

从环境准备、初始化工程、开发调试、发布上线几个方面介绍快应用开发的全流程

03 **代码编写**

介绍前端开发基础，快应用工程各部分的代码编写，注意事项，以及常用工具库推荐

04 **作业相关**

围绕课程作业，对照 demo 工程，对其涉及的各项技术点进行详细讲解，减少大家的开发阻力

HTML

- 用来描述网页的标记语言
- HTML 文档即网页
- HTML 文档包括 HTML 标签和文本
- 浏览器读取 HTML 文档，并以网页的形式显示它们

```
<html>
<body>

<h1>我的第一个标题</h1>

<p>我的第一个段落。</p>

</body>
</html>
```

- 快应用用类 HTML 语法描述界面
- 标签 -> 组件，每种组件有特定的属性和事件，详见“官网->开发文档->框架”

```
<div class="demo-page">
  <text class="title">文本内容</text>
</div>
```

CSS

- 样式定义如何显示 HTML 元素
- 有丰富的选择器，定位要描述的 HTML 标签
- 有丰富的属性和规定的属性值
- 可以写在作为 style 属性的值写在 HTML 内部，也可以写在外部
- 某个标签的所有所有样式定义有优先级、覆盖、叠加等，需要样式计算

- 快应用里使用 CSS 进行组件的样式描述，出于性能原因只实现了 CSS 的子集，详见“官网->开发文档->教程-页面样式与布局”
- border-box 模型和 flex 布局

```
<div style="background-color: #f6f6f6;">
  <text>文本内容</text>
</div>
```

```
.demo-page {
  flex-direction: column;
  align-items: center;
}

/* 应用图标 */
#icon {
  margin-top: 90px;
  width: 134px;
  height: 134px;
  border-radius: 10px;
  border: 1px solid #8d8d8d;
}
```

Javascript

- 网页中的脚本语言，运行在浏览器中的解释型的编程语言
- 脚本在网页中运行，可以改变网页呈现的内容，例如控制 HTML 标签的样式、显示与否、动态修改 HTML 标签和文本内容
- 动态语言、函数式语言
- 标准在不断地完善和增加，有非常多特性，常见异步编程

- 快应用里完全使用 javascript 进行逻辑编程，注意不能使用浏览器里具有的 DOM api
- 可以使用 ES5/ES6 进行代码编写

快应用工程代码结构

```
├── sign                        rpk包签名模块
│   └── debug                   调试环境
│       ├── certificate.pem     证书文件
│       └── private.pem         私钥文件
├── src
│   ├── Common                  公用的资源和组件文件
│   │   └── logo.png            应用图标
│   ├── Demo                    页面目录
│   │   └── index.ux            页面文件，可自定义页面名称
│   ├── app.ux                  APP文件，可引入公共脚本，暴露公共数据和方法等
│   └── manifest.json           项目配置文件，配置应用图标、页面路由等
└── package.json                定义项目需要的各种模块及配置信息
```

- 其他js文件，例如tools.js
- 自定义组件
- 其他静态资源，如图片、字体文件等
- babel.config.js/build/dist/node_modules/.eslintrc.json/README.md
- ……

3.2 工程整体结构讲解

manifest.json

详见“官网->开发文档->框架->manifest 文件”

- 定了应用的基本信息：名称\版本\报名\最小支持平台版本等
- 声明了应用会使用的一些 native 功能接口
- 页面路由信息，编译打包时会打包此处声明的页面
- UI 显示的相关配置
- 全局系统配置和全局数据

组件

详见“官网->开发文档->框架->组件”

- 一个封装了UI模板、数据、逻辑的代码块，达到代码复用、提升效率的目的
- 使用时可以传入参数，定义属性，或者自定义部分代码，来实现不一样的功能
- 组件之间可以进行通信
- 页面可以看作是页面组件
- template 标签是快应用框架提供的组件，明细可参考https://doc.quickapp.cn/widgets/common-events.html
- 快应用还提供了实现自定义组件和动态组件的机制

3. 3 ux 文件讲解

页面或自定义组件是用 ux 文件编写的

3.3 ux 文件讲解

<template>
定义布局与组件

<style>
定义样式

```
<template>
  <!-- template里只能有一个根节点 -->
  <div class="demo-page">
    <text class="title">欢迎打开{{title}}</text>
    <!-- 点击跳转详情页 -->
    <input class="btn" type="button" value="跳转到详情页" onclick="routeDetail" />
  </div>
</template>

<style>
  .demo-page {
    flex-direction: column;
    justify-content: center;
    align-items: center;
  }

  .title {
    font-size: 40px;
    text-align: center;
  }

  .btn {
    width: 550px;
    height: 86px;
    margin-top: 75px;
    border-radius: 43px;
    background-color: #09ba07;
    font-size: 30px;
    color: #ffffff;
  }
</style>
```

3.3 ux 文件讲解

<template>
定义布局与组件

<script>
定义逻辑
数据、方法
事件响应
接口等

```
<template>
  <!-- template里只能有一个根节点 -->
  <div class="demo-page">
    <text class="title">欢迎打开{{title}}</text>
    <!-- 点击跳转详情页 -->
    <input class="btn" type="button" value="跳转到详情页" onclick="routeDetail" />
  </div>
</template>

<script>
import router from '@system.router'

export default {
  // 页面级组件的数据模型，影响传入数据的覆盖机制：private内定义的属性不允许被覆盖
  private: {
    title: '示例页面'
  },
  routeDetail () {
    // 跳转到应用内的某个页面，router用法详见：文档->接口->页面路由
    router.push ({
      uri: '/DemoDetail'
    })
  }
}
</script>
```

Template 模板编写

对照“官网->开发文档->框架->template模板”讲解
https://doc.quickapp.cn/framework/template.html

Style 样式编写

对照“官网->开发文档->框架->style样式”讲解
https://doc.quickapp.cn/framework/style-sheet.html

3. 4 javascript 逻辑编写

javascript 代码存在于以下文件中

- 组件 ux 文件中的组件级脚本逻辑
- app.ux 文件中的应用全局逻辑
- 其他自定义的脚本文件

```
<script>
// 模块引入
import router from '@system.router'
import tools from './tools'

export default {
  // 组件数据对象，页面组件和自定义组件有区别
  private: {
    title: '示例页面'
  },
  computed: {

  },
  // 生命周期回调函数
  onInit () {

  },
  // 事件回调函数
  onButtonClick () {

  },
  // 其他方法
  loadData () {

  }
}
</script>
```

模块引入和使用，可遵循 CommonJS 模块规范或 ES6 模块语法

1. 模块定义——按需要新建 js 文件，编写逻辑、导出

```
function showMenu () {···
}

function createShortcut () {···
}

export default {
  showMenu,
  createShortcut
}
```

CommonJS 语法：
require('./util').showMenu()

2. 在需要使用该模块的文件中引入(一般在脚本的最前面引入模块)

```
import util from './util'
```

3. 在代码中使用

```
util.showMenu()
```

接口声明和使用

1. manifest.json 文件中申明接口

```
"features": [
  {
    "name": "system.router"
  }
],
```

2. 在 javascript 代码中引入接口模块

```
import router from '@system.router'
```

3. 在 javascript 中使用接口提供的 api

```
router.push ({
  uri: '/DemoDetail'
})
```

快应用提供了丰富的接口，包括各种系统能力、数据和文件的存储获取、三方服务等，详细内容可查询：

https://doc.quickapp.cn/features/system/app.html

组件的数据模型

页面组件的数据模型：

- public：用来定义和接收从应用外部传入的数据
- protected：用来定义和接收从应用内部跳转到该页面时传入的数据
- private：页面内部数据，不会被覆盖
- computed：计算属性，其值是由其他数据对象通过方法计算返回的值，实时根据其依赖数据的变化而变化

自定义组件的数据模型：

- data：组件内部数据
- props：使用组件时，可以从外部传入的属性值
- computed:计算属性，其值是由其他数据对象通过方法计算返回的值，实时根据其依赖数据的变化而变化

生命周期

组件或者应用的生命周期，指的是其自身的一些函数，这些函数在特殊的时间点或遇到一些特殊的框架事件时被自动触发。

页面的主要生命周期：

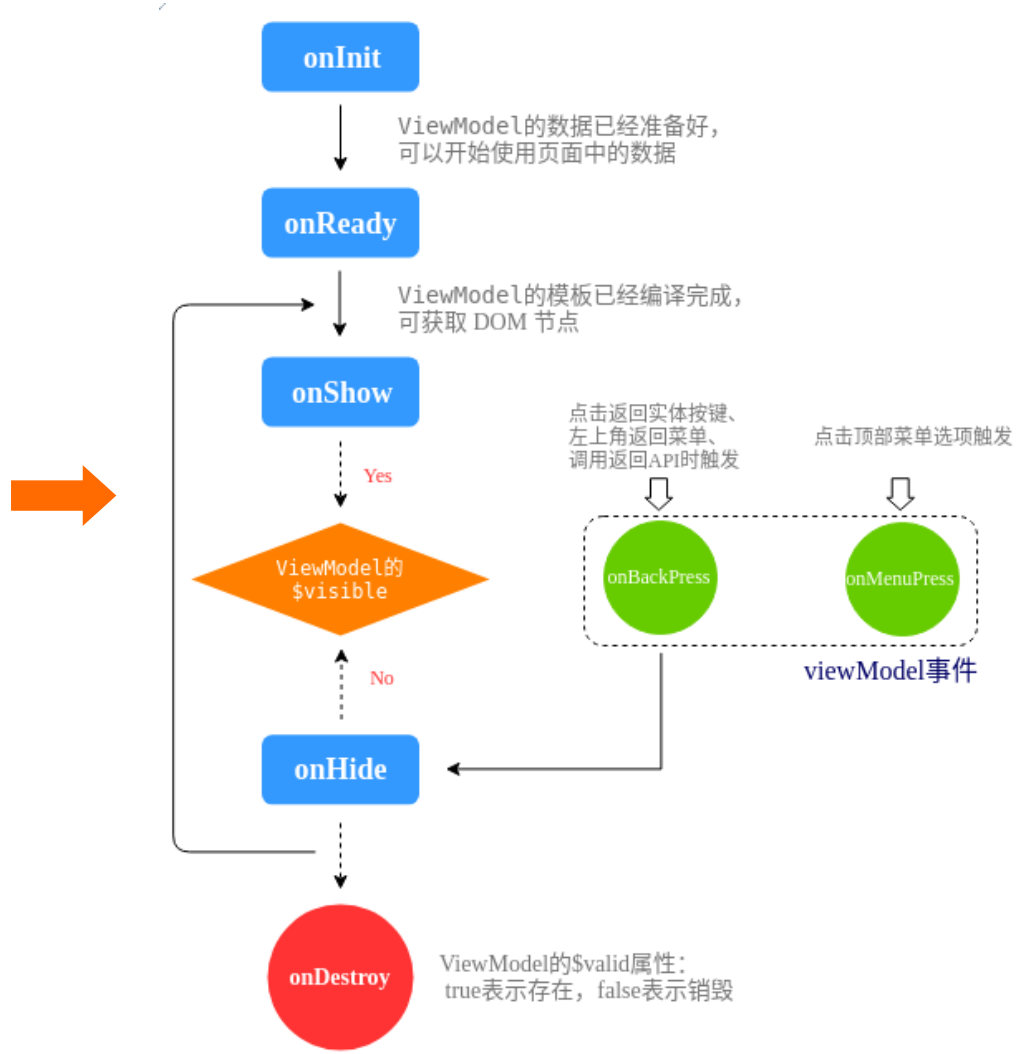
onInit -> onReady -> onShow -> onHide -> onDestroy

自定义组件的主要生命周期：

onInit -> onReady -> onDestroy

app的主要生命周期：

onCreate-> onShow -> onHide -> onDestroy



定义和使用全局方法和数据

1. 在 app.ux 中定义

```
<script>
/**
* 应用级别的配置，供所有页面公用
*/
import util from './util'

export default {
  showMenu: util.showMenu,
  createShortcut: util.createShortcut,
  data: 'test'
}
</script>
```

2. 在其他地方通过 this.$app 使用，$app 是挂载到组件上的代表应用全局的对象

```
this.$app.showMenu()
console.log(this.$app.$def.data)
```

代码调试

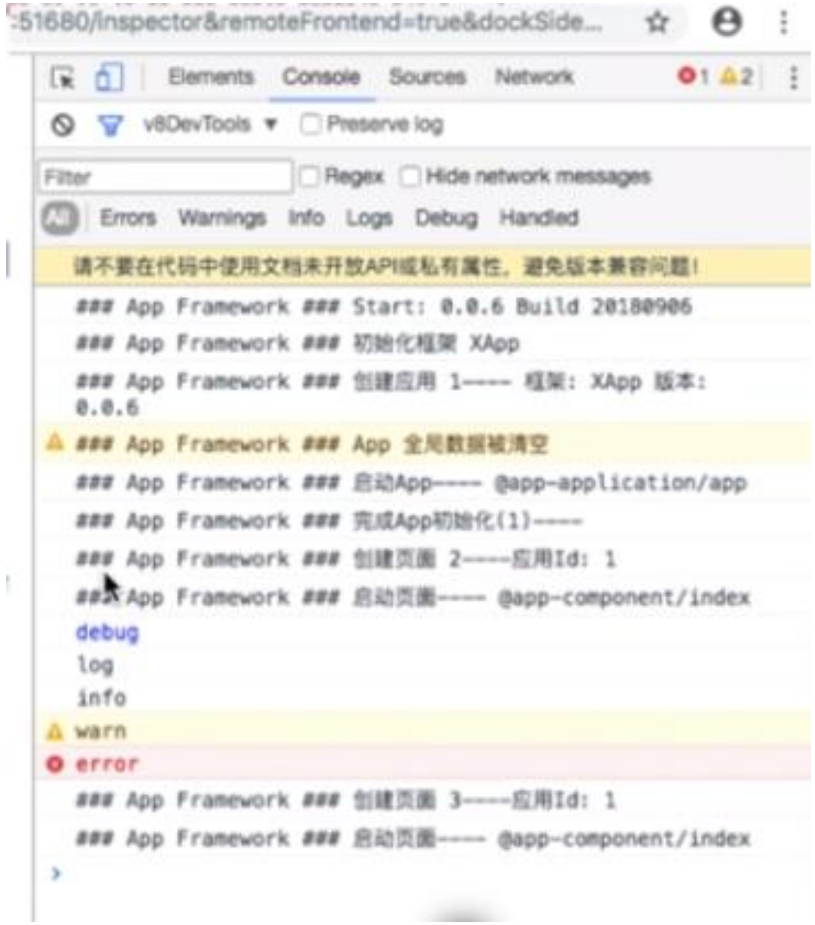
1. manifest.json 文件中修改日志等级

```
"config": {
  "logLevel": "debug"
},
```

2. 在 javascript 代码中输出日志

```
onInit () {
  // 修改页面数据
  this.name = '首页'
  console.log(this.name)
},
```

3. 查看日志：调试时在 chrome 浏览器的 devtools 中查看输出日志进行验证



快应用
Quick App

大纲

Contents

01 **快应用简介**

介绍快应用的产生背景、发展历史；技术架构、基本原理；优势、适合场景、优秀案例

02 **开发、调试、发布**

从环境准备、初始化工程、开发调试、发布上线几个方面介绍快应用开发的全流程

03 **代码编写**

介绍前端开发基础，快应用工程各部分的代码编写，注意事项，以及常用工具库推荐

04 **作业相关**

围绕课程作业，对照 demo 工程，对其涉及的各项技术点进行详细讲解，减少大家的开发阻力

4.1 作业

实现一个 todo 备忘录的快应用，在我们给出的 demo 的基础上去增强，demo 已经实现了备忘录增加、删除、状态改变功能

- 必须实现的功能：
  - 使用 kaldi 进行语音输入待办事项
  - 支持修改待办事项
  - 给待办时间添加完成时间
- 选择实现的功能：
  - 样式优化、动态效果
  - 从手机本地存储、读取数据
  - 到 deadline 了进行震动提醒
  - 增加一个统计页面，对待办事项进行统计
  - 状态从 to do / done 变成 to do / doing / done

- 评分标准
  - 功能都实现了就是60分
  - 考勤、提问附加 5 分
  - 选择实现的功能实现一个就加5分
  - 代码良好酌情加 1-10 分
  - 额外实现的功能每个加 5分

使用 todos demo 工程进行演示和讲解

使用 kaldi demo 工程进行演示和讲解

4.4 本地存储

使用 storage 接口可以将数据存储在手机本地，除非卸载应用或者手动清除，数据会一直存在

1. Manifest.json 中声明接口

```
"features": [
  { "name": "system.storage" }
],
```

2. 代码中导入接口模块

```
import storage from '@system.storage'
// 另一种导入方式
const storage = require('@system.storage')
```

3. 使用 api 进行数据的存储\获取\删除\清除

```
storage.set({
  key: 'A1',
  value: 'V1'
})
const val = storage.get({
  key: 'A1'
})
```

4.5 列表组件的使用

在页面中实现长列表或者屏幕滚动等效果时，为了得到流畅的列表滚动体验，推荐使用list组件

```
<list class="page-main">
  <block for="{{listData}}">
    <list-item type="item">
      <page-main-item item="{{$item}}" idx="{{$idx}}"></page-main-item>
    </list-item>
  </block>
</list>
```

路由就是页面的访问路径，类似于 web 页面的 url：https://quickapp.cn?a=1

从外部访问快应用某个页面，需要访问如下deeplink：

- http://hapjs.org/app/<package>/[path][?key=value]
- https://hapjs.org/app/<package>/[path][?key=value]
- hap://app/<package>/[path][?key=value]

应用内路由声明和跳转

```
"router": {
  "entry": "home",
  "errorPage": "ErrorPage",
  "pages": {
    "ErrorPage": {
      "component": "index"
    },
    "home": {
      "component": "index"
    }
  }
},
```

manifest.json 文件中声明

```
import router from '@system.router'

router.push({
  uri: '/about',
  params: {
    a: '1'
  }
})
```

跳转到另一个页面

```
public: {
  a: '' // 允许被应用外部页面请求传递的数据覆盖
},
protected: {
  a: '' // 允许被应用内部页面请求传递的数据覆盖
},
onInit () {
  // js中输出页面传递的参数
  console.info('key: ' + this.a)
}
```

页面内接收参数

谢谢！