

51CTO学院 丛书

异步图书  
www.epubit.com

- 书中示例以华为**智能手表**为运行载体
- 以流行的**JavaScript**为开发语言
- 提供整个应用开发项目的完整**源代码**



# 鸿蒙

张荣超◎著

## 应用开发实战



中国工信出版集团



人民邮电出版社  
POSTS & TELECOM PRESS

异步图书  
51CTO学院丛书

鸿蒙应用开发实战

张荣超 著

人民邮电出版社

北京

## 图书在版编目 (CIP) 数据

鸿蒙应用开发实战 / 张荣超著. — 北京: 人民邮电出版社, 2021.1

ISBN 978-7-115-55287-7

I. ①鸿... II. ①张... III. ①分布式操作系统—系统开发 IV. ①TP316.4

中国版本图书馆CIP数据核字 (2020) 第223651号

---

◆ 著 张荣超

责任编辑 傅道坤

责任印制 王郁 焦志炜

◆ 人民邮电出版社出版发行 北京市丰台区成寿寺路11号

邮编 100164 电子邮件 315@ptpress.com.cn

网址 <http://www.ptpress.com.cn>

大厂回族自治县聚鑫印刷有限责任公司印刷

◆ 开本: 800×1000 1/16

印张: 14.5

字数: 329千字 2021年1月第1版

印数: 1-2500千字 2021年1月河北第1次印刷

---

定价: 79.00元

读者服务热线: (010)81055410 印装质量热线: (010)81055316

**反盗版热线：(010)81055315**

**广告经营许可证：京东市监广登字20170147号**

# 目录

封面

扉页

版权信息

内容提要

关于作者

致谢

前言

第1章 鸿蒙操作系统简介

1.1 1+8+N全场景

1.2 分布式

1.3 小结

第2章 项目准备工作

2.1 搭建开发环境

2.2 Hello World

第3章 呼吸训练实战项目

3.1 任务1：在主页面中添加一个按钮并响应其单击事件

3.2 任务2：添加训练页面并实现其与主页面之间的相互跳转

3.3 任务3：验证应用和每个页面的生命周期事件

3.4 任务4：在主页面中显示logo和两个选择器

- 3.5 任务5：指定选择器的默认选中项并获取选中项的值
- 3.6 任务6：将主页面中选择器的值传递到训练页面
- 3.7 任务7：修改主页面和训练页面中按钮的文本及样式
- 3.8 任务8：在训练页面显示总共需要坚持的秒数
- 3.9 任务9：在训练页面倒计时显示再坚持的秒数
- 3.10 任务10：再坚持的秒数在倒计时结束时隐藏显示的文本
- 3.11 任务11：在训练页面根据呼吸节奏交替显示“吸气”和“呼气”
- 3.12 任务12：每次吸气或呼气时都实时显示进度百分比
- 3.13 任务13：每次吸气或呼气时logo都顺时针转动一周
- 3.14 任务14：添加倒计时页面并实现由主页面向其跳转
- 3.15 任务15：在倒计时页面进行训练指引的3秒倒计时
- 3.16 任务16：3秒倒计时结束后跳转到训练页面并传递主页面的数据
- 3.17 任务17：呼吸训练结束后右滑查看训练报告
- 3.18 任务18：将第1个训练报告页面的标题修改为压力占比
- 3.19 任务19：在压力占比页面的标题下方显示压力分类的列表
- 3.20 任务20：在压力分类的右边显示对应的压力占比
- 3.21 任务21：在每个列表项的下方显示压力占比的进度条
- 3.22 任务22：添加第2个训练报告页面并响应滑动事件
- 3.23 任务23：在第2个训练报告页面中显示除心率曲线之外的所有内容
- 3.24 任务24：在心率曲线页面中显示绘制的心率曲线

- 3.25 任务25：添加第3个训练报告页面并响应滑动事件
- 3.26 任务26：在第3个训练报告页面中显示除活动分布图之外的所有内容
- 3.27 任务27：在今日活动分布页面中显示绘制的今日活动分布图
- 3.28 任务28：添加第4个训练报告页面并响应滑动事件
- 3.29 任务29：在第4个训练报告页面中显示除压力分布图之外的所有内容
- 3.30 任务30：在压力分布页面中显示绘制的压力分布图
- 3.31 任务31：添加第5个训练报告页面并响应滑动事件
- 3.32 任务32：在第5个训练报告页面中显示除弧形和星号之外的所有内容
- 3.33 任务33：在最大摄氧量页面显示绘制的弧形
- 3.34 任务34：在最大摄氧量界面的对应弧形和角度上显示星号
- 3.35 任务35：添加学习交流联系方式页面并响应滑动事件
- 3.36 任务36：在学习交流联系方式页面中显示二维码并完成项目收尾工作

## 内容提要

本书详细完整地介绍了在HarmonyOS（鸿蒙操作系统）2.0上开发一个呼吸训练App的全部工程。

本书分为3章，内容涵盖了鸿蒙操作系统的简单介绍、开发鸿蒙App项目的准备工作，以及为鸿蒙操作系统开发一个呼吸训练App的全过程。本书采用项目导向和任务导向的方式讲解，分成36个任务，每个任务都分成3部分——运行效果、实现思路、代码详解。本书手把手地对编写的每一行代码进行讲解，确保读者看完本书后，能做出一个完整的项目。

本书适合对在鸿蒙系统上开发应用程序感兴趣的读者阅读学习。



## 关于作者

**张荣超**，华为公司官方认证的首批HarmonyOS（鸿蒙操作系统）课程开发人员，曾就职于HTC、联想、阿里巴巴，先后担任过资深软件开发工程师、项目经理、产品技术主管等职位。他是51CTO学院的金牌讲师、Sun公司认证的Java工程师和Java Web工程师，以及Scrum联盟认证的敏捷项目管理专家。此外，他还是在线知名系列课程《图解Python》的作者。

## 致谢

感谢我的家人，为我完成本书给予了无尽的支持、爱与奉献。

感谢51CTO鸿蒙技术社区的大力引荐，让我有机会能在第一时间参与鸿蒙技术的教育培训和课程开发。感谢51CTO鸿蒙技术社区的宋佳宸、王文文、王雪燕、赵克衡，在我学习鸿蒙系统、开发鸿蒙课程以及推广鸿蒙课程的整个过程中提供了大力的支持和帮助。

感谢华为公司的于小飞和谭景盟，在我录制鸿蒙课程的时候提供了大力的支持和帮助，为本书的顺利完成奠定了坚实的基础。

感谢华为公司编程语言实验室的项目经理王学智，为本书部分技术要点的写作提供了宝贵建议。

感谢学员负云龙的大力支持和协助，他帮我编写了很多示例demo和分析，而且给予了很多有建设性的建议。

## 前言

2020年9月10日，华为公司在2020年华为开发者大会上发布了HarmonyOS（鸿蒙操作系统）2.0版本。鸿蒙操作系统是一款面向全场景的分布式操作系统。鸿蒙操作系统不同于既有的Android、iOS、Windows、Linux等操作系统，它面向的是1+8+N的全场景设备，能够根据不同内存级别的设备进行弹性组装和适配，并且跨设备交互信息。

如果开发人员想要开发基于鸿蒙的App，目前可用的平台有3个：TV、Lite Wearable、Wearable。

如果我们开发的是TV或Wearable上的App，那么目前华为还没有开放基于X86的本机模拟器，因此需要将编写的代码发送到远程的ARM处理器以运行代码。在本机上只能预览运行结果，而无法运行和调试代码。

如果我们开发的是Lite Wearable上的App，那么既可以使用本机的预览器Previewer来预览代码的运行效果，也可以使用本机的模拟器Simulator来运行和调试代码，这给开发人员带来了相当出色的体验！此外，Lite Wearable对应的华为智能手表Watch GT2 Pro已经上市了。在Lite Wearable这个平台上，相关的设备和开发工具是最成熟、最完善的，因此，本书详细讲解的项目是在Lite Wearable上运行的。本书会跟随华为鸿蒙产品和开发工具包的发布节奏，在后续的版本中不断更新和扩充相应的实战项目。

本书详细、完整地介绍了一个呼吸训练App的开发全过程。本书采用项目导向和任务导向的写作方式讲解，总共分为36个任务，每个任务都分成3部分，包括运行效果、实现思路、代码详解。本书对编写的每一行代码进行讲解，可以说做到了手把手教学和保姆级教学。当读

者看完本书最后一页的时候，也就跟随作者成功做出了一个完整的项目。

## 本书读者对象

本书面向想要学习鸿蒙App开发的零基础开发人员。本书会对编写的每一行代码进行讲解，即便读者没有JavaScript的编程经验，也能在本书作者的一步步指导下完成书中整个项目的编写，从而实现项目的所有功能并将项目运行起来。

本书翔实地再现和还原了零基础开发人员在编写一个项目时的每一个过程和步骤，确保没有经验的开发人员也能够学会、学懂。

## 本书的组织结构

本书分为3章，主要内容如下。

第1章，“鸿蒙操作系统简介”，总体介绍了鸿蒙操作系统的两个重要特性，包括1+8+N全场景、分布式。

第2章，“项目准备工作”，介绍了开发鸿蒙App项目的准备工作，包括搭建鸿蒙App开发的环境、讲解在鸿蒙上开发的Hello World项目。

第3章，“呼吸训练实战项目”，从零开始完整而详细地介绍了运行在华为智能手表上的一个名为呼吸训练的实战项目。整个项目采用任务导向的方式，每个任务完成项目的一部分功能，每个任务包括运行效果、实现思路、代码详解3部分。当读者学习完最后一个任务时，就能完成整个项目。


## 资源与支持

本书由异步社区出品，社区（<https://www.epubit.com/>）为您提供相关资源和后续服务。

## 配套资源

本书提供如下资源：

- 本书源代码。

要获得以上配套资源，请在异步社区本书页面中点击 ，跳转到下载界面，按提示进行操作即可。注意：为保证购书读者的权益，该操作会给出相关提示，要求输入提取码进行验证。

如果您是教师，希望获得教学配套资源，请在社区本书页面中直接联系本书的责任编辑。

## 提交勘误

作者和编辑尽最大努力来确保书中内容的准确性，但难免会存在疏漏。欢迎您将发现的问题反馈给我们，帮助我们提升图书的质量。

当您发现错误时，请登录异步社区，按书名搜索，进入本书页面，单击“提交勘误”，输入勘误信息，单击“提交”按钮即可。本书的作者和编辑会对您提交的勘误进行审核，确认并接受后，您将获赠异步社区的100积分。积分可用于在异步社区兑换优惠券、样书或奖品。

详细信息

写书评

提交勘误

页码:

页内位置 (行数):

勘误印次:

B I U ABC      

字数统计

提交

## 扫码关注本书

扫描下方二维码，您将会在异步社区微信服务号中看到本书信息及相关的服务提示。



## 与我们联系

我们的联系邮箱是[contact@epubit.com.cn](mailto:contact@epubit.com.cn)。

如果您对本书有任何疑问或建议，请您发邮件给我们，并请在邮件标题中注明本书书名，以便我们更高效地做出反馈。

如果您有兴趣出版图书、录制教学视频，或者参与图书翻译、技术审校等工作，可以发邮件给我们；有意出版图书的作者也可以到异步社区在线提交投稿（直接访问[www.epubit.com/selfpublish/submission](http://www.epubit.com/selfpublish/submission)即可）。

如果您所在的学校、培训机构或企业，想批量购买本书或异步社区出版的其他图书，也可以发邮件给我们。

如果您在网上发现有针对异步社区出品图书的各种形式的盗版行为，包括对图书全部或部分内容的非授权传播，请您将怀疑有侵权行为的链接发邮件给我们。您的这一举动是对作者权益的保护，也是我们持续为您提供有价值的内容的动力之源。

## 关于异步社区和异步图书

“异步社区”是人民邮电出版社旗下IT专业图书社区，致力于出版精品IT技术图书和相关学习产品，为作译者提供优质出版服务。异步社区创办于2015年8月，提供大量精品IT技术图书和电子书，以及高品质技术文章和视频课程。更多详情请访问异步社区官网

<https://www.epubit.com>。

“异步图书”是由异步社区编辑团队策划出版的精品IT专业图书的品牌，依托于人民邮电出版社近30年的计算机图书出版积累和专业编辑团队，相关图书在封面上印有异步图书的LOGO。异步图书的出版领域包括软件开发、大数据、AI、测试、前端、网络技术等。



异步社区



微信服务号



# 第1章 鸿蒙操作系统简介

2020年9月10日，华为公司在2020年华为开发者大会上发布了HarmonyOS（鸿蒙操作系统）2.0版本。鸿蒙操作系统是一款面向全场景的分布式操作系统。鸿蒙操作系统不同于既有的Android、iOS、Windows、Linux等操作系统，它面向的是1+8+N的全场景设备，能够根据不同内存级别的设备进行弹性组装和适配，并且跨设备交互信息。

## 1.1 1+8+N全场景

就目前而言，基于硬件的生态是相互割裂的，无论是手机、手表、电视还是车机，都有各自独立的生态。这些割裂的生态影响了用户体验。用户期望能够打破单个设备的孤岛，获得多设备之间的无缝连接体验。未来几年，随着人均持有的终端设备数量越来越多，全场景体验将是赢取未来的关键点，为此，鸿蒙操作系统面向的是1+8+N的全场景体验，如图1-1所示。

1+8+N中的“1”指的是处于中间位置的手机，它是用户流量的核心入口。1+8+N中的“8”指的是手机外围的8类设备，包括PC、平板电脑、耳机、眼镜、手表、车机、音响、HD设备，这8类设备在人们日常生活中的使用率仅次于手机。1+8+N中的“N”指的是最外层的所有能够搭载鸿蒙操作系统的IoT（Internet of Things，物联网）设备，这些设备涵盖了各种各样的应用场景，包括运动健康、影音娱乐、智能家庭、移动办公、智慧出行等。针对运动健康这个场景，常见的设备有血压计、智能秤；针对移动办公这个场景，常见的设备有打印机、投

影仪；针对智能家庭这个场景，常见的设备有扫地机、摄像头。之所以称之为“N”，就是因为它涵盖的应用场景非常广泛，可以说是无穷无尽的，这给我们提供了尽情发挥想象力和创造力的空间。

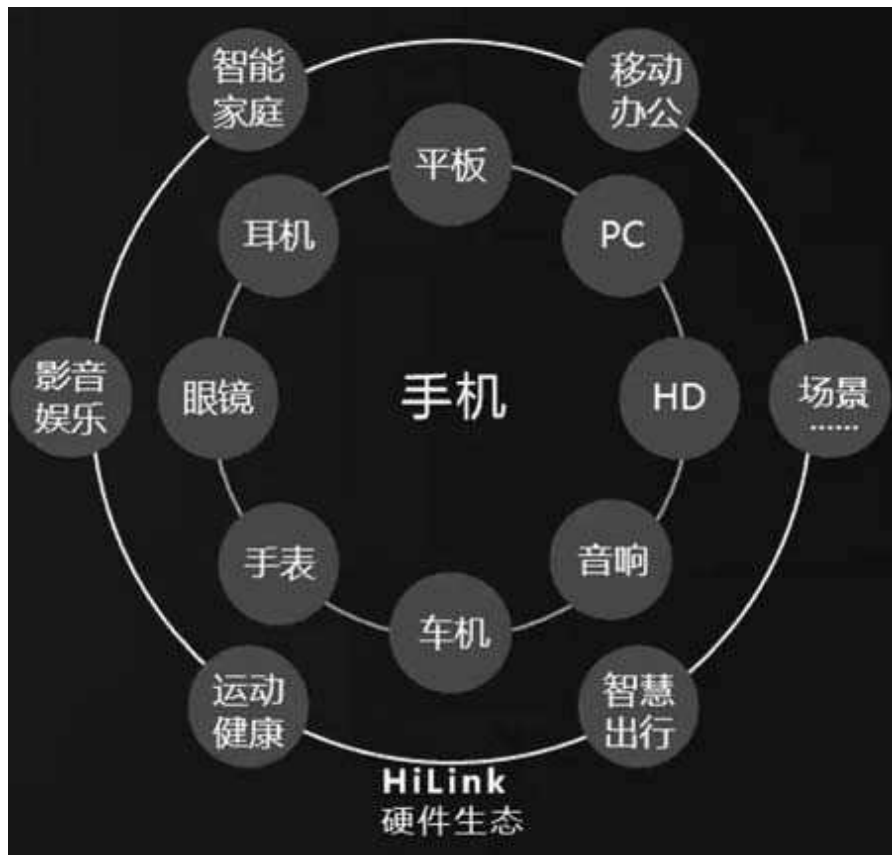


图1-1 1+8+N全场景

## 1.2 分布式

传统的设备是由设备内部的硬总线连在一起的，硬总线是设备内部的部件之间进行通信的基础。如果想让多个设备之间分布式地通信和共享数据，并让多个设备融合为一体，仅仅通过硬总线是很难实现的。

分布式软总线是实现分布式能力的基础，是多种终端设备的统一基座，它为设备之间的互联互通提供了统一的分布式通信能力，能够快速发现并连接设备，以及高效地分发任务和传输数据。

分布式软总线融合了近场和远场的通信技术，并且可以充分发挥近场通信的技术优势。分布式软总线承担了任务总线、数据总线和总线中枢三大功能。其中，任务总线负责将应用程序在多个终端上快速分发；数据总线负责数据在设备间的高性能分发和同步；总线中枢起到协调控制的作用，用于自动发现并组网，以及维护设备间的拓扑关系。

目前，分布式软总线在性能上已经无限逼近硬总线的能力，可以让多个设备融合为一体，如图1-2所示。鸿蒙操作系统的分布式软总线已经可以实现异构融合网络，比如使用蓝牙通信的设备和使用WiFi通信的设备可以互见互联，一次配网之后可以自发现、自连接。分布式软总线也可以实现动态时延校准，比如手机将视频分享给智慧屏，并且将音频分享给音箱，分享之后音视频依然是同步的。

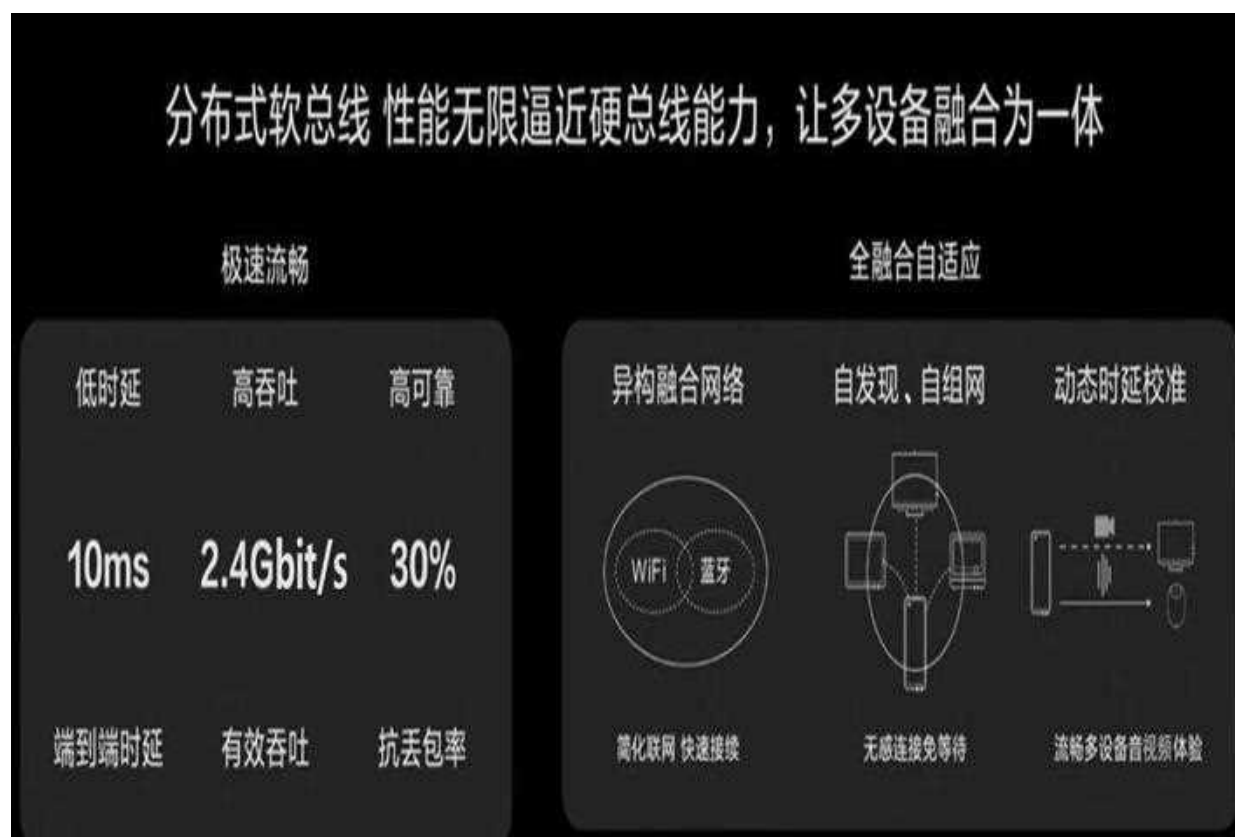


图1-2 分布式软总线的性能

此外，分布式的数据管理让跨设备数据处理如同本地处理一样方便快捷，如图1-3所示。在鸿蒙操作系统的分布式数据管理能力下，在华为5G通信技术的增益下，硬件设备之间的界限将变得越来越模糊——一个设备可能会成为另外一个设备的子部件，或者多个设备成为一个整体设备，从而实现数据共享、算力共享、AI共享。



图1-3 分布式数据管理的方便快捷性

分布式软总线 and 分布式数据管理等核心技术，使得搭载鸿蒙操作系统的设备之间方便快捷地进行分布式通信成为可能，从而让鸿蒙应用可以跨设备接续业务。下面列举几个典型的应用场景。

第1个应用场景是多屏联动课堂，如图1-4所示。学生参加远程课堂时，使用一个平板电脑与远程的老师进行交互，老师的画面显示在电视机大屏上。当学生需要回答老师的问题时，就像在线下的课堂回答问题一样，学生举手示意之后，老师把学生的投影显示在大屏上。学生在作答的过程中，大屏上会同步展示学生的作答过程，就像是学生在线下教室答题时全班的同学都能看到一样。当学生需要在线做主观题时，学生用笔在平板电脑上回答主观题，可以把回答主观题的整个过程同步展示到大屏上，这样老师就可以对学生做的主观题及时地做出评价并反馈给学生了。



图1-4 多屏联动课堂

第2个应用场景是大屏多人体感游戏，如图1-5所示。目前在大屏上运行的很多体感游戏都需要使用游戏手柄。如何让这些体感游戏不再需要游戏手柄呢？可以让用户在玩游戏之前使用手机扫描一个二维码，这样，手机就会秒变游戏手柄。多个用户扫描同一个游戏的二维码之后，就可以通过手机上的体感传感器一起欢快地玩游戏了。

第3个应用场景是协同会议空间，如图1-6所示。在日常的工作中，当我们在会议室里讨论一个问题时，如果主讲人想把讨论的内容投影到大屏上，只需要使用手机扫描一个二维码就可以了。对于分享投屏内容的这部手机，它的所有其他内容都不会被投影到大屏上，这跟目前所有的投屏方式都是不一样的。如果想邀请其他人加入投屏内容的讨论，只需要分享一个二维码就可以让别人加入进来。这样的能力相

当于让我们拥有了一块分布式的白板，从而让我们的讨论变得更加高效。



图1-5 大屏多人感游戏

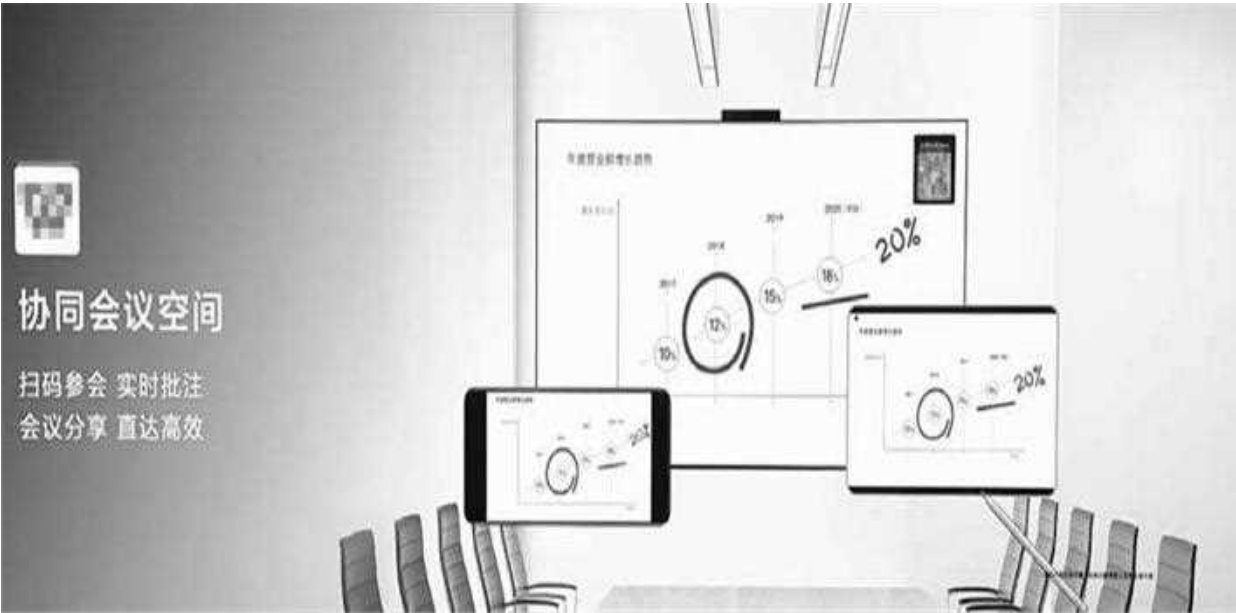


图1-6 协同会议空间

第4个应用场景是协同导航，如图1-7所示。可以把手机的导航信息同步到手表上，这样就可以解放出双手，在户外运动时这尤其方便。

第5个应用场景是协同打车，如图1-8所示。当我们用手机打车之后，在等车时手上可能会拿着很多东西，如果不时地用手机查看车的当前位置信息，就会非常不方便。这时，就可以将手机上的所有打车信息都同步到手表上，从而将手解放出来去拿东西，而通过手表查看车的当前位置信息。当我们上车之后，手机上的所有目的地信息和支付信息也都会同步到手表上。



图1-7 协同导航





图1-8 协同打车

前面列举的几个典型的应用场景，基于全都搭载了鸿蒙操作系统的多个设备，可以非常轻松地实现。

### 1.3 小结

鸿蒙操作系统得益于全场景、分布式的特性，具有适用范围广、系统适配灵活、跨设备通信能力强的特点。

希望通过鸿蒙操作系统能够打造出中国软件的“根”！我们每一个人都可以为鸿蒙操作系统尽一点儿绵薄之力，哪怕只是沧海一粟！

未来3~5年，中国平均每个家庭的终端设备会增长到十几台，未来的市场会以万亿计。鸿蒙操作系统在未来2~3年是生态的成长期，这期间在各个领域都会出现巨大的机会。通过学习鸿蒙操作系统相关的开发（无论是软件开发还是硬件开发），我们都可以更多地参与这场千载难逢的生态变革。

从下一章开始，我们将从零开始完成一个鸿蒙App的实战项目，力争做到手把手教学和保姆级教学。当读者看完本书最后一页的时候，也就成功地做出了一个完整的项目。

## 第2章 项目准备工作

我们会在本章为后文的呼吸训练实战项目做一些准备工作，包括搭建鸿蒙App开发的环境、讲解在鸿蒙上开发的Hello World项目。

### 2.1 搭建开发环境

搭建鸿蒙App开发的环境包括两步：安装Node.js；安装及配置集成开发环境DevEco Studio。接下来详细讲解这两个操作步骤。

#### 2.1.1 安装Node.js

在浏览器中输入Node.js的官网下载链接：[nodejs.org/zh-cn/download/](https://nodejs.org/zh-cn/download/)，然后下载长期支持版的64位Windows安装包，如图2-1所示。

下载的Windows安装包以msi作为扩展名，如图2-2所示。

双击安装包即可开始安装，如图2-3所示。单击Next按钮进入下一步。

在新打开的窗口中，选中复选框以表示同意许可协议中的条款，如图2-4所示。单击Next按钮进入下一步。



图2-1 Node.js的官网下载页面



图2-2 扩展名是msi的Windows安装包



图2-3 开始安装Node.js

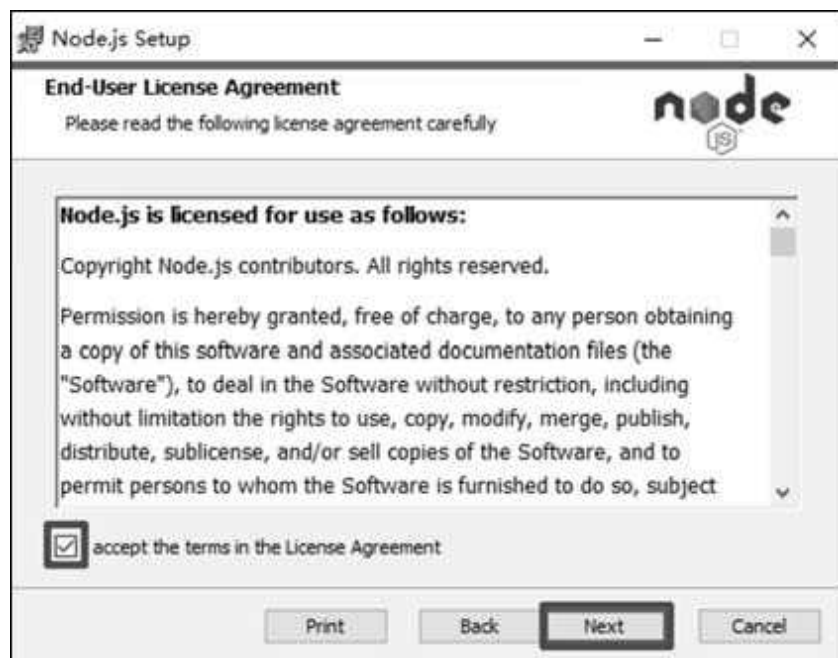


图2-4 终端用户许可协议

在新打开的窗口中，可以自定义Node.js的安装路径，如图2-5所示。这里直接使用默认的安装路径就可以了。单击Next按钮进入下一步。

在新打开的窗口中，可以自定义功能的安装方式，如图2-6所示。这里直接使用默认的安装方式就可以了。单击Next按钮进入下一步。

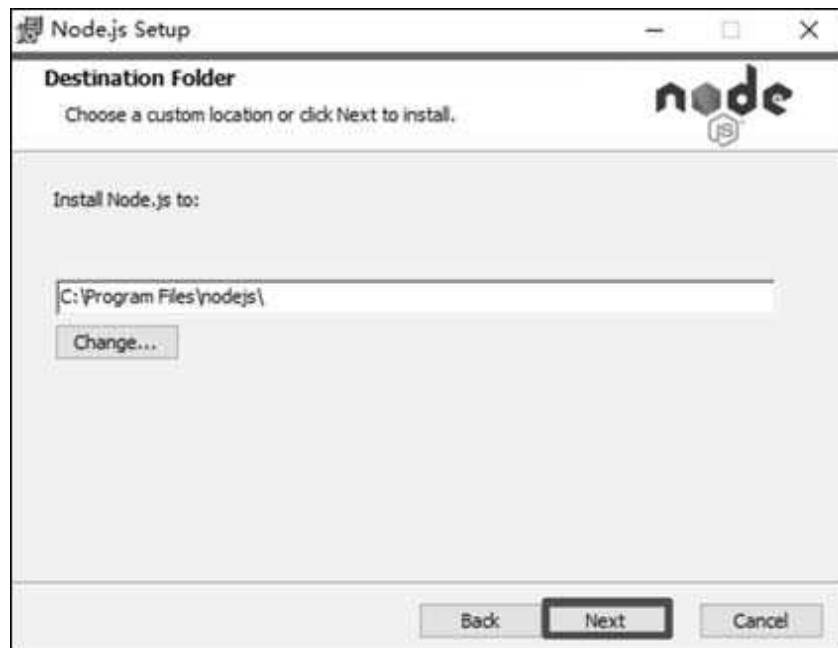


图2-5 自定义Node.js的安装路径

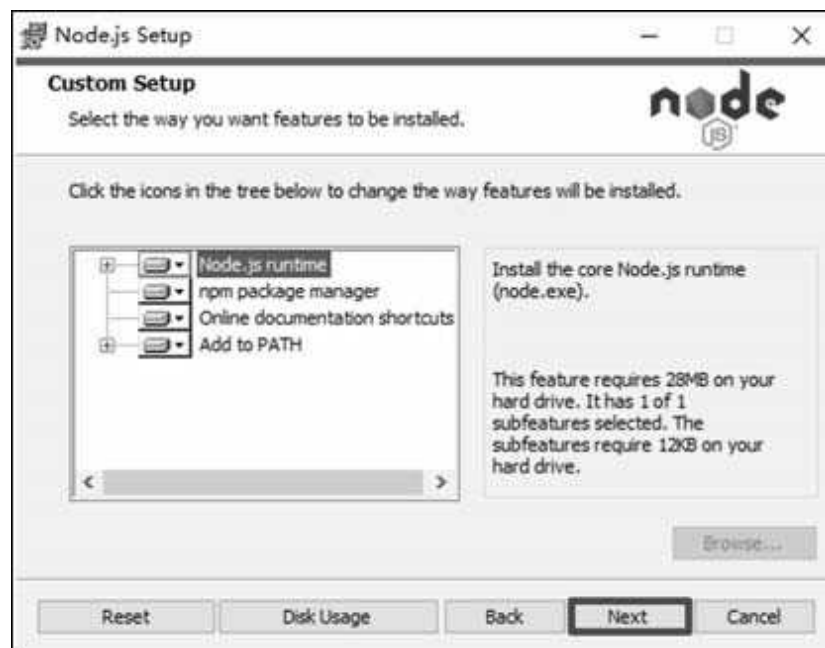


图2-6 自定义功能的安装方式

在新打开的窗口中，可以选择是否安装那些用来编译Native模块的必要工具。因为我们不需要编译Native模块，所以无须选中窗口中的复选框，如图2-7所示。单击Next按钮进入下一步。

在新打开的窗口中，准备安装Node.js，如图2-8所示。单击Install按钮开始安装Node.js。

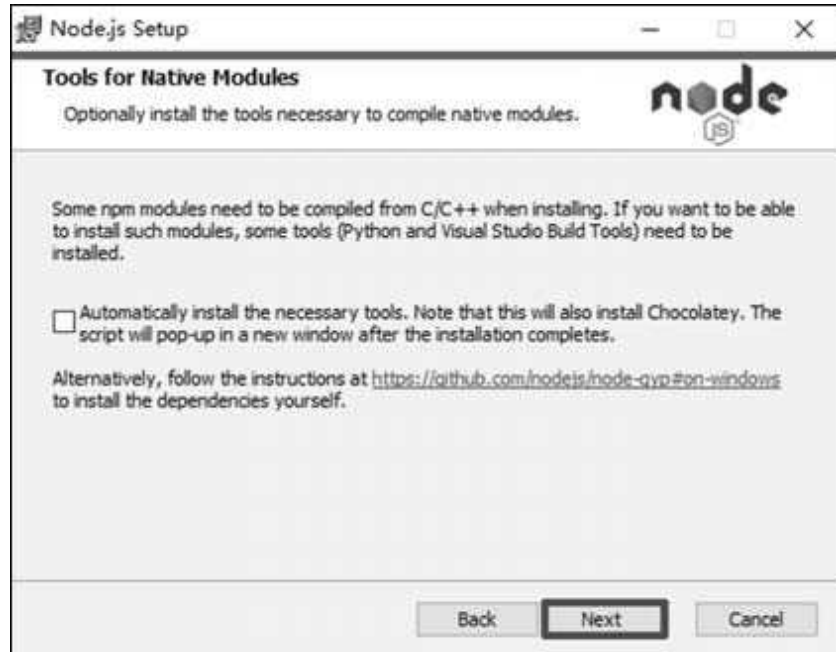


图2-7 自定义安装编译Native模块的必要工具



图2-8 准备安装Node.js

在新打开的窗口中，正在安装Node.js，如图2-9所示。

安装完成之后，会在新打开的窗口中显示Node.js已经被成功地安装了，如图2-10所示。单击Finish按钮以关闭安装窗口。

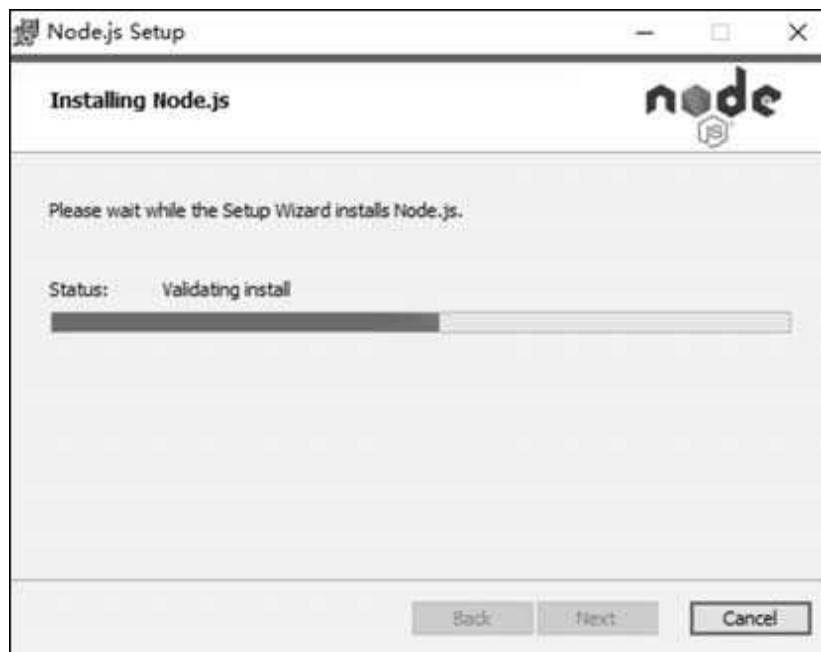


图2-9 正在安装Node.js





图2-10 Node.js已经被成功安装

## 2.1.2 安装及配置DevEco Studio

开发鸿蒙App所使用的集成开发环境是DevEco Studio。在浏览器中输入DevEco Studio的官网下载链接：[developer.harmonyos.com/cn/home](https://developer.harmonyos.com/cn/home)，然后单击页面中的下载图标，如图2-11所示。



图2-11 DevEco Studio的官网页面

在新打开的页面中，显示DevEco Studio的当前最新版本是2.0 Beta1，而且只有64位的Windows版本可供下载，如图2-12所示。单击页面中的下载图标以下载Windows安装包。



图2-12 DevEco Studio的官网下载页面

如果还没有登录华为账号，会打开一个华为账号登录的页面，如图2-13所示。



图2-13 华为账号登录页面

登录华为账号之后，就可以单击下载图标进行下载了。将下载之后的zip压缩包解压之后，就得到了扩展名为exe的Windows安装包，如图2-14所示。

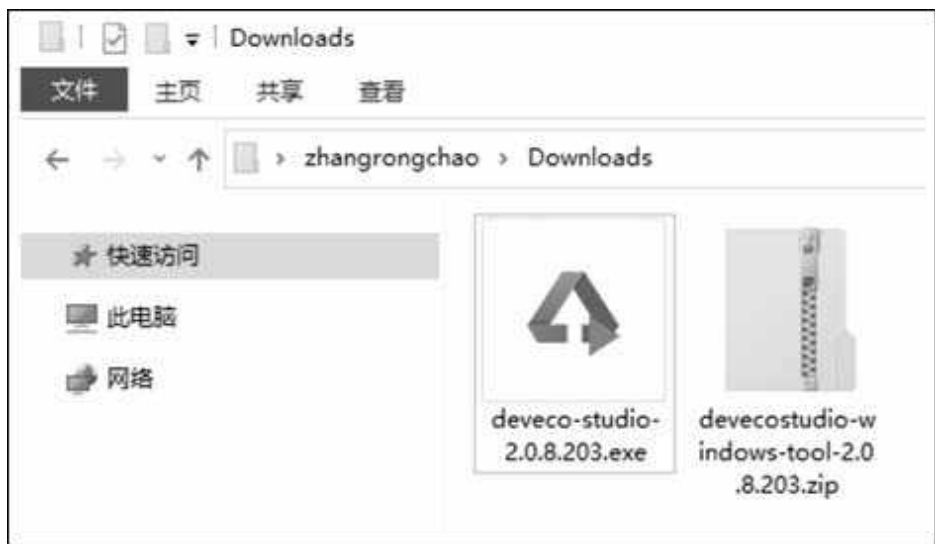


图2-14 下载之后的zip压缩包和解压之后的安装包

双击安装包即可开始安装，如图2-15所示。单击Next按钮进入下一步。

在新打开的窗口中，可以自定义DevEco Studio的安装路径，如图2-16所示。这里直接使用默认的安装路径就可以了。单击Next按钮进入下一步。



图2-15 开始安装DevEco Studio

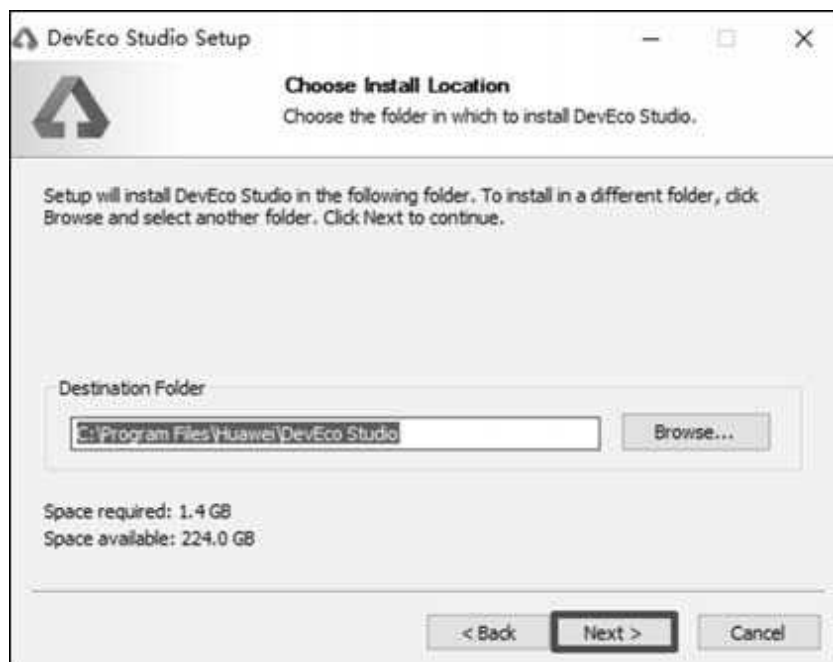


图2-16 自定义DevEco Studio的安装路径

在新打开的窗口中，可以配置DevEco Studio的安装选项，这里在Create Desktop Shortcut区域选中DevEco Studio launcher复选框，如图2-17所示。单击Next按钮进入下一步。

在新打开的窗口中，为DevEco Studio的快捷方式选择一个开始菜单的文件夹，这里使用默认的名称Huawei就可以了，如图2-18所示。单击Install按钮进入下一步。

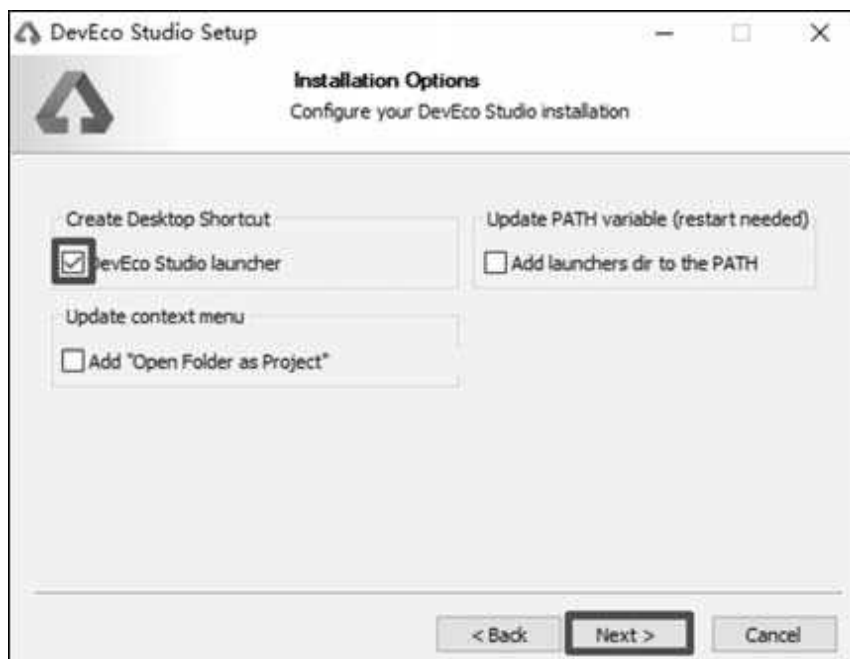


图2-17 配置DevEco Studio的安装选项

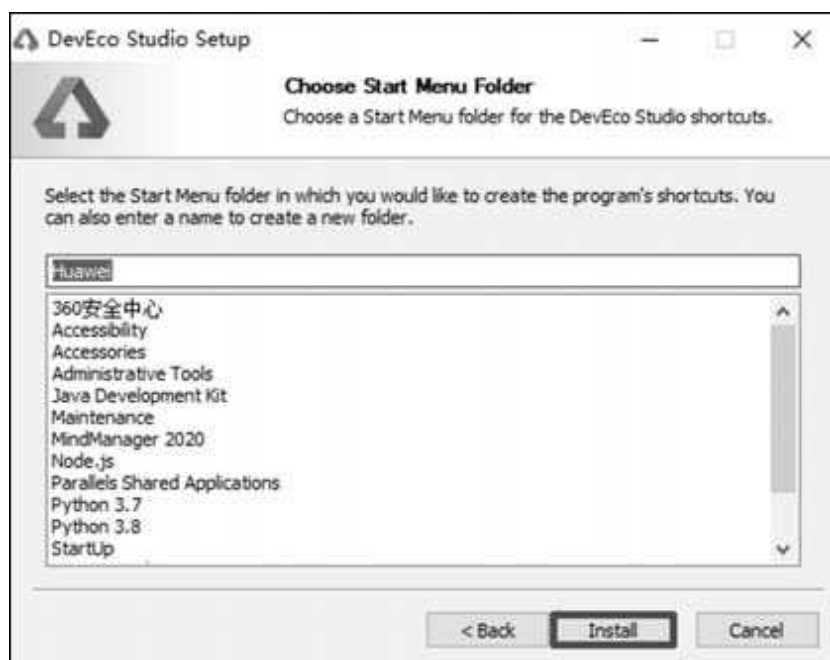


图2-18 选择开始菜单的文件夹

在新打开的窗口中，正在安装DevEco Studio，如图2-19所示。

安装完成之后，在新打开的窗口中显示DevEco Studio已经被安装在了你的电脑上，如图2-20所示。选中Run DevEco Studio复选框，然后

单击Finish按钮进入下一步。

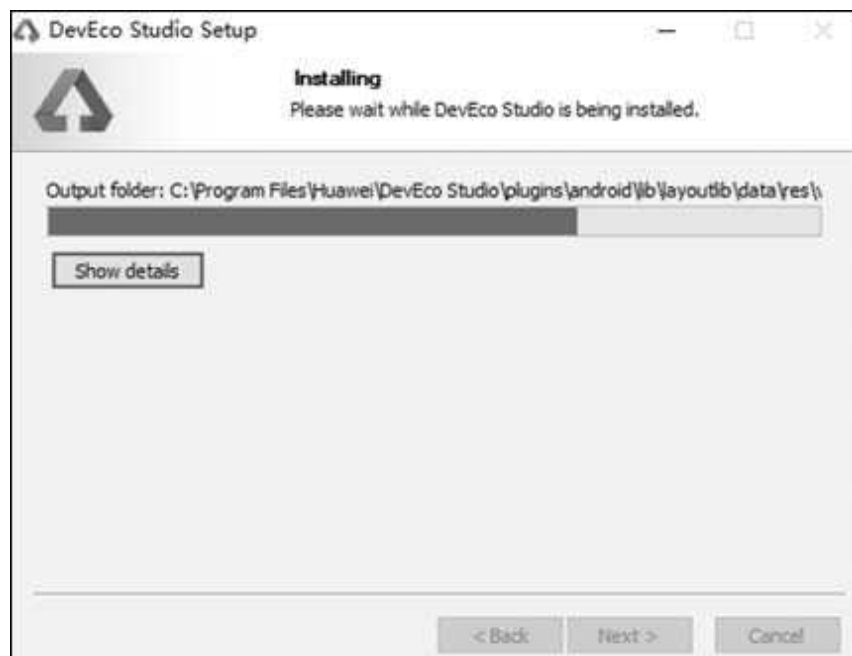


图2-19 正在安装DevEco Studio



图2-20 完成安装DevEco Studio



在新打开的窗口中，可以选择是否导入DevEco Studio的设置。这里选择不导入设置，如图2-21所示。单击OK按钮进入下一步。

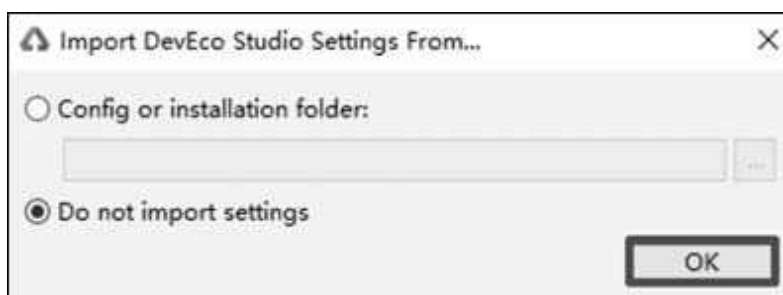


图2-21 是否导入DevEco Studio的设置

在新打开的窗口中，需要确认已经阅读并且接受了用户许可协议中的条款和条件，如图2-22所示。选中confirm that I have read and accept the terms and conditions复选框，然后单击Agree按钮进入下一步。

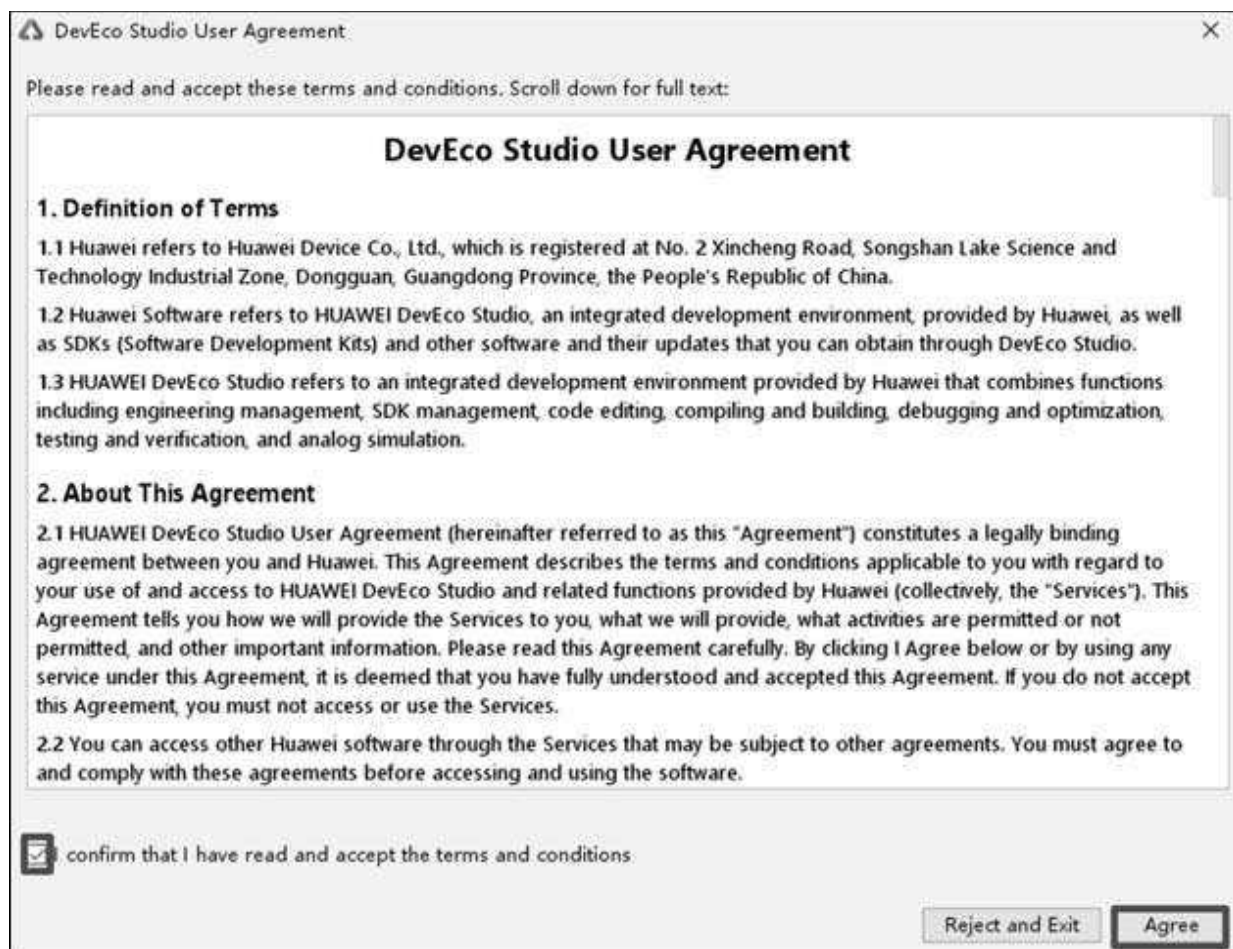


图2-22 DevEco Studio用户许可协议

在新打开的窗口中，下载相关的SDK组件，如图2-23所示。单击Next按钮进入下一步。

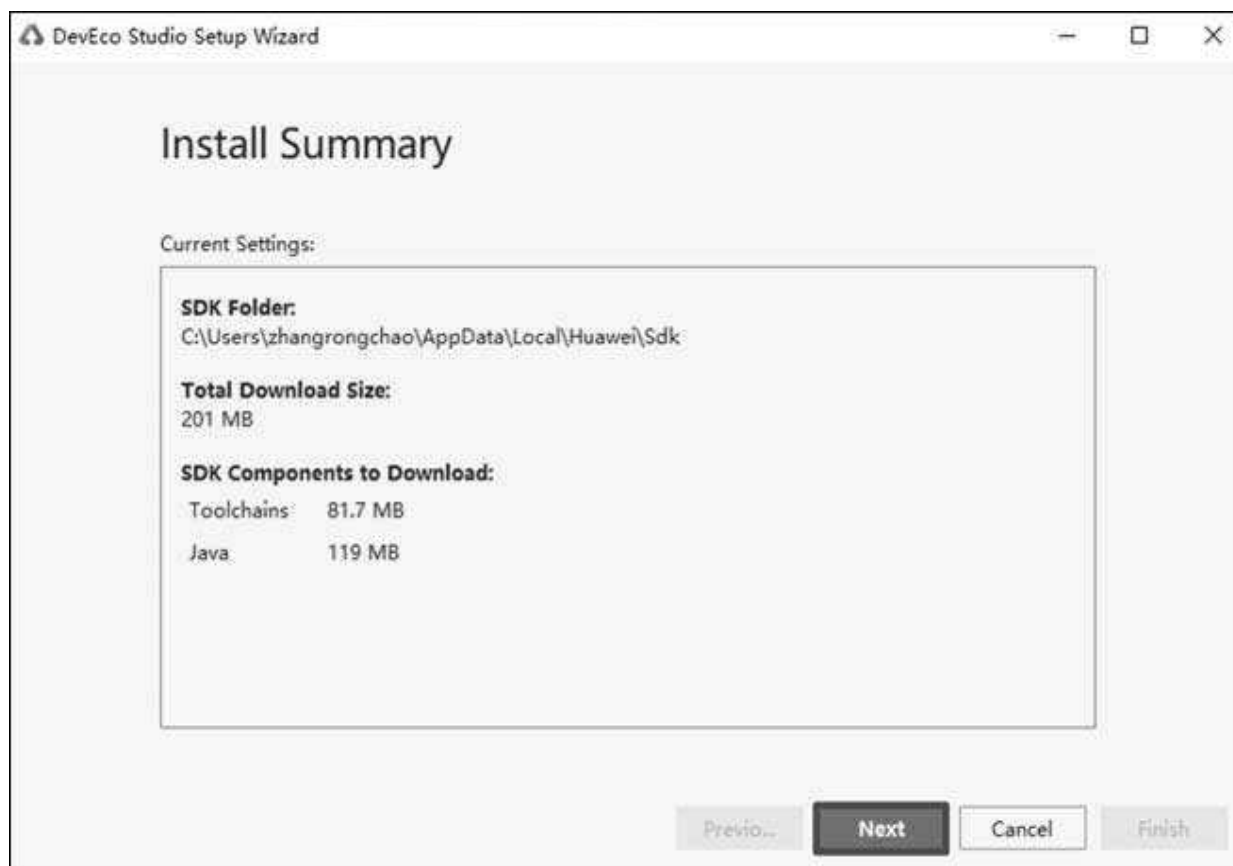


图2-23 下载相关的SDK组件

在新打开的窗口中，需要接受SDK组件的协议许可，如图2-24所示。选中Accept单选按钮，然后单击Next按钮进入到下一步。



图2-24 SDK组件的协议许可

在新打开的窗口中，正在下载SDK组件，如图2-25所示。

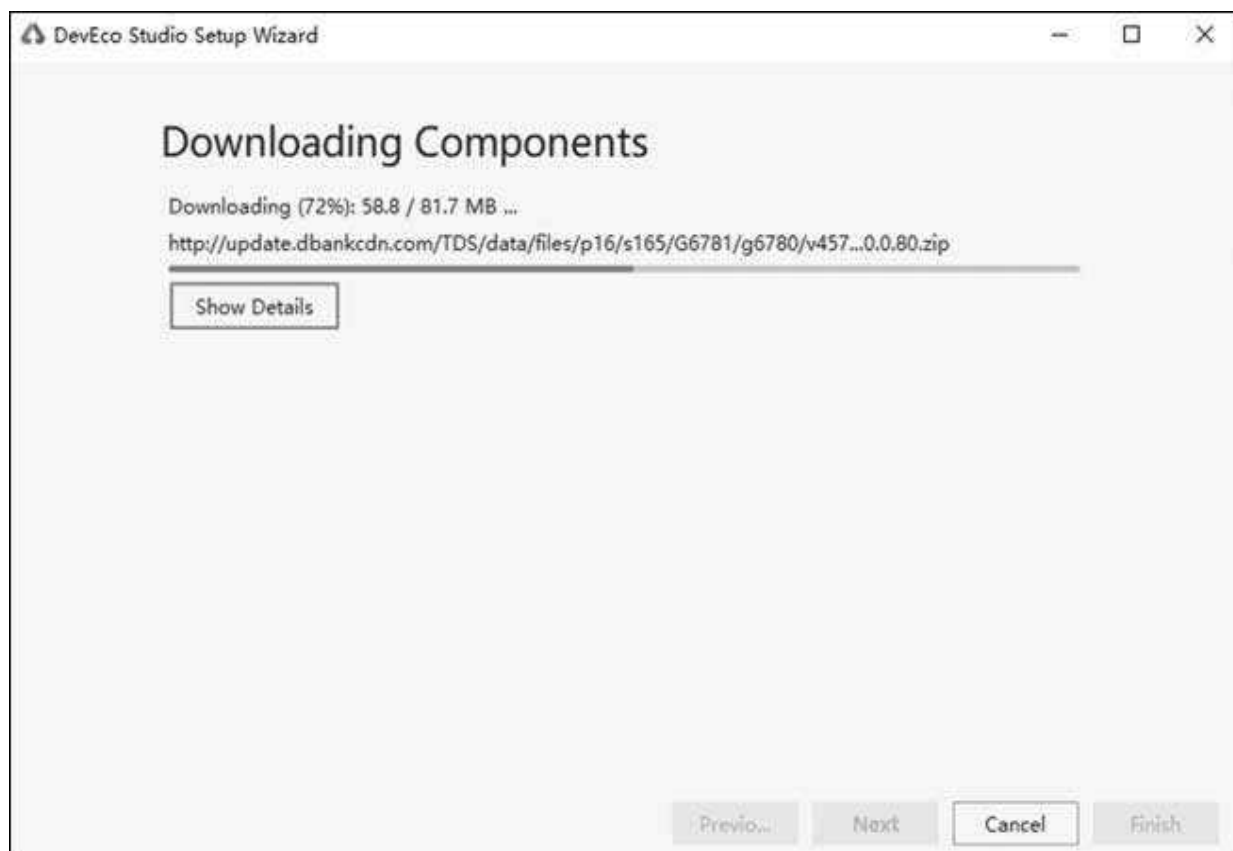


图2-25 正在下载SDK组件

下载完成之后，打开一个新窗口，如图2-26所示。单击Finish按钮进入到下一步。

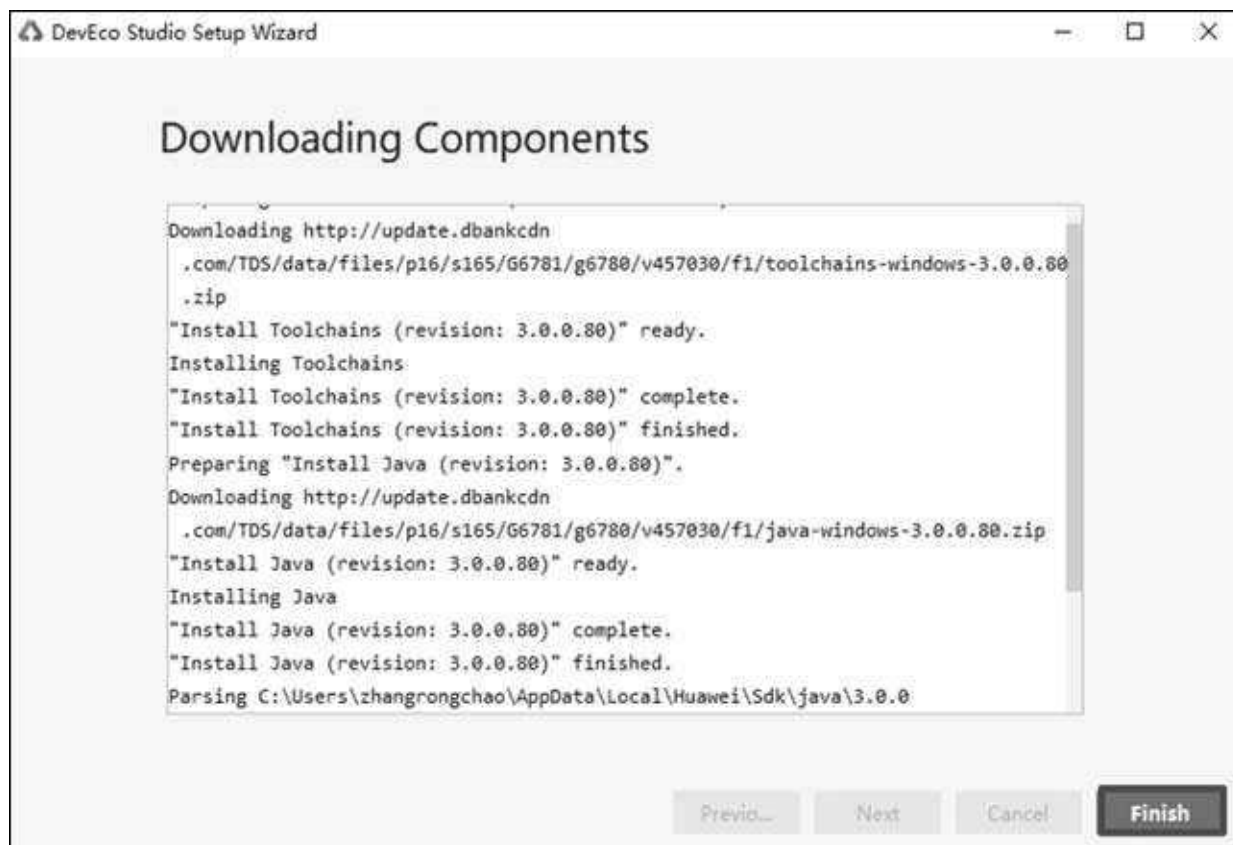


图2-26 下载完成SDK组件

在新打开的窗口中单击Configure按钮，然后在展开的下拉菜单中单击Settings，如图2-27所示。

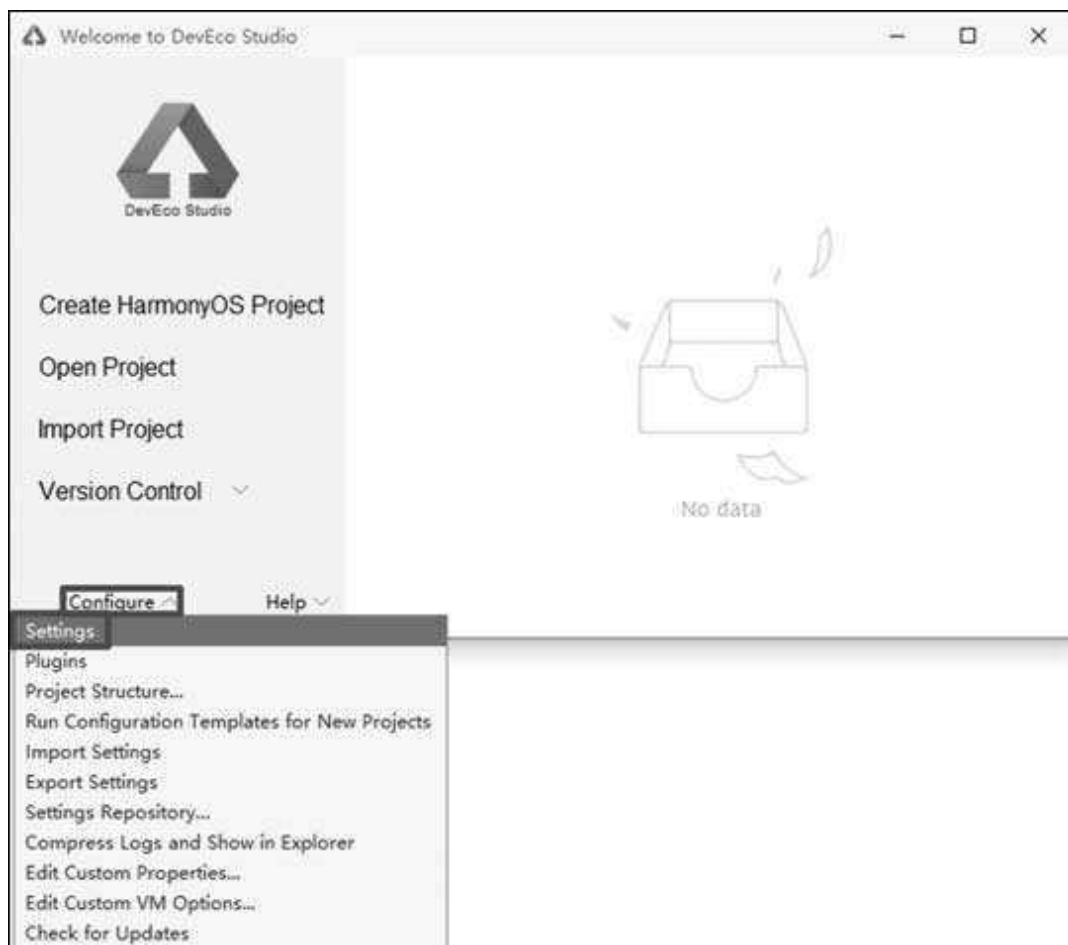


图2-27 配置DevEco Studio

在新打开的窗口中，单击窗口左侧的HarmonyOS SDK，然后单击窗口右侧的SDK Platforms选项卡，并选中Js复选框，如图2-28所示。

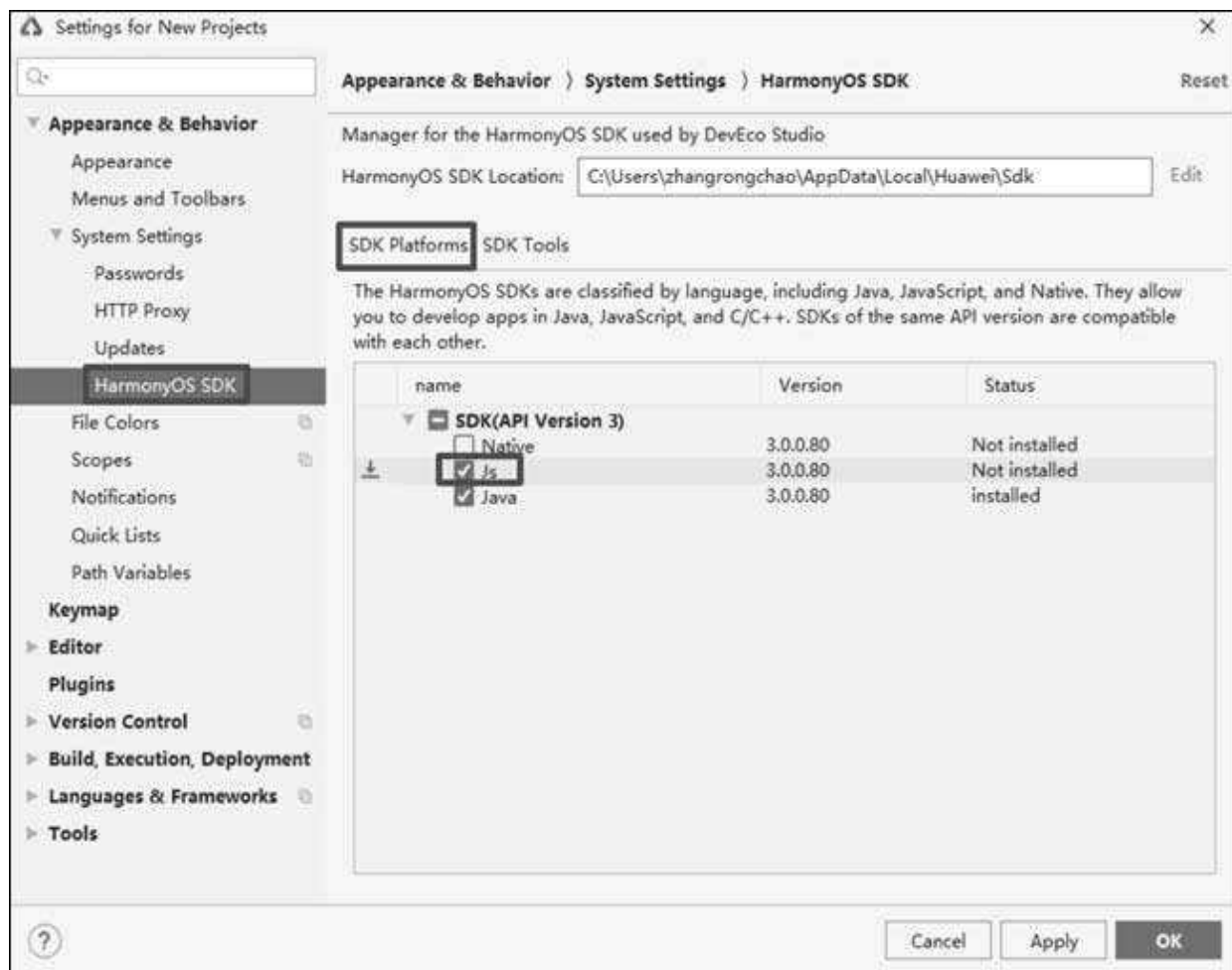


图2-28 配置安装SDK中的Js组件

单击窗口右侧的SDK Tools选项卡，并选中Previewer复选框，如图2-29所示。单击Apply按钮进入到下一步。



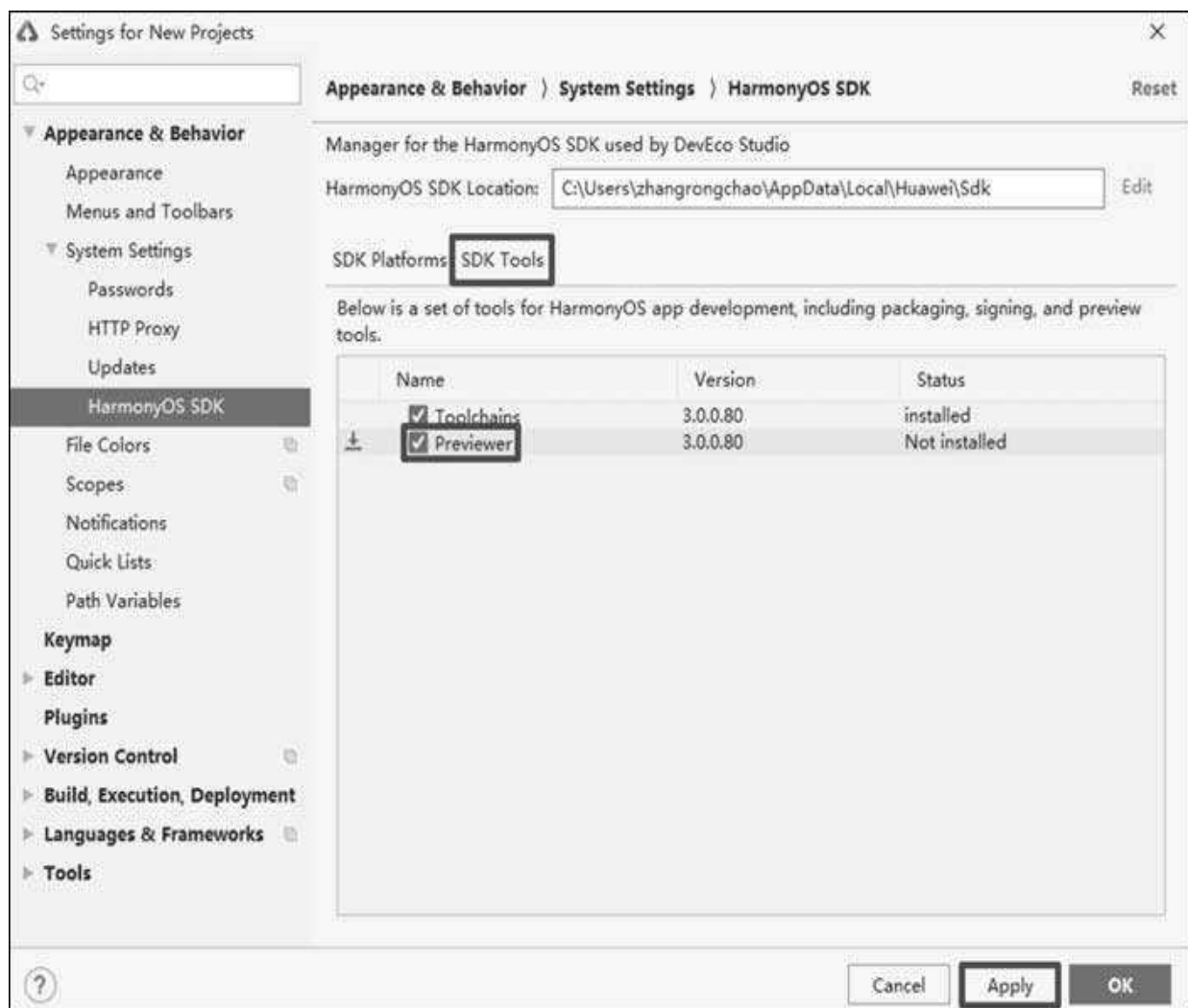


图2-29 配置安装SDK Tools中的Previewer组件

在新打开的窗口中，确认要安装的组件，如图2-30所示。单击OK按钮进入到下一步。

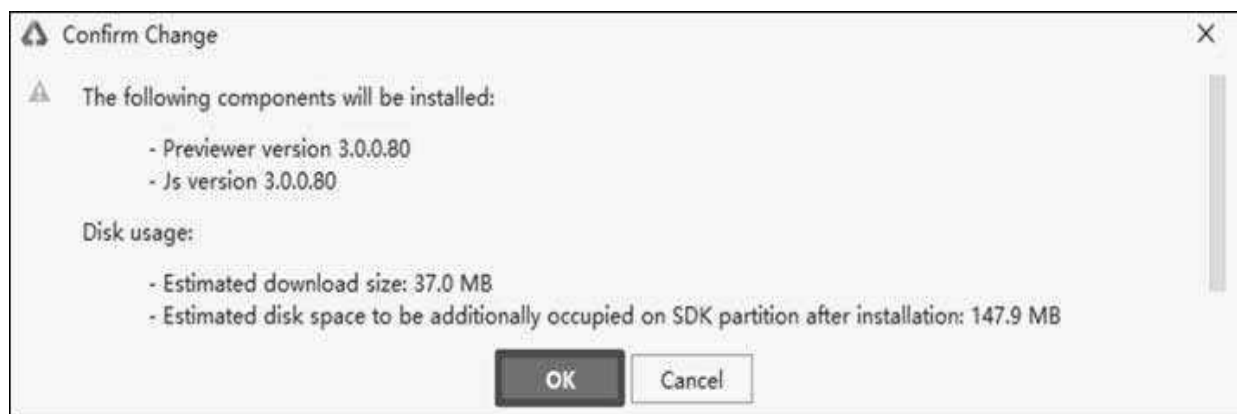


图2-30 确认要安装的组件

在新打开的窗口中，正在安装相关的组件，如图2-31所示。

安装完相关组件之后，打开一个新窗口，如图2-32所示。单击 Finish按钮进入到下一步。



图2-31 正在安装相关的组件



图2-32 相关组件安装完毕

在之前打开的窗口中，单击OK按钮，完成DevEco Studio的配置，如图2-33所示。

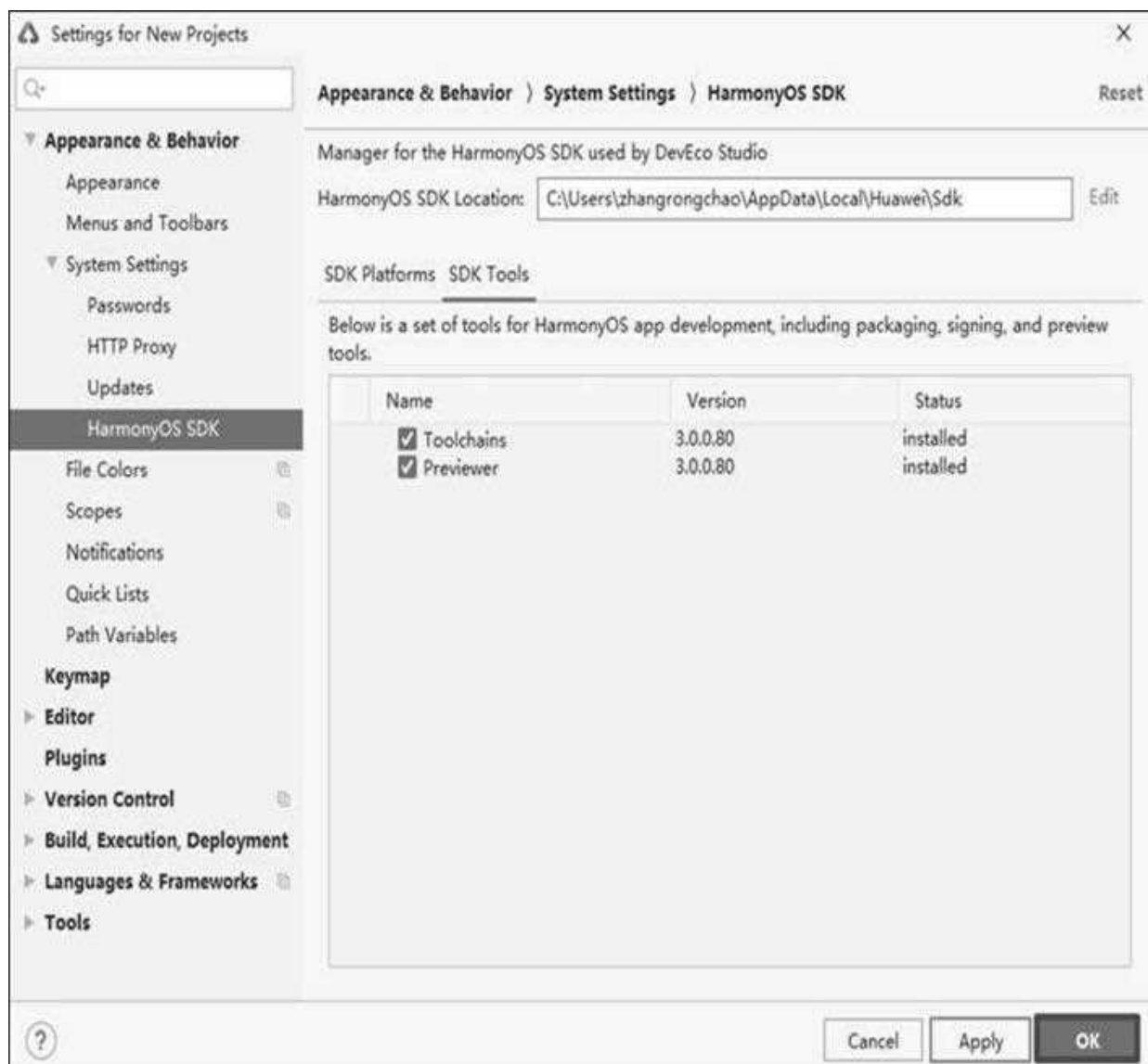


图2-33 完成DevEco Studio的配置

## 2.2 Hello World

搭建好开发环境之后，我们就可以新建一个Hello World项目了。

打开集成开发环境DevEco Studio，单击Create HarmonyOS Project，以创建一个鸿蒙项目，如图2-34所示。

在新打开文件的窗口中，首先选择App所运行的Device和使用的Template。默认选中的Device是TV，可用的Template有6个，如图2-35所示。

其中，有3个Template的名字是以“(JS)”结尾的，有3个Template的名字是以“(Java)”结尾的，这说明开发TV上的App既可以使用编程语言JavaScript，也可以使用编程语言Java。



图2-34 创建一个鸿蒙项目

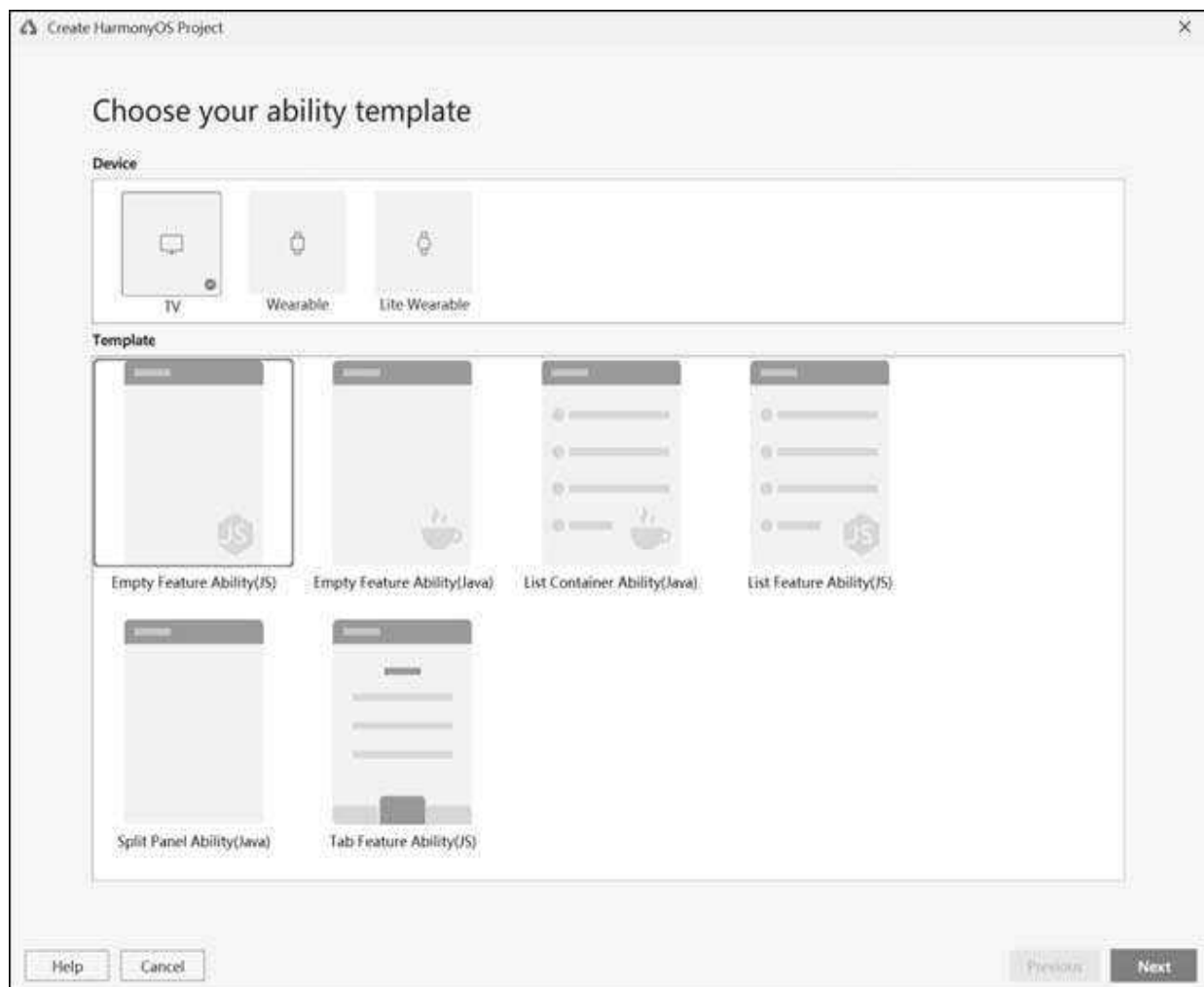


图2-35 TV可用的Template

在Device一栏中选中Wearble，可用的Template有3个，如图2-36所示。

其中，有两个Template的名字是以“(JS)”结尾的，有一个Template的名字是以“(Java)”结尾的，这说明开发Wearable上的App与开发TV上的App一样，既可以使用编程语言JavaScript，也可以使用编程语言Java。

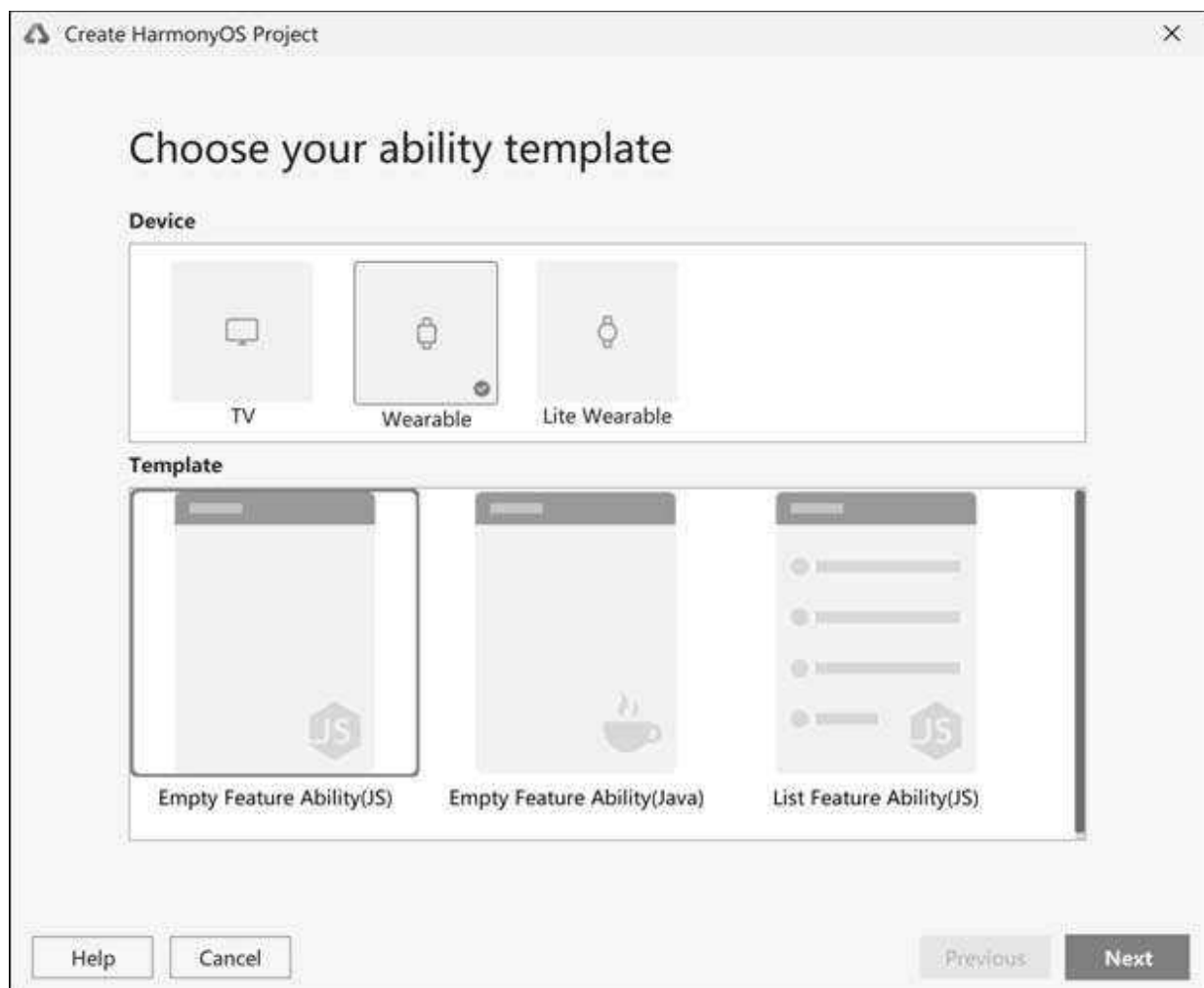


图2-36 Wearable可用的Template

在Device一栏中选中Lite Wearble，可用的Template有两个，如图2-37所示。



图2-37 Lite Wearable可用的Template

不论选择哪一个Template，都是使用编程语言JavaScript创建一个模板，原因是开发Lite Wearable上的App时只能使用编程语言JavaScript，而不能使用编程语言Java。

对于上述3种Device，TV对应的华为产品是智慧屏，Wearable对应的华为产品是智能手表，Lite Wearable对应的华为产品是智能手表。

如果我们开发的是TV或Wearable上的App，由于目前（截至2020年12月1日）华为还没有开放基于X86的本机模拟器，因此需要将编写的代码发送到远程的ARM处理器以运行代码。在本机上只能预览运行结果，而无法运行和调试代码。



如果我们开发的是Lite Wearable上的App，那么既可以使用本机的预览器Previewer来预览代码的运行效果，也可以使用本机的模拟器Simulator来运行和调试代码，这给开发人员带来了相当出色的体验！此外，当读者看到本书的时候，华为的智能手表没准已经上市了。在Lite Wearable平台上，相关的设备和开发工具是最成熟、最完善的，因此，本书详细讲解的项目是运行在智能手表上的。本书会跟随华为鸿蒙产品和开发工具包的发布节奏，在后续的版本中不断更新和扩充相应的实战项目。

在图2-37中，默认选中的Template是Empty Feature Ability，单击Next按钮。在新打开的窗口中配置新建的项目，需要分别配置项目名称、包名、项目的保存位置和可兼容的SDK。将Project Name取名为BreathTraining，DevEco Studio会自动帮我们生成一个Package name，其名称为com.example.breathtraining。Save location和Compatible SDK都使用默认的配置，如图2-38所示。

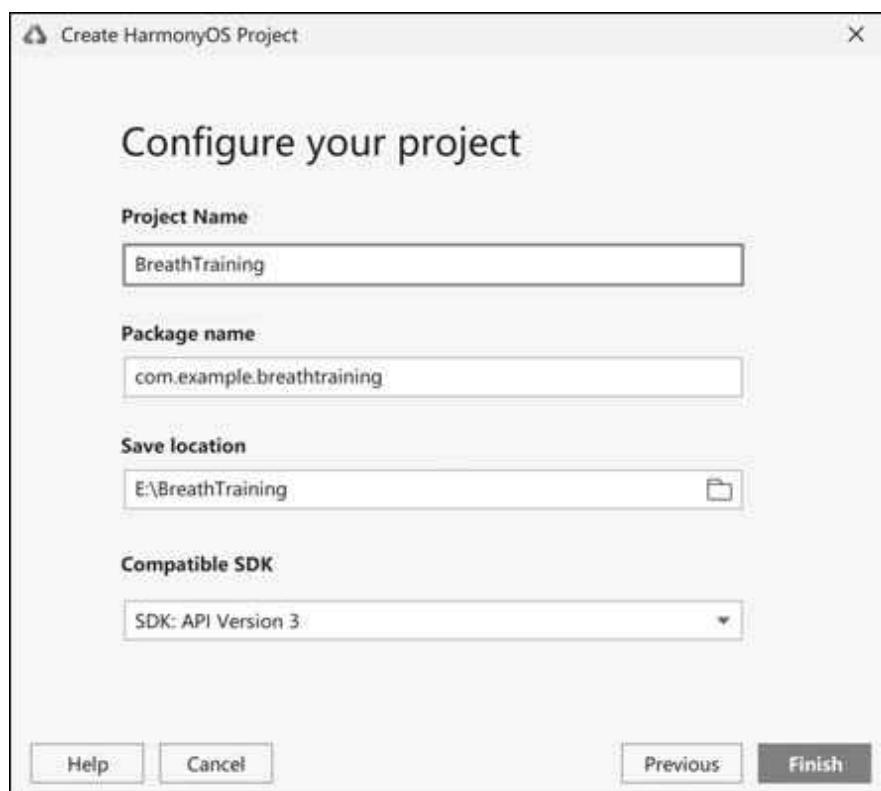


图2-38 配置新建的项目

单击Finish按钮之后，就创建了一个轻量级可穿戴设备的Hello World项目，也就是创建了一个运行在智能手表上的Hello World项目，如图2-39所示。

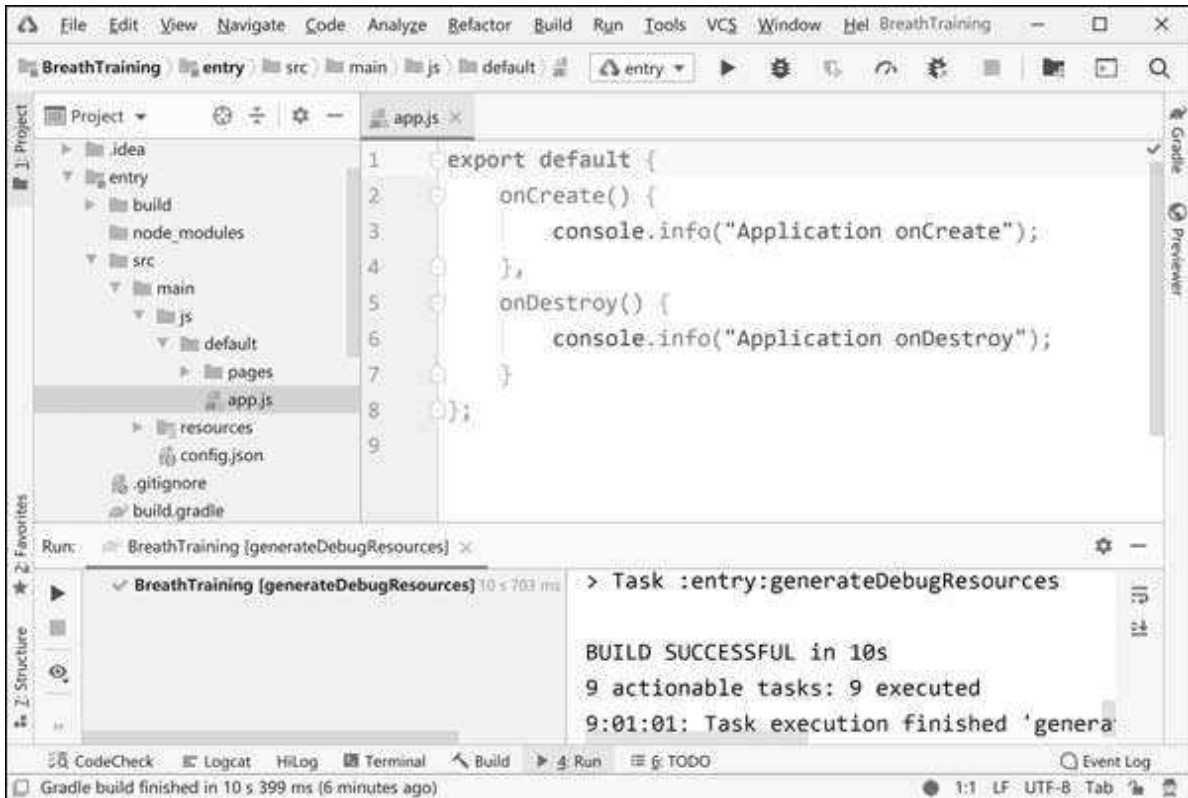


图2-39 新建的Hello World项目

选中菜单栏中的View，在展开的菜单中选中Tool Windows，在展开的子菜单中单击Previewer命令，如图2-40所示。

通过Previewer就可以预览App的运行效果了。在智能手表的主页面显示了文本“Hello World”，如图2-41所示。

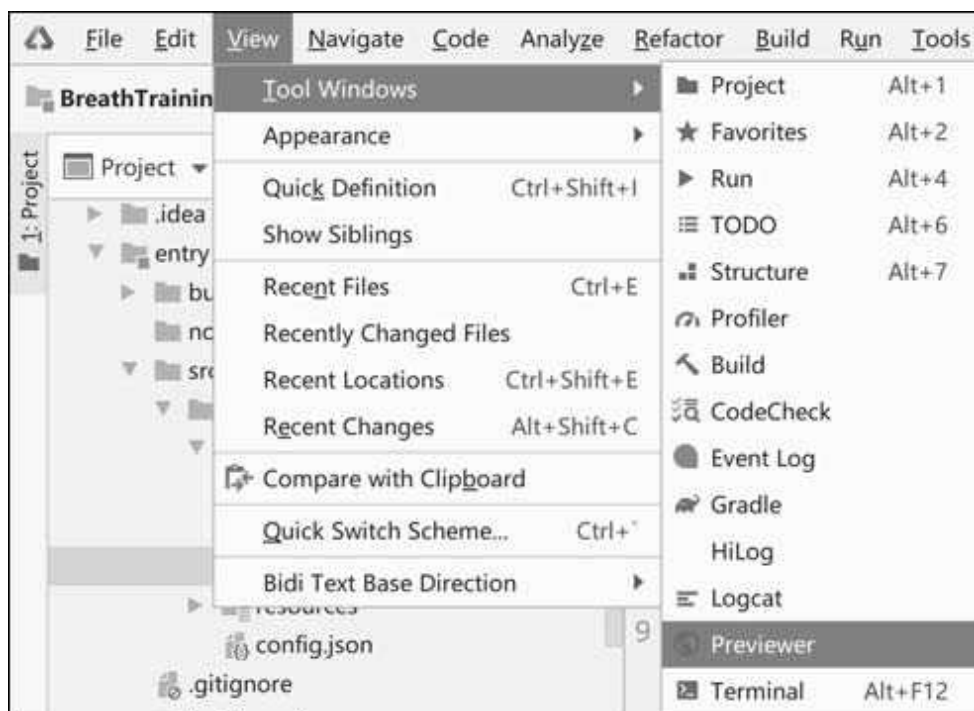


图2-40 菜单栏中的菜单项Previewer



图2-41 Previewer

为了更方便地操作Previewer，可以单击右上角的Show Options Menu设置按钮，如图2-42所示。

在打开的选项菜单中选中View Mode，然后单击Window命令，如图2-43所示。



图2-42 Previewer

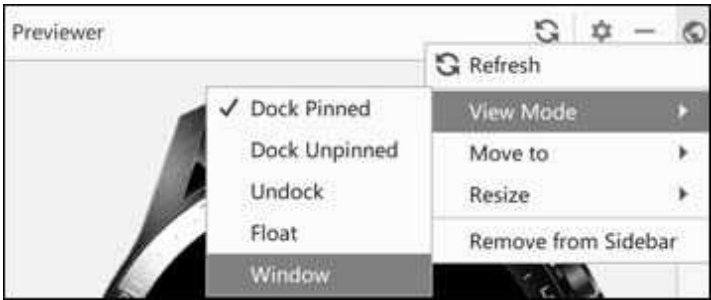


图2-43 设置Previewer的显示模式

这样，Previewer就会显示为一个非模态的浮动窗口——既可以将其最小化到任务栏，也可以将其最大化，如图2-44所示。

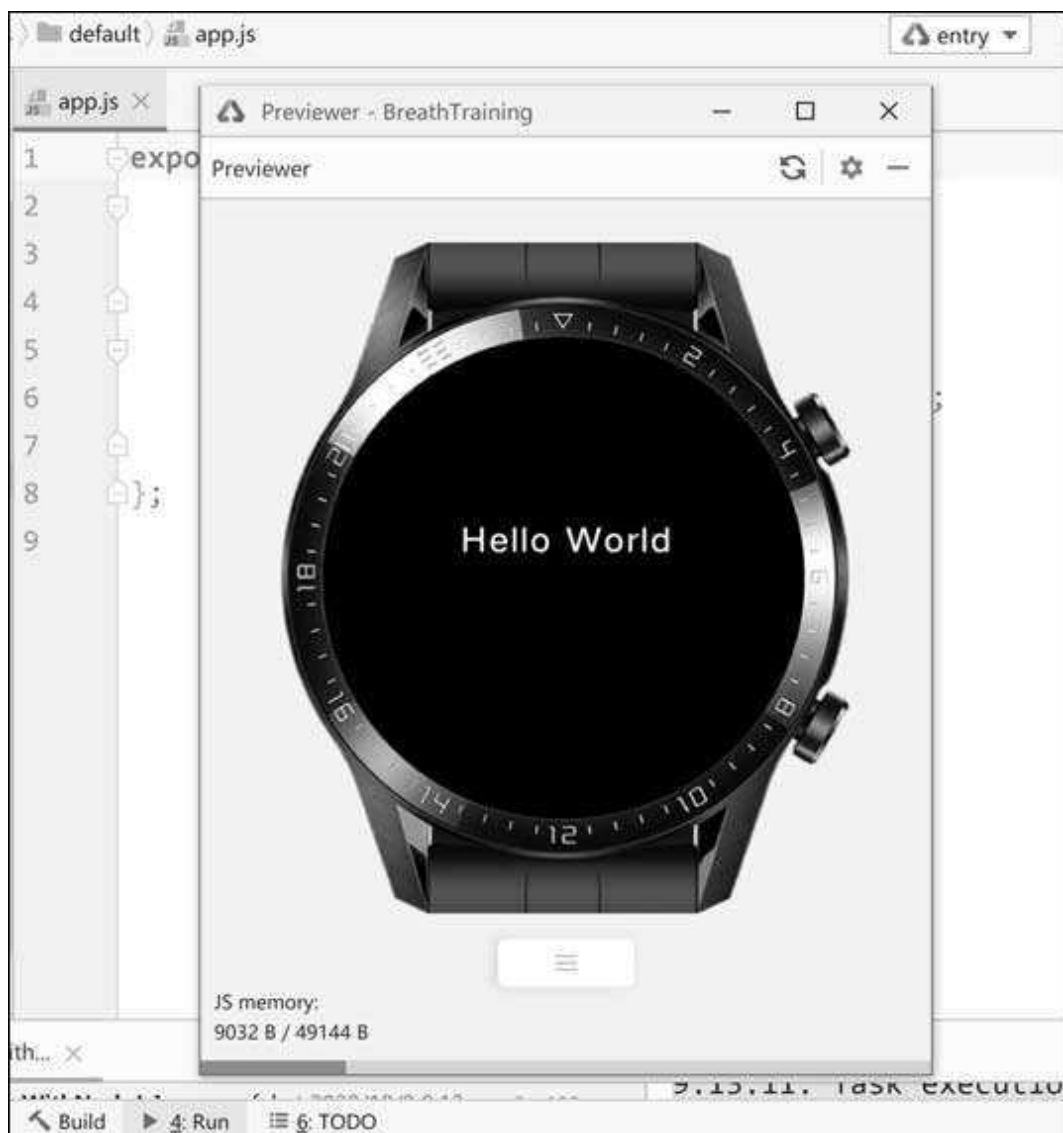


图2-44 显示为非模态浮动窗口的Previewer

单击手表下方的三根横线，会在窗口的右侧列出Previewer的众多设置选项，例如屏幕亮度、心率、步数、地理位置、音量等，如图2-45所示。

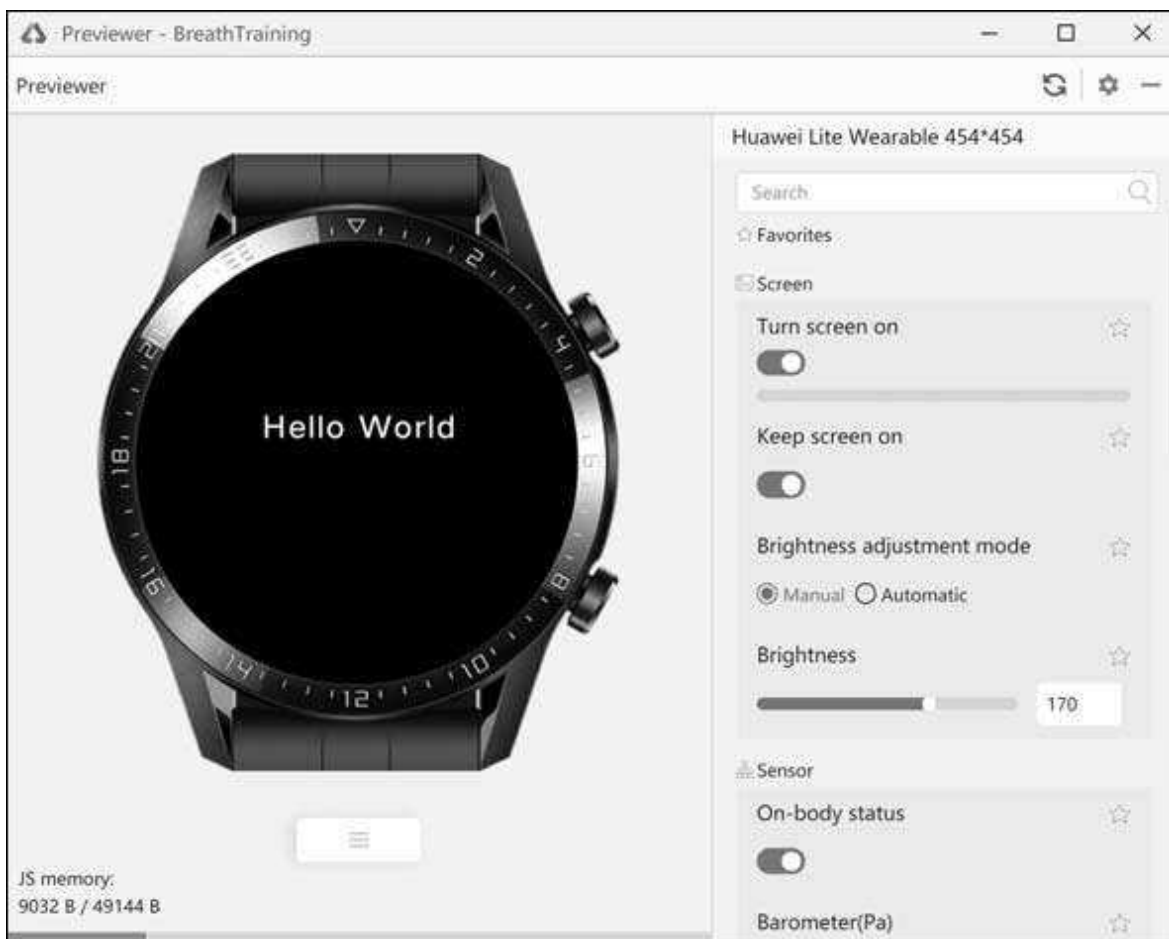


图2-45 Previewer的众多设置选项

再次单击三根横线，Previewer的设置选项就被隐藏起来了。

接下来，我们简单分析一下这个运行在智能手表上的Hello World项目。

在项目中依次展开entry/src/main/js/default/pages/index目录，在index目录中包含3个文件：index.html、index.css和index.js，如图2-46所示。这3个文件共同组成了我们在Previewer中看到的App主页面。

很多读者看到这里可能会想：原来开发鸿蒙智能手表的App使用的是Web前端的技术啊，那真的是太好了，因为Web前端的技术非常容易上手。使用Web前端技术开发鸿蒙智能手表的App的确降低了开发人员的门槛，不过HTML、CSS和JavaScript是浏览器渲染网页所使用的技

术，如果直接将其搬到鸿蒙智能手表上显然是不合适的。因此，鸿蒙对HTML、CSS和JavaScript做了很多裁剪和优化。例如，如果大家仔细看，会发现在index目录中有一个文件是index.html，而不是index.html。再例如，index.js文件只支持ECMAScript 5.1的语法，使用ECMAScript6语法编写的代码会被自动转换为ECMAScript 5.1语法的代码。

如果你是一位有经验的Web前端开发工程师，在开发鸿蒙智能手表的App时会相对比较轻松，但是仍然有一些细节需要注意，仍然有一些智能手表特有的技术知识点需要掌握。在本书的呼吸训练实战项目中，我会尽可能多地向大家呈现这些细节和特有的技术知识点。

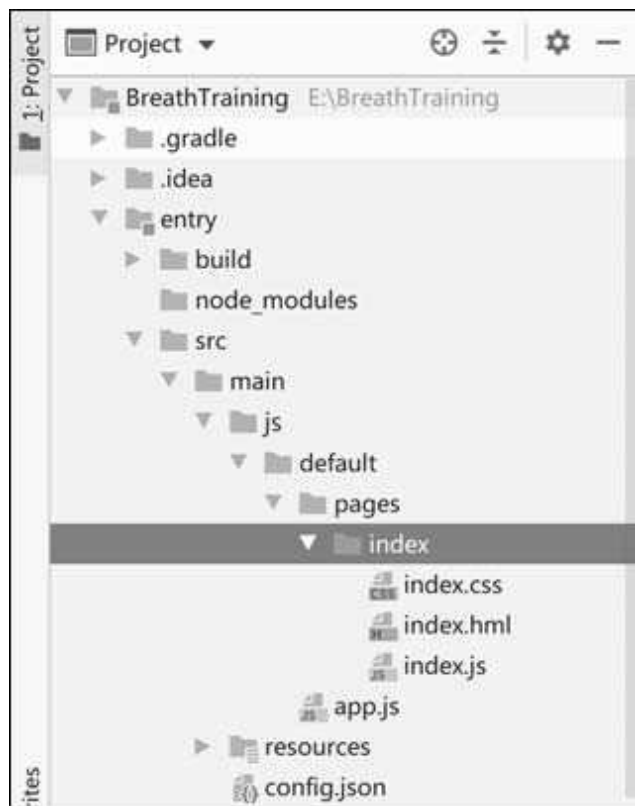


图2-46 项目中的子目录index

鸿蒙智能手表的任何一个页面，都对应着一个html文件、一个css文件和一个js文件，这三个文件的关系如图2-47所示。html文件是页面的结构，它用于描述页面中包含哪些组件；css文件是页面的样式，它用

于描述页面中的组件都长什么样；js文件是页面的行为，它用于描述页面中的组件是如何进行交互的。

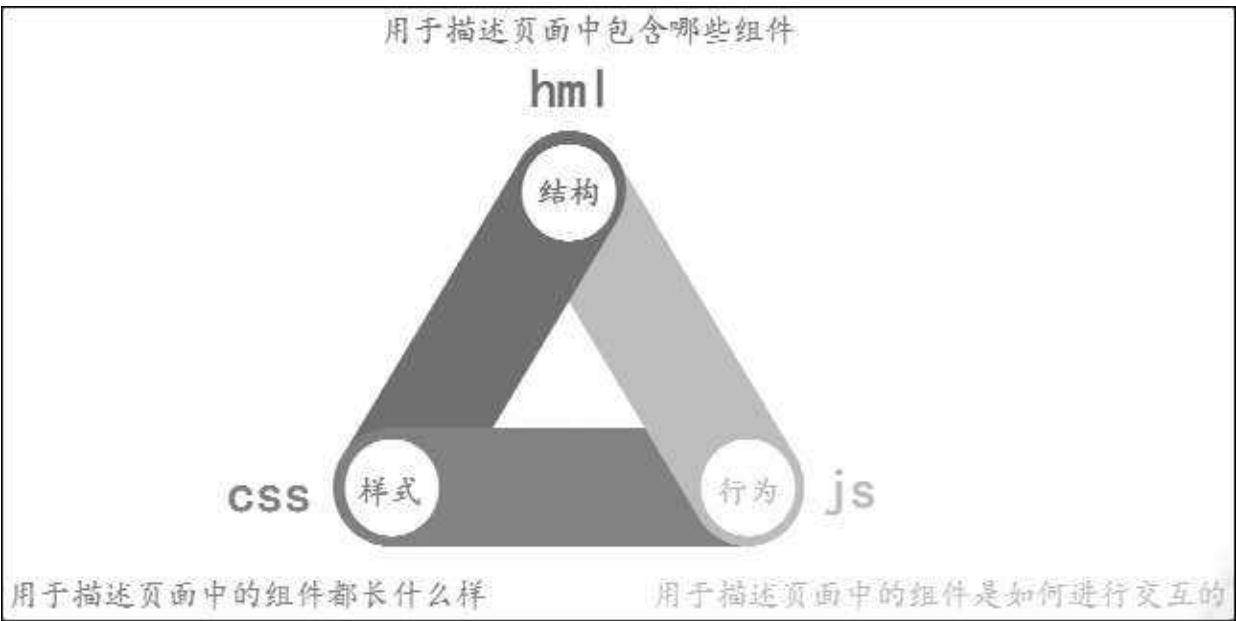


图2-47 页面对应的3个文件的关系

我们分别打开这3个文件看一下。

在文件index.html中定义了页面中包含的两个组件：一个组件是容器div；另一个组件是文本框text，如图2-48所示。

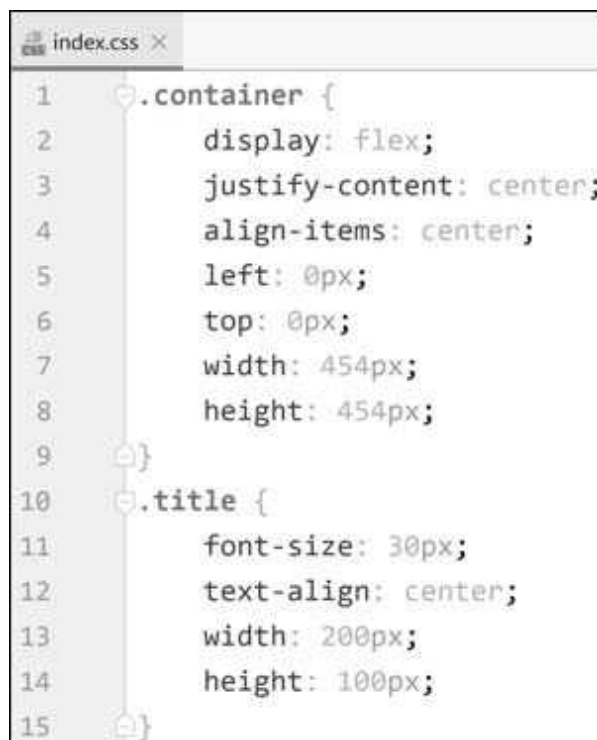
我们在Previewer中看到的文本“Hello World”就显示在这个text文本框中。

在文件index.css中通过类选择器定义了div和text这两个组件分别长什么样，比如宽度、高度、字体大小、对齐方式等，如图2-49所示。

```
index.html x
1 <div class="container">
2   <text class="title">
3     Hello {{title}}
4   </text>
5 </div>
```

图2-48 文件index.html





```
1  .container {
2      display: flex;
3      justify-content: center;
4      align-items: center;
5      left: 0px;
6      top: 0px;
7      width: 454px;
8      height: 454px;
9  }
10 .title {
11     font-size: 30px;
12     text-align: center;
13     width: 200px;
14     height: 100px;
15 }
```

图2-49 文件index.css

在类选择器container中，组件div的宽度（width）和高度（height）都被设置为了454px。454px是智能手表页面的最大宽度和最大高度。智能手表中页面的坐标系如图2-50所示。

中间的实心圆是智能手表的表盘。坐标原点是表盘的外接正方形的左上角。表盘的外接正方形的边长是454px，因此表盘的最大宽度和最大高度都是454px。所有位于表盘之外并且位于外接正方形之内的部分都不会被显示。此外，页面中所有组件的宽度和高度都可以使用具体的像素值来指定，例如，在index.css中text文本框的宽度和高度分别被指定为了200px和100px。

文件index.js中定义了一个变量title，它的值是'World'，如图2-51所示。这个title变量是index.html中使用两个花括号括起来的占位符。占位符title的值是在程序的运行过程中动态确定的，这种技术称之为动态数据绑定。

对于这个Hello World项目中的其他目录和文件，在后面的章节中我们用到哪一个就详细讲解哪一个，而不是一股脑地给大家全部介绍一遍。本书采用的是项目驱动和面向问题的学习方式，边做边学，在做中学，在学中做。这样，大家对知识点的理解就会更加深刻，对知识点的掌握也会更加容易。

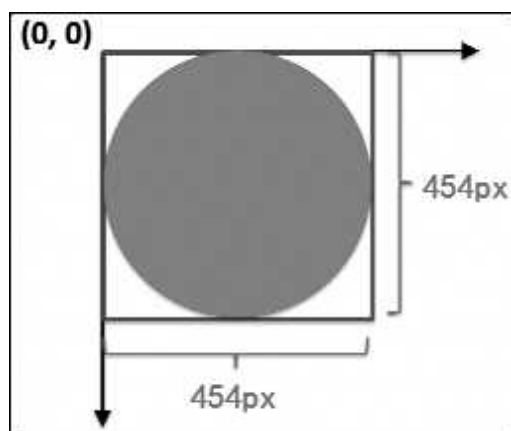


图2-50 页面的坐标系



图2-51 文件index.js

从下一章开始，我们就在这个Hello World项目的基础上不断地进行改造和完善，最终构建出一个完整的呼吸训练App。

## 第3章 呼吸训练实战项目

我们会在本章完成一个运行在鸿蒙智能手表上的名为呼吸训练的App。

主页面（见图3-1）的中部是鸿蒙呼吸训练的logo，logo的左右两边各有一个选择器。左边的选择器用于设定呼吸训练的时长，单位是分，可选值有1、2、3，默认值是2，也就是两分钟。右边的选择器用于设定每次呼气或吸气的节奏，可选值有较慢、舒缓、较快，默认值是舒缓。用户可以根据自己的喜好对时长和节奏进行设定。

设定好之后，单击logo下方的“点击开始”按钮。单击之后，跳转到3秒倒计时页面，显示“请保持静止，秒后跟随训练指引进行吸气和呼气”。倒计时页面如图3-2所示。



图3-1 主页面



图3-2 倒计时页面

3秒倒计时结束后，跳转到训练页面。根据主页面中设定的时长和节奏，每呼气或吸气一次，logo顺时针转动一周；logo的下方交替显示“呼气”和“吸气”，括号中显示的是当前呼气或吸气的百分比，每次呼气或吸气都会显示100次进度。再下方显示的是再坚持的秒数。正在训练时的训练页面如图3-3所示。

不管训练是否完成，在任何一个时刻，都可以单击最下方的“点击重新开始”按钮，单击之后，跳转到App的主页面。用户可以根据自己的喜好对时长和节奏重新进行设定，设定好之后重新开始一次呼吸训练。

当完成设定时长的训练后，页面中显示“已完成（100%）”，以及“右滑查看训练报告”。完成训练时的训练页面如图3-4所示。



图3-3 正在训练时的训练页面



图3-4 完成训练时的训练页面

在页面中向右滑动后，跳转到压力占比页面，如图3-5所示。

在压力占比页面中向上滑动后，跳转到心率曲线页面，如图3-6所示。



图3-5 压力占比页面



图3-6 心率曲线页面

在心率曲线页面中向上滑动后，跳转到今日活动分布页面，如图3-7所示。在今日活动分布页面中向上滑动后，跳转到压力分布页面，如

图3-8所示。在压力分布页面中向上滑动后，跳转到最大摄氧量页面，如图3-9所示。



图3-7 今日活动页面



图3-8 压力分布页面

在最大摄氧量页面中向上滑动后，跳转到学习交流联系方式页面，如图3-10所示。



图3-9 最大摄氧量页面



图3-10 学习交流联系方式页面



在学习交流联系方式页面中向下滑动后，跳转到最大摄氧量页面。在最大摄氧量页面中向下滑动后，跳转到压力分布页面。在压力分布页面中向下滑动后，跳转到今日活动分布页面。在今日活动分布页面中向下滑动后，跳转到心率曲线页面。在心率曲线页面中向下滑动后，跳转到压力占比页面。

对于上述6个页面，在任何一个页面中向左滑动后都会跳转到主页面。

上述页面就是我们学习完本章之后所完成的App的运行效果。

通过这个实战项目，我们可以掌握鸿蒙智能手表App开发的众多核心技能，并且极大地降低学习成本。学习完这个实战项目之后，大家再去看官方文档或者开发一个全新的App，就会感觉非常容易。

接下来让我们开启鸿蒙App项目实战之旅！

## 3.1 任务1：在主页面中添加一个按钮并响应其单击事件

### 3.1.1 运行效果

该任务实现的运行效果是这样的：在主页面的文本“Hello World”的下方显示一个按钮。运行效果如图3-11所示。

单击该按钮后在Debug工具窗口中打印一条log“我被点击了”。运行效果如图3-12所示。



图3-11 显示一个按钮

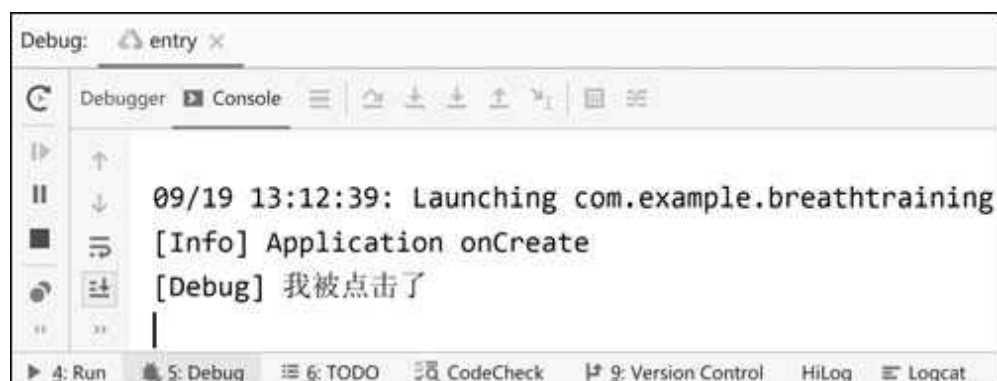


图3-12 打印log的Debug工具窗口

### 3.1.2 实现思路

使用input组件显示一个按钮。通过input组件的onclick属性指定一个自定义函数。这样，当单击按钮时就会触发按钮的onclick单击事件，从而自动调用onclick指定的自定义函数。

### 3.1.3 代码详解

打开index.html文件。

添加一个input组件。将type属性的值设置为“button”，以显示一个按钮。将属性value的值设置为“点我”，以设置按钮上显示的文本。将class属性的值设置为“btn”，以通过index.css中名为btn的类选择器设置按钮的样式。

上述讲解如代码清单3-1所示。

### 代码清单3-1 index.html

```
<div class="container">
  <text class="title">
    Hello {{title}}
  </text>
  <input type="button" value="点我" class="btn" />
</div>
```

打开文件index.css。

添加一个名为btn的类选择器，以设置按钮的样式。将width和height的值分别设置为200px和50px。

因为文本框和按钮是竖向排列的，所以在container类选择器中添加一个flex-direction样式，并将它的值设置为column，以竖向排列div容器内的所有组件。这样，因为无须再使用弹性布局的显示方式，所以就可以删掉display样式了。left和top这两个样式用于定位div容器在页面坐标系中的位置，其默认值都是0px，因此可以将left和top这两个样式都删掉。

上述讲解如代码清单3-2所示。

### 代码清单3-2 index.css

```
.container {
  flex-direction: column;
```

```
    display: flex;
    justify-content: center;
    align-items: center;
    left: 0px;
    top: 0px;
    width: 454px;
    height: 454px;
  }
  .title {
    font-size: 30px;
    text-align: center;
    width: 200px;
    height: 100px;
  }
  .btn {
    width: 200px;
    height: 50px;
  }
}
```

保存所有代码后打开Previewer，在主页面的文本“Hello World”的下方显示出了一个按钮。运行效果如图3-13所示。



图3-13 运行效果

接下来要实现的运行效果是：单击该按钮后打印一条log。

打开index.js文件。

定义一个名为clickAction的函数，并在函数体中打印一条log“我被点击了”。在data的右花括号和clickAction之间添加一个逗号。

上述讲解如代码清单3-3所示。

### 代码清单3-3 index.js

```
export default {
  data: {
    title: 'World'
  },
  clickAction() {
    console.log("我被单击了");
  }
}
```

打开index.html文件。

在input组件中添加一个onclick属性，并将它的值设置为刚刚定义的clickAction函数。这样，当单击按钮时就会触发按钮的onclick单击事件，从而自动调用clickAction函数。

上述讲解如代码清单3-4所示。

### 代码清单3-4 index.html

```
<div class="container">
  <text class="title">
    Hello {{title}}
  </text>
  <input type="button" value="点我" class="btn"
  onclick="clickAction" />
</div>
```

要想看到打印出来的log，必须以Debug模式运行应用。在DevEco Studio的工具栏中有一个小虫子图标样式的按钮，如图3-14所示。

单击该按钮，或者按Shift+F9组合键，会弹出一个对话框，用于选择部署应用的目标设备。在对话框中选中下面的“Huawei Lite Wearable Simulator”，然后单击OK按钮，如图3-15所示。

App的运行效果就显示在了模拟器Simulator的这个工具窗口中，而不是显示在预览器Previewer中。运行效果如图3-16所示。

与操作预览器Previewer类似，为了更方便地操作模拟器Simulator，可以单击右上角的Show Options Menu设置按钮，在打开的选项菜单中选中View Mode，然后单击Window，将Simulator的显示模式设置为非模态的浮动窗口，如图3-17所示。

单击主页面中的按钮后，在左下方的Debug工具窗口中就打印出了log“我被点击了”。运行效果如图3-18所示。



图3-14 Debug应用的小虫子图标按钮

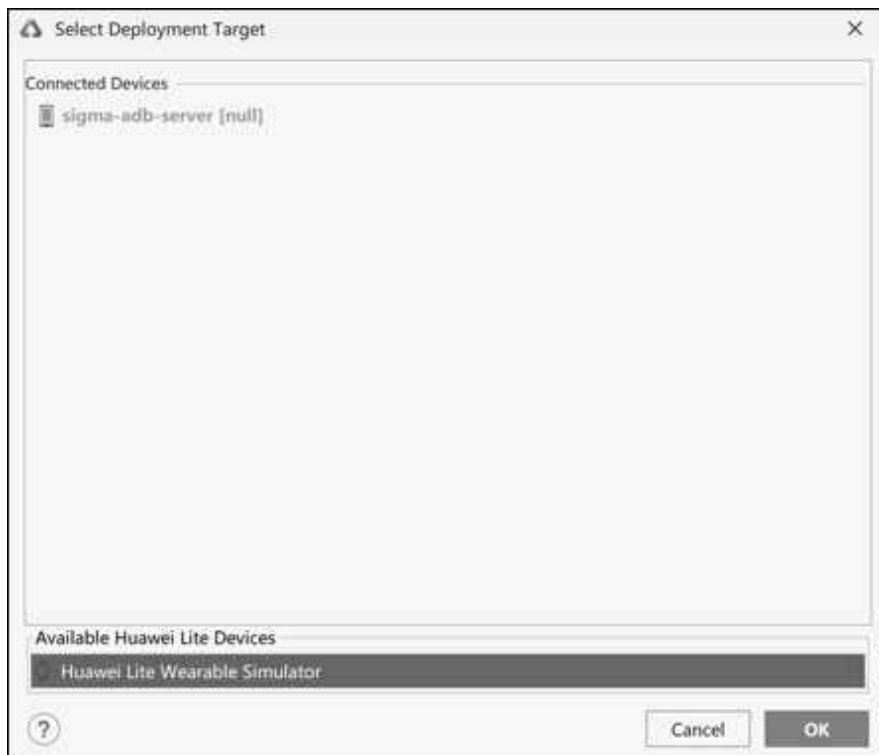


图3-15 选择部署应用的目标设备



图3-16 模拟器simulator工具窗口

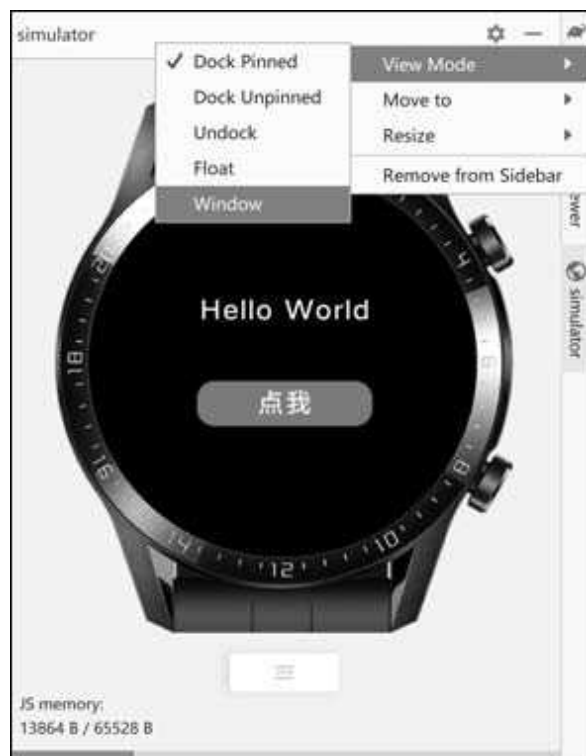


图3-17 设置simulator的显示模式

单击工具栏中的正方形按钮，停止在Debug模式下运行的应用，如图3-19所示。

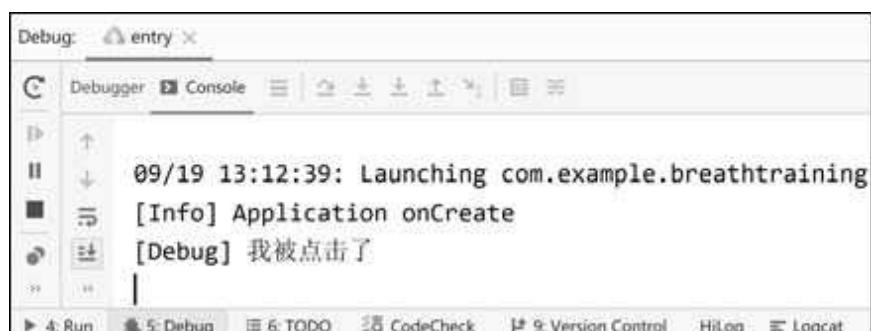


图3-18 打印log的Debug工具窗口



图3-19 停止在Debug模式下运行的应用的图标按钮

## 3.2 任务2：添加训练页面并实现其与主页面之间的相互跳转

### 3.2.1 运行效果

该任务实现的运行效果是这样的：单击主页面中的按钮，跳转到训练页面；单击训练页面中的按钮，跳转到主页面。运行效果如图3-20和图3-21所示。





图3-20 主页面



图3-21 训练页面

### 3.2.2 实现思路

把主页面和训练页面做一个对比，很容易发现：两个页面中所包含组件的结构以及对应组件的样式和行为几乎是一样的。因此，只需要在主页面的基础上稍作修改就可以实现训练页面了。

可以调用router.replace()语句实现页面间的跳转，在调用该语句时通过uri指定目标页面的地址。

### 3.2.3 代码详解

在项目的pages子目录上单击右键，在弹出的菜单中选中New，然后在弹出的子菜单中单击JS Page，以新建一个JS页面，如图3-22所示。

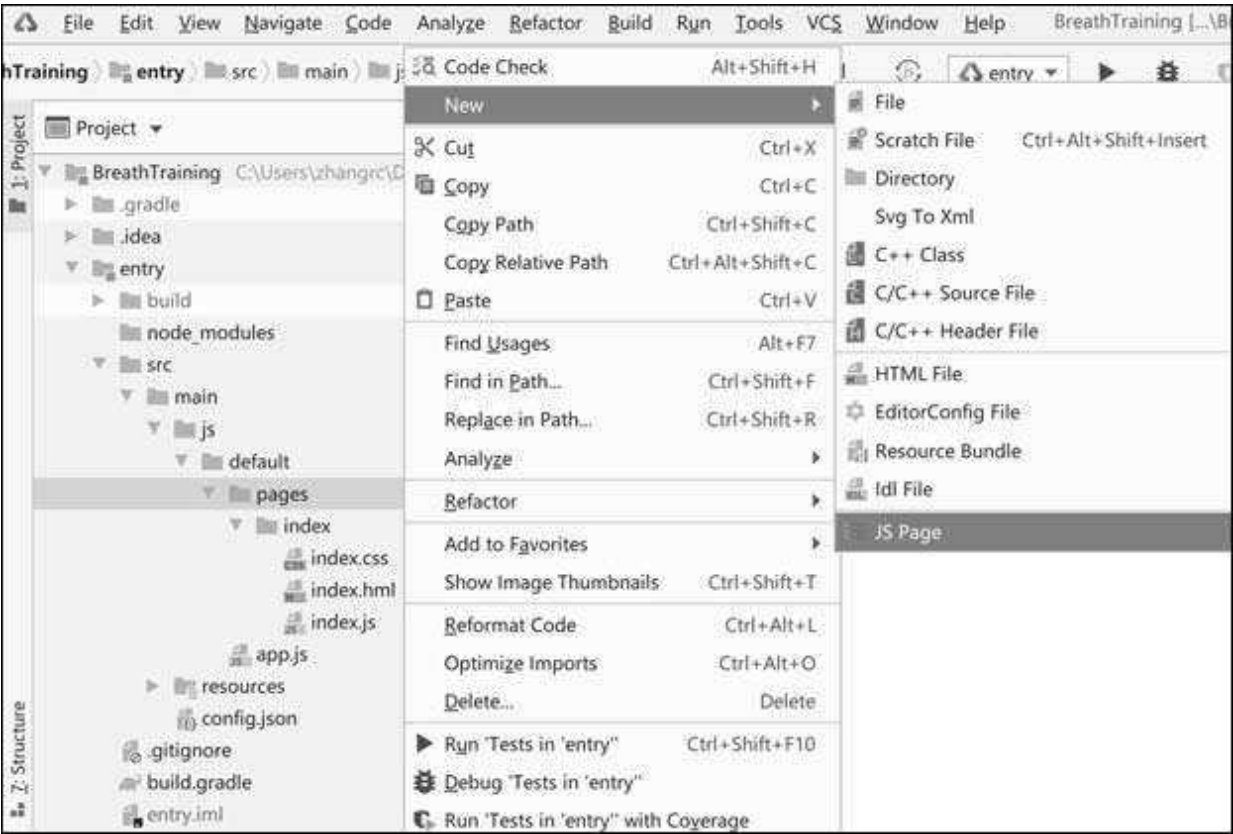


图3-22 新建一个JS页面

在打开的窗口中，将JS页面的名称设置为training，然后单击Finish按钮，如图3-23所示。

这样，在pages目录下就自动创建了一个名为training的子目录。该子目录中自动创建了3个文件：training.html、training.css和training.js，如图3-24所示。这3个文件共同组成了训练页面。

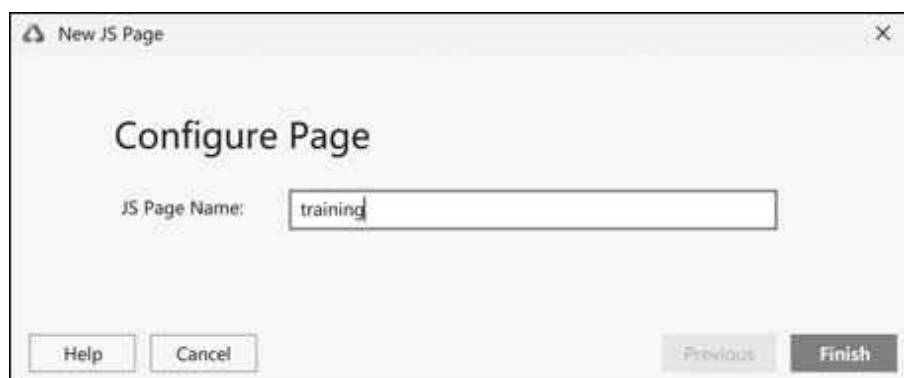


图3-23 配置JS页面的名称

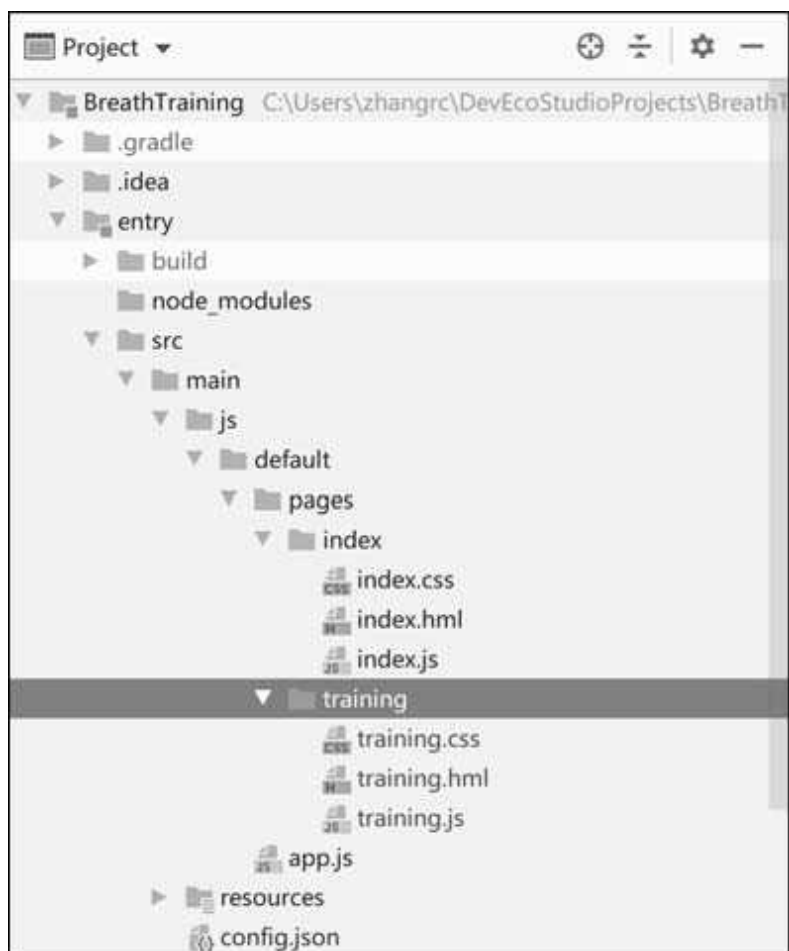


图3-24 自动创建的training子目录

为了能够在主页面的基础上稍作修改，可复制index.html中的所有内容，将其粘贴到training.html中。在training.html中，将text组件中的显示文本修改为“训练页面”，并将input组件中value属性的值修改为“返回”。

上述讲解如代码清单3-5所示。

#### 代码清单3-5 training.html

```
<div class="container">
  <text class="txt">
    训练页面
  </text>
```

```
<input type="button" value="返回" class="btn"
onclick="clickAction" />
</div>
```

---

复制index.css中的所有内容，粘贴到training.css中。

复制index.js中的所有内容，粘贴到training.js中。

因为在training.html中没有使用title占位符，所以在training.js中删除title及其动态数据绑定的值'World'。

为了能够在训练页面中单击按钮之后跳转到主页面，可在clickAction函数的函数体中删除打印log的语句，并且添加一条页面跳转语句router.replace()；。从'@system.router'中导入router，并且在一对花括号中将主页面的uri设置为'pages/index/index'。

上述讲解如代码清单3-6所示。

#### 代码清单3-6 training.js

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

export default {
  data: {
    title: 'World'
  },
  clickAction() {
    console.log("我被单击了");
    router.replace({
      uri: 'pages/index/index'
    });
  }
}
```

---

为什么主页面的uri是'pages/index/index'呢？这是因为在项目的config.json文件中对主页面的uri进行了定义。所有页面的uri都需要在config.json中进行定义。当新建训练页面时，会在config.json中自动生成训练页面的uri：pages/training/training，如图3-25所示。



图3-25 config.json中定义的页面uri

打开index.js文件。

为了能够在主页面中单击按钮之后跳转到训练页面，可在clickAction函数的函数体中删除打印log的语句，并且添加一条页面跳转的语句router.replace()；。从'@system.router'中导入router，并且在在一对花括号中将训练页面的uri设置为'pages/training/training'。

上述讲解如代码清单3-7所示。

### 代码清单3-7 index.js

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

export default {
  data: {
    title: 'World'
  },
  clickAction() {
```

```
console.log("我被点击了");  
router.replace({  
  uri: 'pages/training/training'  
});  
}  
}
```

---

保存所有代码后打开Previewer，单击主页面中的按钮，跳转到了训练页面；单击训练页面中的按钮，跳转到了主页面。运行效果如图3-26和图3-27所示。



图3-26 主页面



图3-27 训练页面

## 3.3 任务3：验证应用和每个页面的生命周期事件

### 3.3.1 运行效果

该任务的运行效果是这样的：主页面显示后，在Debug工具窗口中依次打印log“应用正在创建”“主页面的onInit()正在被调用”“主页面的onReady()正在被调用”“主页面的onShow()正在被调用”。运行效果如图3-28所示。

以主页面跳转到训练页面后，在Debug工具窗口中依次打印log“主页面的onDestroy()正在被调用”“训练页面的onInit()正在被调用”“训练页面的onReady()正在被调用”“训练页面的onShow()正在被调用”。运行效果如图3-29所示。



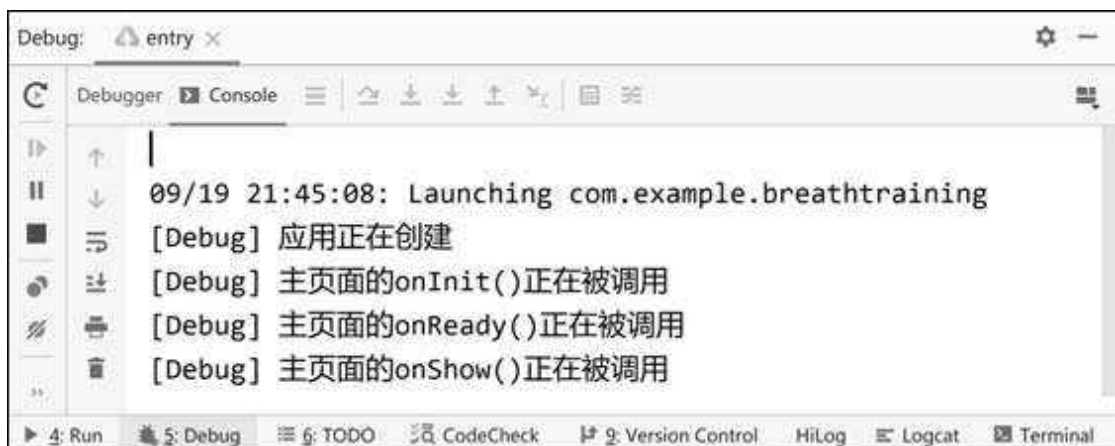


图3-28 主页面显示后打印出的log

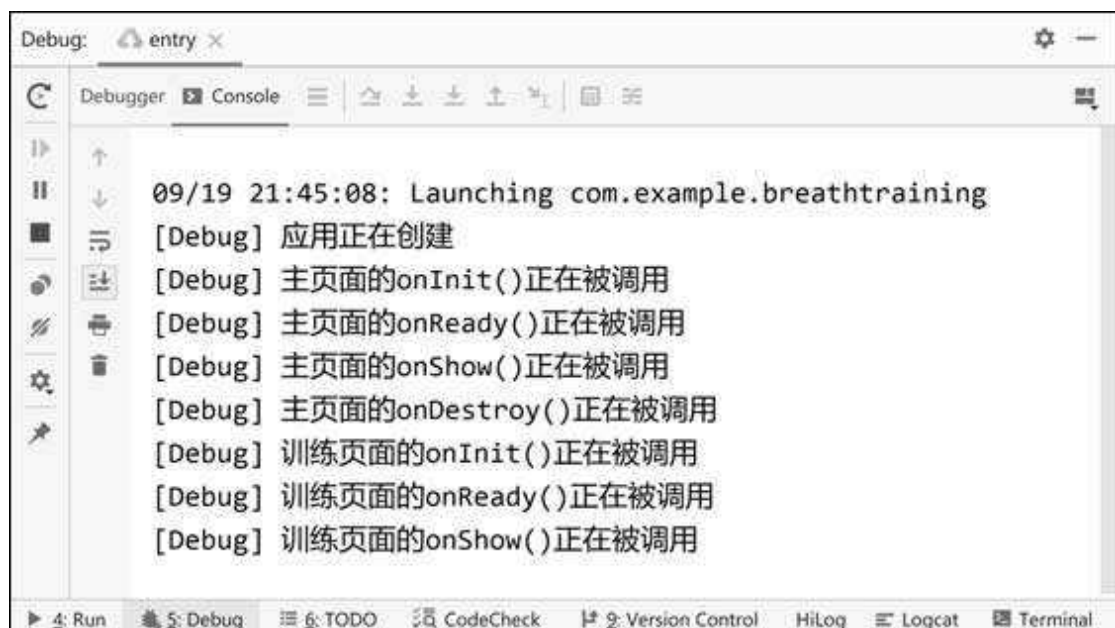


图3-29 主页面跳转到训练页面后打印出的log

### 3.3.2 实现思路

对于鸿蒙智能手表应用中的每个页面，在其从表盘上显示出来到其从表盘上消失的整个过程中，会在不同的阶段自动触发相应的生命

周期事件。对于应用而言，在其生命周期的整个过程中，也会在不同的阶段自动触发相应的生命周期事件。

应用和页面的生命周期事件可以用图3-30来表示。

应用的生命周期事件有两个：在应用创建时会触发onCreate事件；在应用销毁时会触发onDestroy事件。

页面的生命周期事件有4个，分别是onInit、onReady、onShow和onDestroy。其中，onInit事件表示页面的数据已经准备好，可以使用js文件中的数据；onReady事件表示页面已经编译完成，可以将页面显示给用户；onShow事件表示页面正在显示；onDestroy事件表示页面正在销毁。

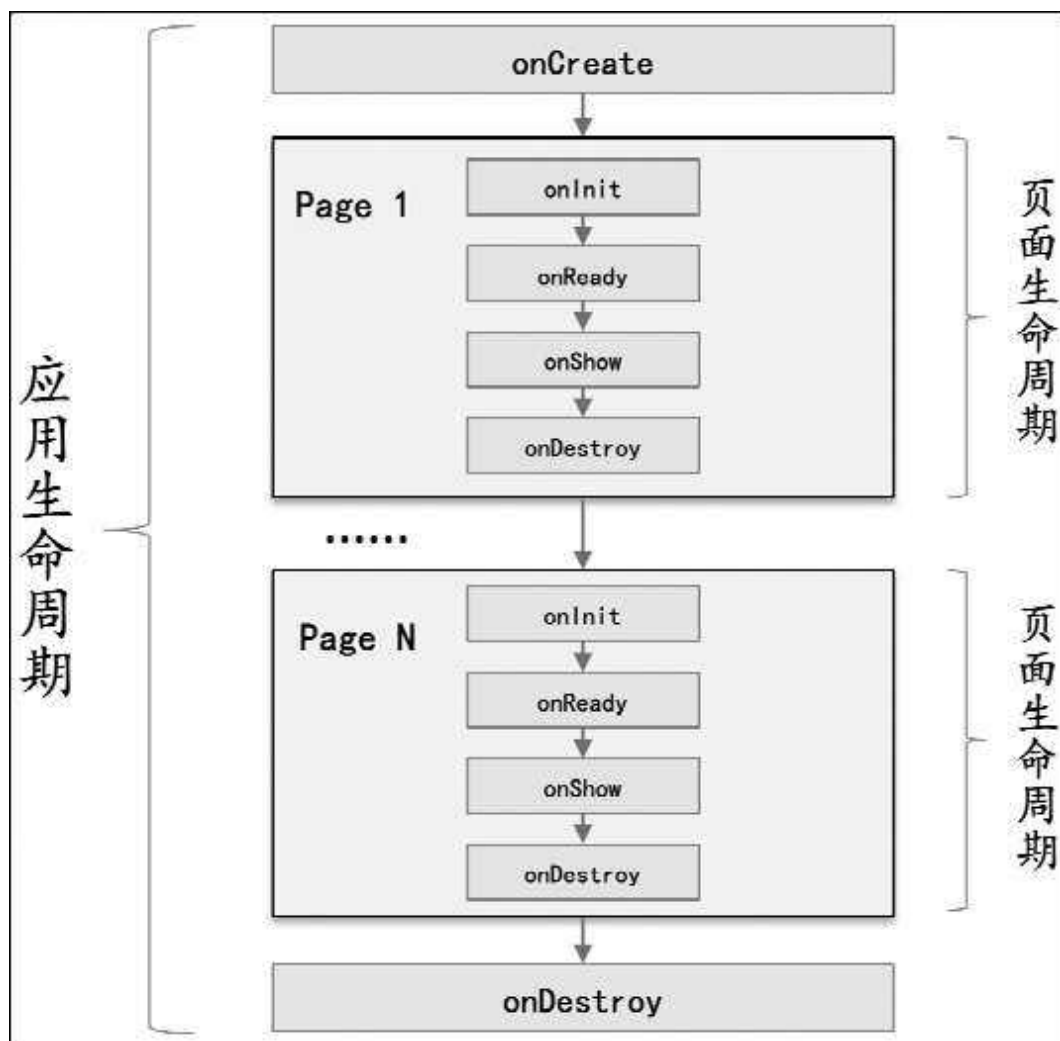


图3-30 应用和每个页面的生命周期事件

无论是应用还是页面，当触发某个生命周期事件时都会自动调用与该事件同名的函数。

### 3.3.3 代码详解

打开项目中的app.js文件，这个文件用于全局的JavaScript逻辑和应用的生命周期管理。在app.js中已经默认实现了应用的两个生命周期事件函数onCreate()和onDestroy()。在这两个函数的函数体中各打印了一条log，如图3-31所示。

```
1 export default {  
2   onCreate() {  
3     console.info("Application onCreate");  
4   },  
5   onDestroy() {  
6     console.info("Application onDestroy");  
7   }  
8 };
```

图3-31 默认实现的应用生命周期事件函数

为了能够打印出Debug级别的日志，我们将info修改为log。然后，修改打印的log文本，将“Application onCreate”修改为“应用正在创建”，并将“Application onDestroy”修改为“应用正在销毁”。

上述讲解如代码清单3-8所示。

代码清单3-8 app.js

```
export default {
  onCreate() {
    console.log("应用正在创建");
  },
  onDestroy() {
    console.log("应用正在销毁");
  }
};
```

---

打开index.js文件。

分别实现主页面的4个生命周期事件函数：onInit()、onReady()、onShow()和onDestroy()，在函数体中分别打印log“主页面的onInit()正在被调用”“主页面的onReady()正在被调用”“主页面的onShow()正在被调用”“主页面的onDestroy()正在被调用”。

上述讲解如代码清单3-9所示。

### 代码清单3-9 index.js

---

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

export default {
  data: {
    title: 'World'
  },
  clickAction() {
    router.replace({
      uri: 'pages/training/training'
    });
  },
  onInit() {
    console.log("主页面的onInit()正在被调用");
  },
  onReady() {
    console.log("主页面的onReady()正在被调用");
  },
  onShow() {
    console.log("主页面的onShow()正在被调用");
  },
  onDestroy() {
    console.log("主页面的onDestroy()正在被调用");
  }
};
```

```
}  
}
```

---

打开training.js文件。

分别实现训练页面的4个生命周期事件函数：onInit()、onReady()、onShow()和onDestroy()，在函数体中分别打印log“训练页面的onInit()正在被调用”“训练页面的onReady()正在被调用”“训练页面的onShow()正在被调用”“训练页面的onDestroy()正在被调用”。

上述讲解如代码清单3-10所示。

### 代码清单3-10 training.js

---

```
import router from '@system.router'  
  
export default {  
  data: {  
  
  },  
  clickAction() {  
    router.replace({  
      uri: 'pages/index/index'  
    });  
  },  
  onInit() {  
    console.log("训练页面的onInit()正在被调用");  
  },  
  onReady() {  
    console.log("训练页面的onReady()正在被调用");  
  },  
  onShow() {  
    console.log("训练页面的onShow()正在被调用");  
  },  
  onDestroy() {  
    console.log("训练页面的onDestroy()正在被调用");  
  }  
}
```

---

为了能够看到打印出来的log，我们在保存所有代码后以Debug模式运行应用。单击工具栏中的小虫子图标，在弹出的对话框中选中下面的Huawei Lite Wearable Simulator，然后单击OK按钮。当Simulator中显示出主页面之后，打开Debug工具窗口就可以看到打印出的log了。首先打印log“应用正在创建”，然后依次打印log“主页面的onInit()正在被调用”“主页面的onReady()正在被调用”“主页面的onShow()正在被调用”。运行效果如图3-32所示。

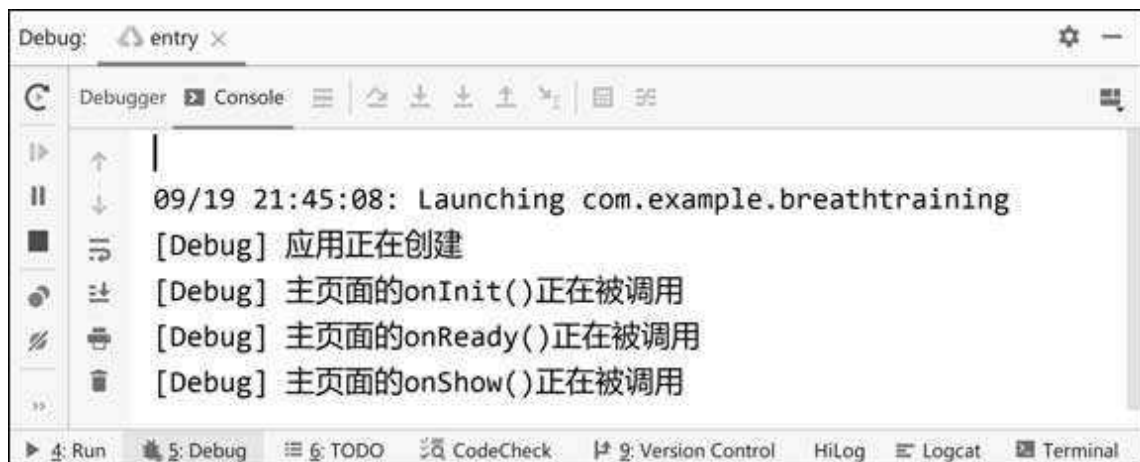


图3-32 主页面显示后打印出的log

在Simulator中单击主页面中的按钮，跳转到训练页面。Debug工具窗口中首先打印log“主页面的onDestroy()正在被调用”，然后依次打印log“训练页面的onInit()正在被调用”“训练页面的onReady()正在被调用”“训练页面的onShow()正在被调用”。运行效果如图3-33所示。

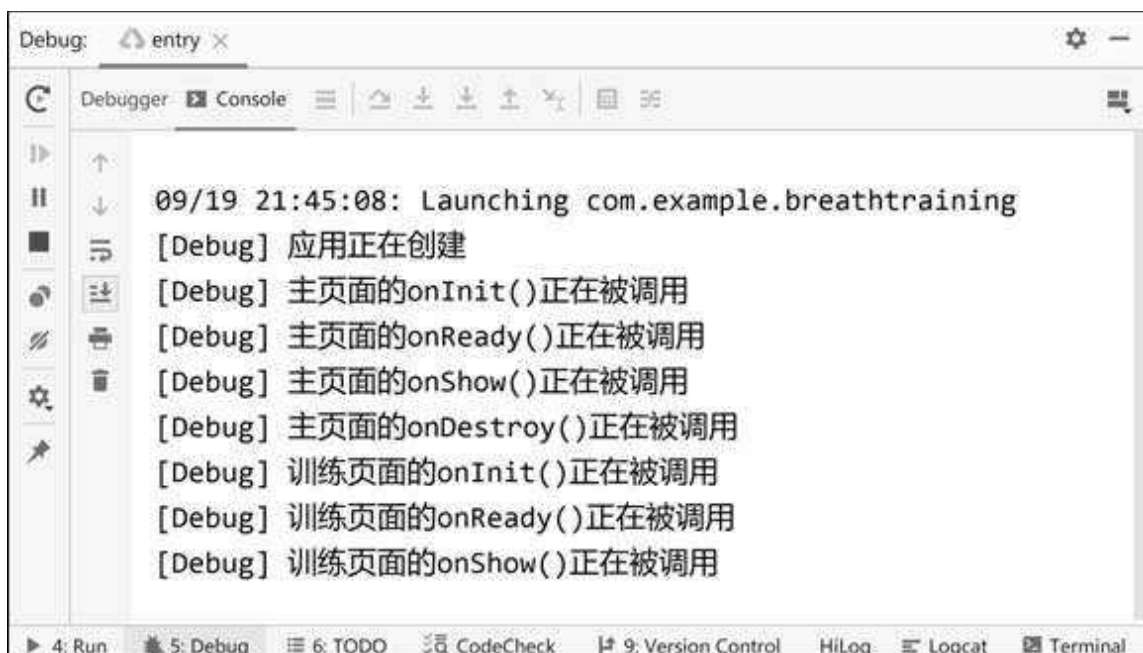


图3-33 主页面跳转到训练页面后打印出的log

通过打印出的log可知：当跳转到一个新页面时，当前页面就被销毁了。

注意：鸿蒙智能手表的应用中是没有后台页面的，在某一时刻只能运行并显示一个页面。

## 3.4 任务4：在主页面中显示logo和两个选择器

### 3.4.1 运行效果

该任务实现的运行效果是这样的：在主页面的中部显示鸿蒙呼吸训练的logo。logo的左右两边各显示一个选择器。其中，左边的选择器用于设定呼吸训练的时长，单位是分，可选值有1、2、3；右边的选择器用于设定每次呼气或吸气的节奏，可选值有较慢、舒缓、较快。运行效果如图3-34所示。



图3-34 显示logo和两个选择器的主页面

### 3.4.2 实现思路

使用image组件显示鸿蒙呼吸训练的logo。使用picker-view组件显示logo左右两边的选择器。

### 3.4.3 代码详解

在项目的/entry/src/main/js/default目录上单击右键，在弹出的菜单中选中New，然后在弹出的子菜单中单击Directory，以新建一个用于存放图片资源的目录，如图3-35所示。



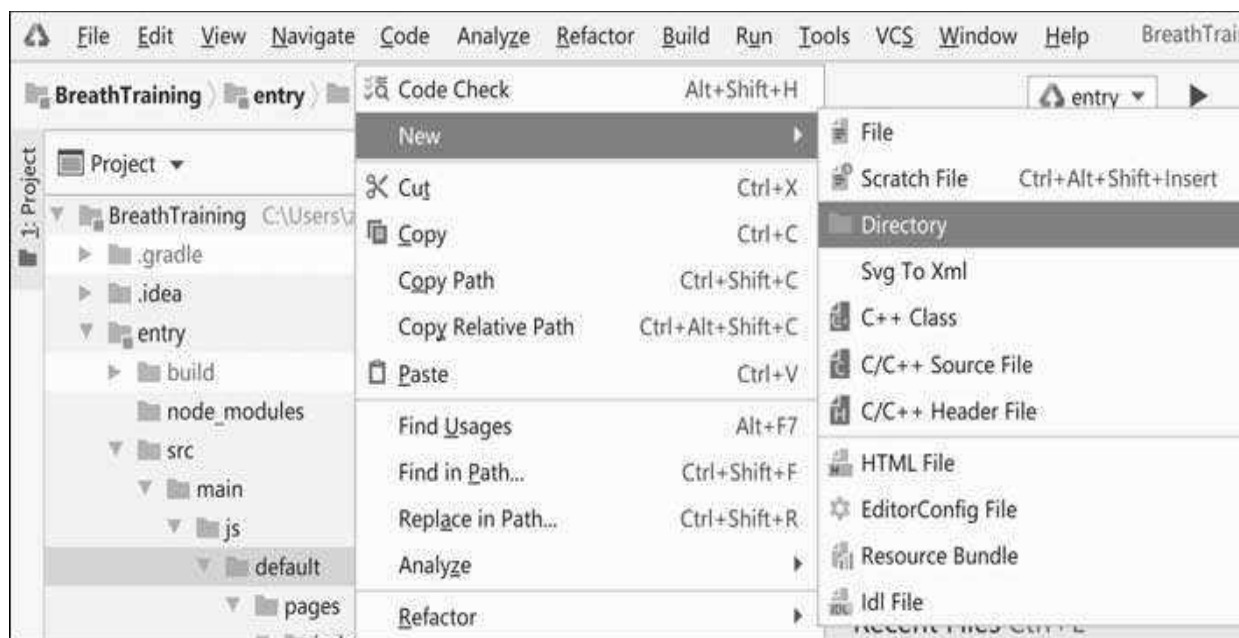


图3-35 新建一个用于存放图片资源的目录

在弹出的对话框中，输入新建目录的名称common，然后单击OK按钮，如图3-36所示。

把logo的图片hm.png添加到common目录中，如图3-37所示。



图3-36 输入新建目录的名称

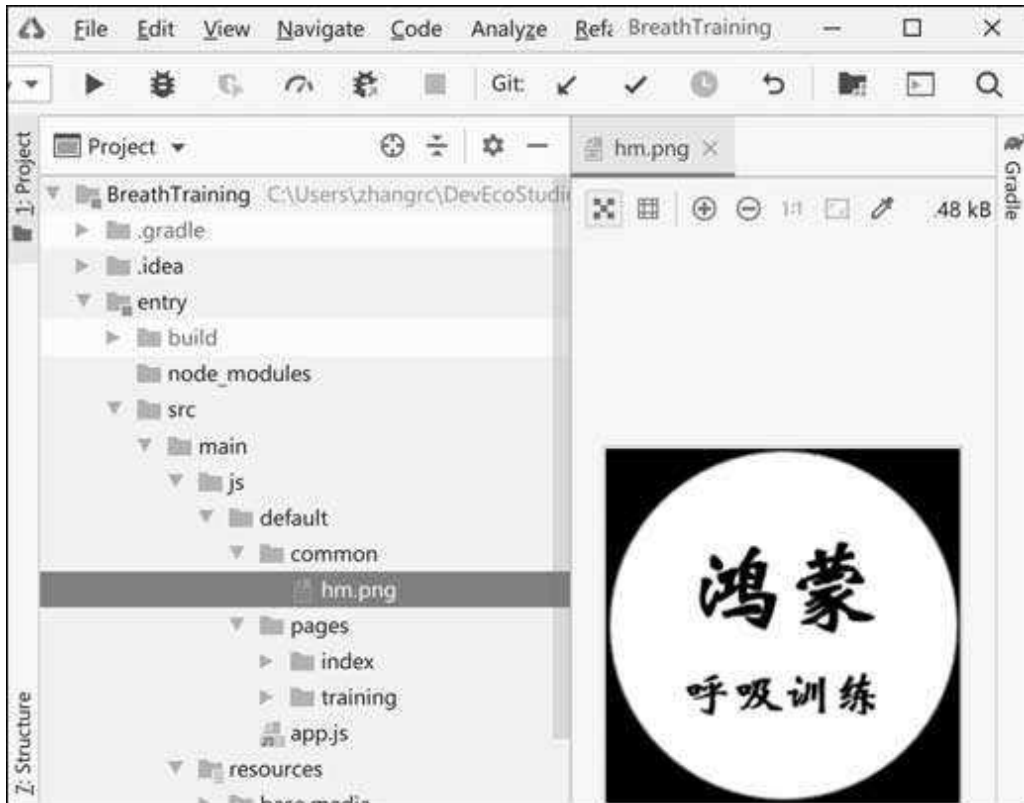


图3-37 目录common中的logo图片

打开index.html文件。

删除text组件的所有内容。添加一个image组件，以显示logo的图片。将src属性的值设置为“/common/hm.png”，以指定logo图片在项目中的位置。将class属性的值设置为“img”，以通过index.css中名为img的类选择器设置组件image的样式。

上述讲解如代码清单3-11所示。

#### 代码清单3-11 index.html

```
<div class="container">
  <text class="title">
    Hello {{title}}
  </text>
  <image src="/common/hm.png" class="img" />
  <input type="button" value="点我" class="btn">
```

```
onclick="clickAction" />
</div>
```

---

打开index.js文件。

因为在index.html中删除了占位符title，所以删除title及其动态数据绑定的值'World'。

上述讲解如代码清单3-12所示。

### 代码清单3-12 index.js

---

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

export default {
  data: {
    title: 'World'
  },
  .....
}
```

---

打开index.css文件。

因为在index.html中删除了text组件，所以删除title类选择器的所有内容。

添加一个名为img的类选择器，以设置image组件的样式。因为logo图片的宽度和高度都是208px，所以将width和height的值都设置为208px。

上述讲解如代码清单3-13所示。

### 代码清单3-13 index.css

---

```
.container {
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 454px;
```

```
        height: 454px;
    }
    .title {
        font-size: 30px;
        text-align: center;
        width: 200px;
        height: 100px;
    }
    .img {
        width: 208px;
        height: 208px;
    }
    .btn {
        width: 200px;
        height: 50px;
    }
}
```

---

保存所有代码后打开Previewer，在主页面中显示出了logo的图片。运行效果如图3-38所示。

接下来要实现的运行效果是：在logo的左右两边各显示一个选择器。运行效果如图3-39所示。



图3-38 显示logo图片的主页面



图3-39 显示的logo和两个选择器的主页面

左边选择器的单位是“分”，这个“分”字并不是选择器的一部分，而是单独显示在一个文本框中。

通过观察主页面的结构，可以发现：左边的选择器、文本框、图片和右边的选择器这4个组件是横向排列的。因此，为了方便管理，可以把这4个组件单独放在一个容器中。

打开index.html文件。

在div容器组件的内部嵌套一个div容器组件，将其class属性的值设置为“container2”，以通过index.css中名为container2的类选择器设置其样式。

对于外部的div容器组件，将其class属性的值设置为“container1”。

把image组件转移到最外层的div组件中。

在image组件的上方和下方各添加一个picker-view选择器组件，将class属性的值分别设置为“pv1”和“pv2”，并且通过动态数据绑定的方式将range属性的值分别设置为“{{picker1range}}”和“{{picker2range}}”。

在image组件的上方添加一个text组件。将class属性的值设置为“txt”，并将显示的文本设置为“分”。

上述讲解如代码清单3-14所示。

#### 代码清单3-14 index.html

```
<div class="container1">
  <div class="container2">
    <picker-view class="pv1" range="{{picker1range}}" />
    <text class="txt">
      分
    </text>
    <image src="/common/hm.png" class="img" />
    <picker-view class="pv2" range="{{picker2range}}" />
  </div>
  <input type="button" value="点我" class="btn"
onclick="clickAction" />
</div>
```

打开index.css文件。

将container类选择器的名称修改为container1。

添加一个container2类选择器，以设置div嵌套容器组件的样式。将flex-direction的值设置为row，以横向排列容器内的所有组件。将justify-content和align-items的值都设置为center，让容器内的组件在水平方向和竖直方向都居中显示。将margin-top的值设置为50px，让嵌套容器组件与表盘的上边缘保持一定的间距。将width的值设置为454px，并将height的值设置为250px。

添加一个pv1类选择器，以设置左边的picker-view选择器组件的样式。将width和height的值分别设置为30px和250px。

添加一个pv2类选择器，以设置右边的picker-view选择器组件的样式。将width和height的值分别设置为80px和250px。

添加一个txt类选择器，以设置显示“分”的这个文本框的样式。将text-align的值设置为center，让文本框中的文本在水平方向上居中显示。将width和height的值分别设置为50px和36px。

为了在logo的左右两边各留出一段空隙，在img类选择器中将margin-left和margin-right的值都设置为15px。

上述讲解如代码清单3-15所示。

### 代码清单3-15 index.css

```
.container1 {
    flex-direction: column;
    justify-content: center;
    align-items: center;
    width: 454px;
    height: 454px;
}
.container2 {
    flex-direction: row;
    justify-content: center;
    align-items: center;
    margin-top: 50px;
    width: 454px;
    height: 250px;
}
.pv1 {
    width: 30px;
    height: 250px;
}
.pv2 {
    width: 80px;
    height: 250px;
}
.txt {
    text-align: center;
    width: 50px;
    height: 36px;
}
.img {
    width: 208px;
    height: 208px;
    margin-left: 15px;
```

```
        margin-right: 15px;
    }
    .btn {
        width: 200px;
        height: 50px;
    }
}
```

---

打开index.js文件。

对于index.html中的占位符picker1range和picker2range，在data中将这两个占位符的值分别指定为：["1", "2", "3"]和["较慢", "舒缓", "较快"]。

上述讲解如代码清单3-16所示。

#### 代码清单3-16 index.js

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

export default {
    data: {
        picker1range: ["1", "2", "3"],
        picker2range: ["较慢", "舒缓", "较快"]
    },
    clickAction() {
        router.replace({
            uri: 'pages/training/training'
        });
    },
    .....
}
```

---

保存所有代码后打开Previewer，在logo的左右两边各显示出了一个选择器。其中，左边的选择器用于设定呼吸训练的时长，单位是分，可选值有1、2、3；右边的选择器用于设定每次呼气或吸气的节奏，可选值有较慢、舒缓、较快。运行效果如图3-40所示。





图3-40 显示logo和两个选择器的主页面

## 3.5 任务5：指定选择器的默认选中项并获取选中项的值

### 3.5.1 运行效果

该任务实现的运行效果是这样的：主页面中的两个选择器都默认选中了中间的选项。运行效果如图3-41所示。

当手动改变某个选择器的选中项时，会将选中项的值打印到log中。运行效果如图3-42所示。



图3-41 默认选中中间选项的两个选择器

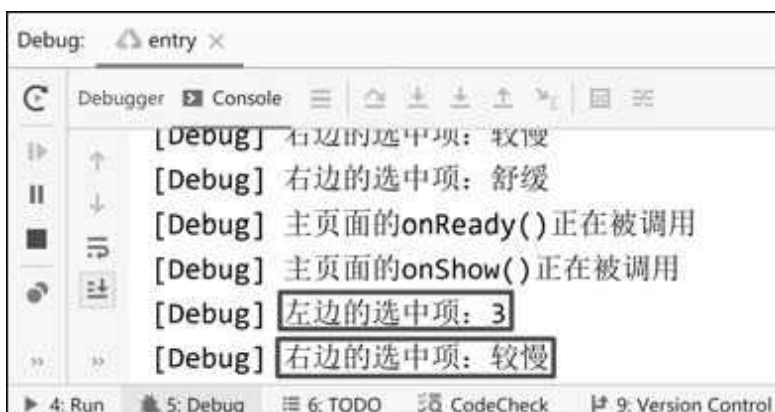


图3-42 log中打印的手动选中项的值

## 3.5.2 实现思路

通过selected属性设置选择器的默认选中项。当改变某个选择器的选中项时会触发onchange事件。可以在触发onchange事件时让系统自动调用我们指定的自定义函数。系统会将选中项的值传递给自定义函数的形参。这样，就可以在函数体中通过形参获取选中项的值了。

### 3.5.3 代码详解

打开index.html文件。

在两个选择器组件picker-view中各添加一个selected属性，并将属性值都设置为“1”。

上述讲解如代码清单3-17所示。

代码清单3-17 index.html

```
<div class="container1">
  <div class="container2">
    <picker-view class="pv1" range="{{picker1range}}"
selected="1" />
    <text class="txt">
      分
    </text>
    <image src="/common/hm.png" class="img" />
    <picker-view class="pv2" range="{{picker2range}}"
selected="1" />
  </div>
  <input type="button" value="点我" class="btn"
onclick="clickAction" />
</div>
```

保存所有代码后打开Previewer，主页面中的两个选择器都默认选中了中间的选项。运行效果如图3-43所示。



图3-43 默认选中中间选项的两个选择器

打开index.js文件。

添加一个名为changeAction1的自定义函数，以响应左边选择器的onchange事件。在函数中定义一个名为pv的形参，这样，在函数体中就可以通过pv.newValue获取选中项的值了。获取之后，将其打印到log中。

同样，再添加一个名为changeAction2的自定义函数，以响应右边选择器的onchange事件。在函数体中将选中项的值打印到log中。

上述讲解如代码清单3-18所示。

#### 代码清单3-18 index.js

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

export default {
  .....
  clickAction() {
```

```
        router.replace({
          uri: 'pages/training/training'
        });
      },
      changeAction1(pv) {
        console.log("左边的选中项：" + pv.newValue);
      },
      changeAction2(pv) {
        console.log("右边的选中项：" + pv.newValue);
      },
      .....
    }
  }
```

---

打开index.html文件。

在两个选择器组件picker-view中各添加一个onchange属性，并将属性值分别设置为两个自定义函数的函数名：changeAction1和changeAction2。

上述讲解如代码清单3-19所示。

#### 代码清单3-19 index.html

---

```
<div class="container1">
  <div class="container2">
    <picker-view class="pv1" range="{{picker1range}}"
selected="1" onchange="changeAction1" />
    <text class="txt">
      分
    </text>
    <image src="/common/hm.png" class="img" />
    <picker-view class="pv2" range="{{picker2range}}"
selected="1" onchange="changeAction2" />
  </div>
  <input type="button" value="点我" class="btn"
onclick="clickAction" />
</div>
```

---

当改变某个选择器的选中项时会触发onchange事件，系统就会自动调用我们指定的changeAction1或changeAction2自定义函数，从而将选

中项的值传递给自定义函数的pv形参。这样，就可以在函数体中通过pv.newValue获取选中项的值了。

保存所有代码后以Debug模式运行应用。当Simulator中显示出主界面之后，在Debug工具窗口中依次打印出了log“左边的选中项：2”“右边的选中项：舒缓”。运行效果如图3-44所示。这说明对于我们指定的默认选中项，也会触发选择器的onchange事件。

将左边的选择器手动选择为“3”，此时会触发左边选择器的onchange事件，从而将选中项的值打印到log中。在Debug工具窗口中打印出了log“左边的选中项：3”。将右边的选择器手动选择为“较慢”，此时会触发右边选择器的onchange事件，从而将选中项的值打印到log中。在Debug工具窗口中打印出了log“右边的选中项：较慢”。运行效果如图3-45所示。



图3-44 log中打印的默认选中项的值

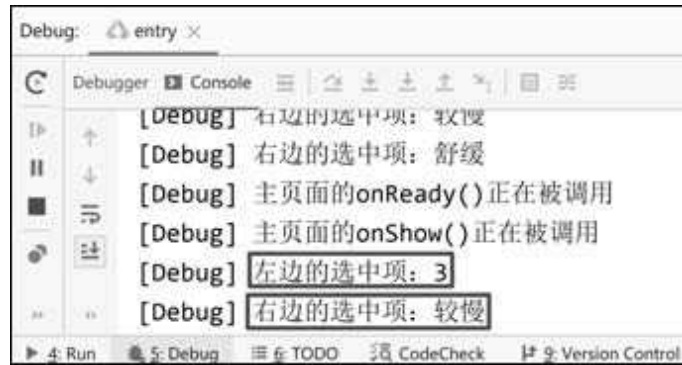


图3-45 log中打印的手动选中项的值

## 3.6 任务6：将主页面中选择器的值传递到训练页面

### 3.6.1 运行效果

该任务实现的运行效果是这样的：在主页面中将左边的选择器手动选择为“3”，并将右边的选择器手动选择为“较快”。单击主页面中的按钮以跳转到训练页面。在Debug工具窗口中依次打印log“接收到的左边选择器的值：3”“接收到的右边选择器的值：较快”。运行效果如图3-46所示。

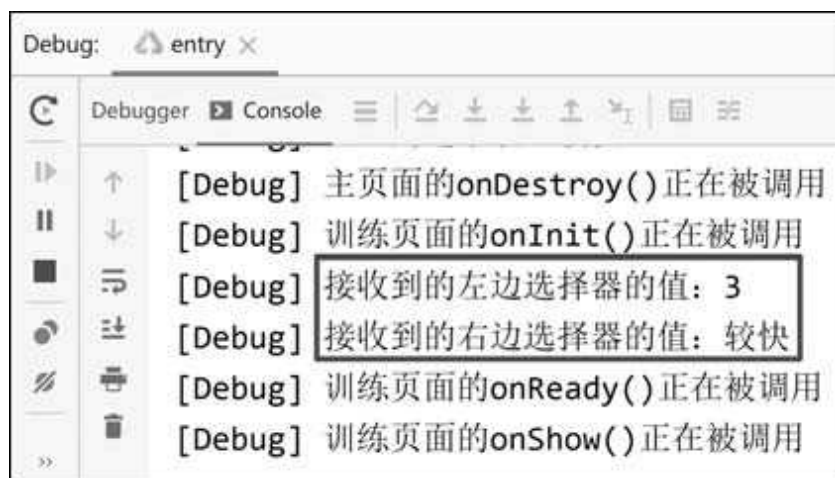


图3-46 log中打印的从主页面中传递过来的值

## 3.6.2 实现思路

在主页面跳转到训练页面时，通过params指定要传递的数据。所有要传递的数据都作为value存放在一个字典中。在训练页面的生命周期事件函数onInit()中，从字典中取出value以获取传递过来的数据。

## 3.6.3 代码详解

因为指定的默认选中项会触发选择器的onchange事件，所以在单击主页面中的按钮之前，在触发onchange事件时先将两个选中项的值保存起来。

打开index.js文件。

声明picker1value和picker2value两个全局变量，将其初始值都设置为null。

在changeAction1函数的函数体中，将选中项的值赋值给picker1value变量。在changeAction2函数的函数体中，将选中项的值赋



值给picker2value变量。

在单击主页面中的按钮进行页面跳转时，不仅要通过uri指定目标页面，还要通过params指定要传递的数据。所有要传递的数据都作为value存放在一个字典中。当跳转到训练页面时，我们可以将两个选择器的值存放在这样一个字典中：{"data1": picker1value, "data2": picker2value}。

上述讲解如代码清单3-20所示。

### 代码清单3-20 index.js

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

var picker1value = null;
var picker2value = null;

export default {
  data: {
    picker1range: ["1", "2", "3"],
    picker2range: ["较慢", "舒缓", "较快"]
  },
  clickAction() {
    router.replace({
      uri: 'pages/training/training',
      params: {"data1": picker1value, "data2": picker2value}
    });
  },
  changeAction1(pv) {
    console.log("左边的选中项：" + pv.newValue);
    picker1value = pv.newValue;
  },
  changeAction2(pv) {
    console.log("右边的选中项：" + pv.newValue);
    picker2value = pv.newValue;
  },
  .....
}
```

打开training.js文件。

在生命周期事件函数onInit()中获取主页面传递过来的值，并将其打印到log中。因为两个选择器的值存放在了一个字典中，并且对应的key分别是data1和data2，所以在训练页面中可以通过this.data1获取传递过来的左边选择器的值，并通过this.data2获取传递过来的右边选择器的值。

上述讲解如代码清单3-21所示。

#### 代码清单3-21 training.js

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

export default {
  data: {

  },
  .....
  onInit() {
    console.log("训练页面的onInit()正在被调用");

    console.log("接收到的左边选择器的值：" + this.data1);
    console.log("接收到的右边选择器的值：" + this.data2);
  },
  .....
}
```

保存所有代码后以Debug模式运行应用。在主页面中将左边的选择器手动选择为“3”，并将右边的选择器手动选择为“较快”。单击主页面中的按钮以跳转到训练页面。在Debug工具窗口中依次打印出了log“接收到的左边选择器的值：3”“接收到的右边选择器的值：较快”。

运行效果如图3-47所示。

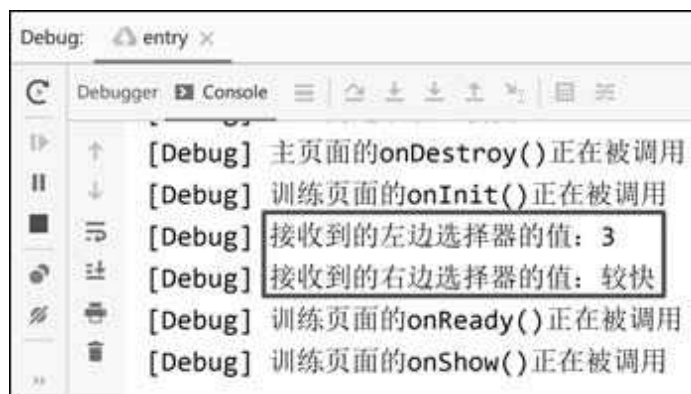


图3-47 log中打印的从主页面中传递过来的值

## 3.7 任务7：修改主页面和训练页面中按钮的文本及样式

### 3.7.1 运行效果

该任务实现的运行效果是这样的：在主页面中，按钮的背景色显示为黑色；按钮的文本显示为“单击开始”；按钮的文本比之前显示得大一些。运行效果如图3-48所示。

在训练页面中，按钮的背景色显示为黑色，按钮的文本显示为“单击重新开始”，按钮的文本比之前显示得大一些。此外，按钮与其上面的文本框的间隔距离比之前变大一些。运行效果如图3-49所示。



图3-48 修改文本及样式后主页面中的按钮



图3-49 修改文本及样式后训练页面中的按钮

### 3.7.2 实现思路

通过input组件的value属性的值修改按钮的文本。通过font-size样式修改按钮的文本大小。通过background-color样式修改按钮的背景色。通过border-color样式修改按钮的边框颜色。

### 3.7.3 代码详解

打开index.html文件。

将input组件的value属性的值修改为“单击开始”，以修改主页面上按钮的显示文本。

上述讲解如代码清单3-22所示。

#### 代码清单3-22 index.html

```
<div class="container1">
    .....
    <input type="button" value="单击开始" class="btn"
onclick="clickAction" />
</div>
```

打开index.css文件。

通过btn类选择器修改主页面上按钮的样式。添加一个font-size样式，其值为38px，以设置按钮的文本大小。添加一个background-color样式，其值为#000000，以将按钮的背景色设置为黑色。添加一个border-color样式，其值也为#000000，以将按钮的边框颜色也设置为黑色。

上述讲解如代码清单3-23所示。

#### 代码清单3-23 index.css

```

.....
.img {
    width: 208px;
    height: 208px;
    margin-left: 15px;
    margin-right: 15px;
}
.btn {
    width: 200px;
    height: 50px;
    font-size: 38px;
    background-color: #000000;
    border-color: #000000;
}

```

保存所有代码后打开Previewer，主页面中按钮的背景色显示为了黑色，按钮的文本显示为“单击开始”，按钮的文本比之前显示得大一些了。运行效果如图3-50所示。



图3-50 修改文本及样式后主页面中的按钮

接下来，对训练页面的按钮也做类似的修改。  
打开training.html文件。

将input组件的value属性的值修改为“单击重新开始”，以修改训练页面上按钮的显示文本。

上述讲解如代码清单3-24所示。

#### 代码清单3-24 training.html

```
<div class="container">
  <text class="title">
    训练页面
  </text>
  <input type="button" value="单击重新开始" class="btn"
onclick="clickAction" />
</div>
```

打开training.css文件。

通过btn类选择器修改训练页面上按钮的样式。添加一个font-size样式，其值为38px，以设置按钮的文本大小。文本变大了之后，可能会导致按钮的宽度不够，因此将width的值修改为300px。添加一个background-color样式，其值为#000000，以将按钮的背景色设置为黑色。添加一个border-color样式，其值也为#000000，以将按钮的边框颜色也设置为黑色。添加一个margin-top样式，其值为40px，以让按钮与其上面的文本框保持一定的间距。

上述讲解如代码清单3-25所示。

#### 代码清单3-25 training.css

```
.....
.btn {
  width: 300px;
  height: 50px;
  font-size: 38px;
  background-color: #000000;
  border-color: #000000;
```

```
margin-top: 40px;  
}
```

---

保存所有代码后打开Previewer，训练页面中按钮的背景色显示为了黑色，按钮的文本显示为“单击重新开始”，按钮的文本比之前显示得大一些了。此外，按钮与其上面的文本框的间隔距离比之前变大了一些。运行效果如图3-51所示。



图3-51 修改文本及样式后训练页面中的按钮

## 3.8 任务8：在训练页面显示总共需要坚持的秒数

### 3.8.1 运行效果

该任务实现的运行效果是这样的：在主页面中将左边选择器的值手动选择为“3”，然后单击主页面中的按钮后跳转到训练页面，训练页



面中显示“总共需要坚持180秒”。运行效果如图3-52所示。



图3-52 显示总共需要坚持多少秒数的训练页面

### 3.8.2 实现思路

在训练页面显示之前，在其生命周期事件函数onInit()中对左边选择器的值进行转换，从而将分钟数转换为秒数。

### 3.8.3 代码详解

打开training.html文件。

将text组件的显示文本修改为“总共需要坚持{{seconds}}秒”。因为需要坚持的秒数取决于主页面中左边选择器的值，所以这里采用动态数据绑定的方式。

上述讲解如代码清单3-26所示。

#### 代码清单3-26 training.html

```
<div class="container">
  <text class="title">
    总共需要坚持 {{seconds}} 秒
  </text>
  <input type="button" value="单击重新开始" class="btn"
onclick="clickAction" />
</div>
```

打开training.css文件。

通过title类选择器修改文本框的样式。文本变多了之后，可能会导致文本框的宽度不够，因此将width的值修改为400px，将height的值修改为40px。

上述讲解如代码清单3-27所示。

#### 代码清单3-27 training.css

```
.....
.title {
  font-size: 30px;
  text-align: center;
  width: 400px;
  height: 40px;
}
.....
```

打开training.js文件。

在data中将seconds占位符的值初始化为0。

声明picker1value和picker2value两个全局变量，将其初始值都设置为null。

在生命周期事件函数onInit()中，将获取到的两个选择器的值分别赋值给picker1value和picker2value变量。

对于主页面中左边选择器的值，其单位是分，而在训练页面中要显示的是总共需要坚持的秒数，因此需要根据单位进行数值转换。声明一个全局变量picker1seconds，该变量用于保存左边选择器的值转换之后得到的秒数。将其初始值设置为null。在onInit()函数中对左边选择器的值进行转换，转换规则为：如果picker1value的值为“1”，就转换为60并赋值给picker1seconds；如果picker1value的值为“2”，就转换为120并赋值给picker1seconds；如果picker1value的值为“3”，就转换为180并赋值给picker1seconds。转换之后将picker1seconds赋值给this.seconds，这样在训练页面显示之前，seconds占位符被再次初始化为转换之后得到的秒数。

上述讲解如代码清单3-28所示。

### 代码清单3-28 training.js

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

var picker1value = null;
var picker2value = null;

var picker1seconds = null;

export default {
  data: {
    seconds: 0
  },
  clickAction() {
    router.replace({
      uri: 'pages/index/index'
    });
  },
  onInit() {
    console.log("训练页面的onInit()正在被调用");

    console.log("接收到的左边选择器的值：" + this.data1);
```

```
console.log("接收到的右边选择器的值：" + this.data2);

picker1value = this.data1;
picker2value = this.data2;

if(picker1value == "1") {
    picker1seconds = 60;
} else if(picker1value == "2") {
    picker1seconds = 120;
} else if(picker1value == "3") {
    picker1seconds = 180;
}

this.seconds = picker1seconds;
},
.....
}
```

---

注意：在把左边选择器的值转换为对应的秒数时，为了方便初学者学习，我们使用了if/else语句对其逐一进行转换。这种转换方式比较容易理解，但是它非常笨拙。按照JavaScript的语法，还有更简单的写法：`picker1seconds=picker1value*60`；。在JavaScript中，一个表示整数的字符串与一个整数相乘，运算结果为整数类型。

保存所有代码后打开Previewer，在主页面中将左边选择器的值手动选择为“3”，然后单击主页面中的按钮后跳转到训练页面，训练页面中显示出“总共需要坚持180秒”。运行效果如图3-53所示。



图3-53 显示总共需要坚持多少秒数的训练页面

## 3.9 任务9：在训练页面倒计时显示再坚持的秒数

### 3.9.1 运行效果

该任务实现的运行效果是这样的：单击主页面中的按钮跳转到训练页面。在训练页面中每隔1秒更新显示一次再坚持的秒数。运行效果如图3-54所示。



图3-54 倒计时显示再坚持的秒数

## 3.9.2 实现思路

在训练页面的生命周期事件函数onShow()中调用setInterval()函数创建一个定时器，并在调用时指定定时器要执行的动作以及时间间隔。

## 3.9.3 代码详解

打开training.html文件。

将text组件显示的文本“总共需要坚持”修改为“再坚持”。

上述讲解如代码清单3-29所示。

代码清单3-29 training.html

```
<div class="container">
  <text class="title">
    再坚持 {{seconds}} 秒
  </text>
  <input type="button" value="单击重新开始" class="btn"
onclick="clickAction" />
</div>
```

---

打开training.js文件。

为了能够每隔1秒更新显示一次再坚持的秒数，可以在训练页面正在显示时，也就是在生命周期事件函数onShow()中调用setInterval()函数创建一个timer1定时器，在调用时指定定时器要执行的动作以及时间间隔。

创建一个全局变量timer1，将其初始值设置为null。因为每隔1秒更新显示一次，所以将setInterval()的第2个实参指定为1000，单位是毫秒。第1个实参指定定时器要执行的动作。我们可以自定义一个名为run1的函数，然后将第1个实参指定为this.run1。

在run1()函数的函数体中，首先让this.seconds自减1，然后判断this.seconds是否为0，如果为0，那就清除timer1定时器并将timer1置为null。

当单击训练页面中的按钮跳转到主页面时，需要首先在clickAction()函数中清除timer1定时器并将timer1置为null。

上述讲解如代码清单3-30所示。

### 代码清单3-30 training.js

---

```
.....

var picker1seconds = null;

var timer1 = null;

export default {
```

```

data: {
    seconds: 0
},
clickAction() {
    clearInterval(timer1);
    timer1 = null;

    router.replace({
        uri: 'pages/index/index'
    });
},
run1() {
    this.seconds--;
    if(this.seconds == 0) {
        clearInterval(timer1);
        timer1 = null;
    }
},
.....
onShow() {
    console.log("训练页面的onShow()正在被调用");

    timer1 = setInterval(this.run1, 1000);
},
onDestroy() {
    console.log("训练页面的onDestroy()正在被调用");
}
}

```

---

保存所有代码后打开Previewer，单击主页面中的按钮跳转到训练页面。在训练页面中每隔1秒更新显示一次再坚持的秒数。运行效果如图3-55所示。





图3-55 倒计时显示再坚持的秒数

## 3.10 任务10：再坚持的秒数在倒计时结束时隐藏显示的文本

### 3.10.1 运行效果

该任务实现的运行效果是这样的：单击主页面中的按钮跳转到训练页面，训练页面中倒计时显示再坚持的秒数，再坚持的秒数在倒计时结束时会隐藏显示的文本，从而变为不可见。运行效果如图3-56所示。



图3-56 再坚持的秒数变为不可见

### 3.10.2 实现思路

将某个组件的show属性的值设置为false从而隐藏该组件。隐藏组件之后，它在页面中所占的空间仍然是存在的。

### 3.10.3 代码详解

打开training.html文件。

在text组件中添加一个show属性，该属性的默认值是true，表示显示text组件。如果将show属性的值设置为false，就会隐藏text组件。text组件在隐藏之后，它只是变为不可见，并没有从页面中删除，因此它所占的空间仍然是存在的。

通过动态数据绑定的方式指定show属性的值，将占位符的名称指定为isShow。

上述讲解如代码清单3-31所示。

#### 代码清单3-31 training.html

```
<div class="container">
  <text class="title" show="{{isShow}}">
    再坚持 {{seconds}} 秒
  </text>
  <input type="button" value="单击重新开始" class="btn"
onclick="clickAction" />
</div>
```

打开training.js文件。

在data中将isShow占位符初始化为true。

在倒计时结束时，将isShow占位符设置为false，从而隐藏文本框显示的文本，将其变为不可见。

上述讲解如代码清单3-32所示。

#### 代码清单3-32 training.html

```
.....

export default {
  data: {
    seconds: 0,
    isShow: true
  },
  clickAction() {
    clearInterval(timer1);
    timer1 = null;

    router.replace({
      uri: 'pages/index/index'
    });
  },
}
```

```
run1() {  
    this.seconds--;  
    if(this.seconds == 0) {  
        clearInterval(timer1);  
        timer1 = null;  
  
        this.isShow = false;  
    }  
},  
.....  
}
```

保存所有代码后打开Previewer，单击主页面中的按钮跳转到训练页面。训练页面中倒计时显示再坚持的秒数，再坚持的秒数在倒计时结束时会隐藏显示的文本，变为不可见。运行效果如图3-57所示。



图3-57 再坚持的秒数变为不可见

### 3.11 任务11：在训练页面根据呼吸节奏交替显示“吸气”和“呼气”

### 3.11.1 运行效果

该任务实现的运行效果是这样的：单击主页面中的按钮跳转到训练页面，训练页面中根据主页面选择的呼吸节奏交替显示“吸气”和“呼气”。当本次呼吸训练结束时，显示“已完成”。运行效果如图3-58、图3-59和图3-60所示。



图3-58 显示“吸气”的训练页面



图3-59 显示“呼气”的训练页面



图3-60 显示“已完成”的训练页面

### 3.11.2 实现思路

在训练页面的生命周期事件函数onShow()中调用setInterval()创建一个定时器，并在调用时指定定时器要执行的动作以及时间间隔。

### 3.11.3 代码详解

打开training.html文件。

在“再坚持的秒数”的上方添加一个text组件，以交替显示“吸气”和“呼气”。将class属性的值设置为“txt1”，以通过training.css中名为txt1的类选择器设置文本框的样式。通过动态数据绑定的方式指定文本框中显示的文本，并将占位符的名称指定为breath。

为了命名上的统一，将“再坚持的秒数”对应的类选择器名称修改为txt2。

上述讲解如代码清单3-33所示。

#### 代码清单3-33 training.html

```
<div class="container">
  <text class="txt1">
    {{breath}}
  </text>
  <text class="txt2" show="{{isShow}}">
    再坚持 {{seconds}} 秒
  </text>
  <input type="button" value="单击重新开始" class="btn"
onclick="clickAction" />
</div>
```

打开training.css文件。

将title类选择器的名称修改为txt2。

在txt2类选择器的上方添加一个名为txt1的类选择器，以设置“吸气”和“呼气”所对应的文本框的样式。将font-size字体大小的值设置为

38px。将文本的text-align对齐方式设置为center，以居中对齐文本框中的文本。将width的值设置为最大值454px，并将height的值设置为40px。为了与下面的文本框保持一定的间距，将margin-bottom样式的值设置为10px。

上述讲解如代码清单3-34所示。

#### 代码清单3-34 training.css

```
.container {
    flex-direction: column;
    justify-content: center;
    align-items: center;
    width: 454px;
    height: 454px;
}
.txt1 {
    font-size: 38px;
    text-align: center;
    width: 454px;
    height: 40px;
    margin-bottom: 10px;
}
.txt2 {
    font-size: 30px;
    text-align: center;
    width: 400px;
    height: 40px;
}
.....
```

打开training.js文件。

在data中给breath占位符指定一个初始值“吸气”。

在主页面中可以选择的呼吸节奏有3个选项：较慢、舒缓、较快。到底何为较慢，何为舒缓，何为较快呢？我们要将其定量，比如，每6秒吸气或呼气一次称之为较慢；每4秒吸气或呼气一次称之为舒缓；每



2秒吸气或呼气一次称之为较快。因此，需要对主页面中选择的呼吸节奏进行转换。

定义一个全局变量picker2seconds，用于保存转换后得到的秒数。在训练页面的生命周期事件函数onInit()中，对右边选择器的picker2value值进行转换，转换规则为：如果picker2value的值为“较慢”，就转换为6然后赋值给picker2seconds；如果picker2value的值为“舒缓”，就转换为4然后赋值给picker2seconds；如果picker2value的值为“较快”，就转换为2然后赋值给picker2seconds。

为了能够交替显示“吸气”和“呼气”，可以在训练页面正在显示时，也就是在生命周期事件函数onShow()中再调用setInterval()创建一个timer2定时器，在调用时指定定时器要执行的动作以及时间间隔。创建一个全局变量timer2，将其初始值设置为null。将setInterval()的第2个实参指定为picker2seconds\*1000，以指定时间间隔，单位是毫秒。因为第1个实参指定定时器要执行的动作，所以我们可以自定义一个名为run2的函数，然后将第1个实参指定为this.run2。

对于主页面中选择的呼吸时长，为了能够在训练页面中判断是否已经呼吸结束，可声明一个全局变量counter作为计数器，将其初始值设置为0。

在run2函数的函数体中，首先让counter自增1，然后判断counter是否为吸气和呼气的总次数，也就是左边的选择器转换后的picker1seconds值除以右边的选择器转换后的picker2seconds值。如果计数器达到了吸气和呼气的总次数，那么本次呼吸训练结束，清除timer2定时器并将timer2置为null。此外，将breath占位符设置为“已完成”。如果计数器还没有达到吸气和呼气的总次数，也就是本次呼吸训练还没有结束，那就切换显示的文本。如果breath占位符的值为“吸气”，那就

将其修改为“呼气”；如果breath占位符的值为“呼气”，那就将其修改为“吸气”。

当单击训练页面中的按钮跳转到主页面时，在clickAction()函数中清除timer2定时器并将timer2置为null。

为了方便测试，将左边的选择器转换后的值都修改得小一点儿，否则等待的时间太长。当左边选择器的值为“1”时，将其转换为12；当左边选择器的值为“2”时，将其转换为24；当左边选择器的值为“3”时，将其转换为36。

上述讲解如代码清单3-35所示。

#### 代码清单3-35 training.js

```
.....

var picker1seconds = null;
var picker2seconds = null;

var timer1 = null;
var timer2 = null;

var counter = 0;

export default {
  data: {
    seconds: 0,
    isShow: true,
    breath: "吸气"
  },
  clickAction() {
    clearInterval(timer1);
    timer1 = null;

    clearInterval(timer2);
    timer2 = null;

    router.replace({
      uri: 'pages/index/index'
    });
  },
}
```

```

run1() {
    this.seconds--;
    if(this.seconds == 0) {
        clearInterval(timer1);
        timer1 = null;

        this.isShow = false;
    }
},
run2() {
    counter++;
    if(counter == picker1seconds / picker2seconds) {
        clearInterval(timer2);
        timer2 = null;
        this.breath = "已完成";
    } else {
        if(this.breath == "吸气") {
            this.breath = "呼气";
        } else if(this.breath == "呼气") {
            this.breath = "吸气";
        }
    }
},
onInit() {
    console.log("训练页面的onInit()正在被调用");

    console.log("接收到的左边选择器的值：" + this.data1);
    console.log("接收到的右边选择器的值：" + this.data2);

    picker1value = this.data1;
    picker2value = this.data2;

    if(picker1value == "1") {
        picker1seconds = 12;
    } else if(picker1value == "2") {
        picker1seconds = 24;
    } else if(picker1value == "3") {
        picker1seconds = 36;
    }

    if(picker2value == "较慢") {
        picker2seconds = 6;
    } else if(picker2value == "舒缓") {
        picker2seconds = 4;
    } else if(picker2value == "较快") {
        picker2seconds = 2;
    }
}

```

```
    this.seconds = picker1seconds;
  },
  onReady() {
    console.log("训练页面的onReady()正在被调用");
  },
  onShow() {
    console.log("训练页面的onShow()正在被调用");

    timer1 = setInterval(this.run1, 1000);
    timer2 = setInterval(this.run2, picker2seconds * 1000);
  },
  onDestroy() {
    console.log("训练页面的onDestroy()正在被调用");
  }
}
```

保存所有代码后打开Previewer，单击主页面中的按钮跳转到训练页面，训练页面中根据主页面选择的呼吸节奏交替显示“吸气”和“呼气”。当本次呼吸训练结束时，显示“已完成”。运行效果如图3-61、图3-62和图3-63所示。



图3-61 显示“吸气”的训练页面



图3-62 显示“呼气”的训练页面



图3-63 显示“已完成”的训练页面

## 3.12 任务12：每次吸气或呼气时都实时显示进度百分比

### 3.12.1 运行效果

该任务实现的运行效果是这样的：单击主页面中的按钮跳转到训练页面。在训练页面中每次吸气或呼气时都在一个小括号中实时显示进度百分比。每次吸气或呼气都会显示100次进度。当本次呼吸训练结束时，显示“（100%）”。运行效果如图3-64和图3-65所示。



图3-64 实时显示吸气或呼气的进度百分比



图3-65 呼吸训练结束时显示“（100%）”

### 3.12.2 实现思路

在训练页面的生命周期事件函数onShow()中调用setInterval()创建一个定时器，并在调用时指定定时器要执行的动作以及时间间隔。

### 3.12.3 代码详解

打开training.html文件。

对于交替显示的“吸气”和“呼气”，在其后面添加一个小括号。小括号中以动态数据绑定的方式指定进度百分比。百分比的数值来自名为percent的占位符。

上述讲解如代码清单3-36所示。

## 代码清单3-36 training.html

```
<div class="container">
  <text class="txt1">
    {{breath}}({{percent}}%)
  </text>
  <text class="txt2" show="{{isShow}}">
    再坚持 {{seconds}} 秒
  </text>
  <input type="button" value="单击重新开始" class="btn"
onclick="clickAction" />
</div>
```

打开training.js文件。

在data中给percent占位符指定一个初始值“0”。

为了能够实时显示进度百分比，可以在训练页面正在显示时，也就是在生命周期事件函数onShow()中调用setInterval()创建一个定时器timer3，在调用时指定定时器要执行的动作以及时间间隔。

创建一个全局变量timer3，将其初始值设置为null。

每次吸气或呼气的时长是picker2seconds秒，在这段时间里要显示100次进度。因此，将setInterval()的第2个实参指定为picker2seconds/100\*1000，以指定时间间隔，单位是毫秒。因为第1个实参指定定时器要执行的动作，所以我们可以自定义一个名为run3的函数，然后将第1个实参指定为this.run3。

在run3函数的函数体中，首先将this.percent转换为整数，然后将其加1后再转换为字符串。如果加1后的进度百分比小于10，那么在其对应的字符串前面补个0；如果加1后的进度百分比为100，则将其对应的字符串重置为“0”；如果本次呼吸训练结束，也就是如果定时器timer2为null，那么清除timer3定时器并将timer3置为null，此外，将data中的percent占位符设置为“100”。



当单击训练页面中的按钮跳转到主页面时，在clickAction()函数中清除定时器timer3并将timer3置为null。

上述讲解如代码清单3-37所示。

### 代码清单3-37 training.js

```
.....

var timer1 = null;
var timer2 = null;
var timer3 = null;

var counter = 0;

export default {
  data: {
    seconds: 0,
    isShow: true,
    breath: "吸气",
    percent: "0"
  },
  clickAction() {
    clearInterval(timer1);
    timer1 = null;

    clearInterval(timer2);
    timer2 = null;

    clearInterval(timer3);
    timer3 = null;

    router.replace({
      uri: 'pages/index/index'
    });
  },
  run1() {
    this.seconds--;
    if(this.seconds == 0) {
      clearInterval(timer1);
      timer1 = null;

      this.isShow = false;
    }
  },
}
```

```

run2() {
    counter++;
    if(counter == picker1seconds / picker2seconds) {
        clearInterval(timer2);
        timer2 = null;
        this.breath = "已完成";
    } else {
        if(this.breath == "吸气") {
            this.breath = "呼气";
        } else if(this.breath == "呼气") {
            this.breath = "吸气";
        }
    }
},
run3() {
    this.percent = (parseInt(this.percent) + 1).toString();
    if(parseInt(this.percent) < 10) {
        this.percent = "0" + this.percent;
    }
    if(parseInt(this.percent) == 100) {
        this.percent = "0";
    }
    if(timer2 == null) {
        clearInterval(timer3);
        timer3 = null;
        this.percent = "100";
    }
},
.....
onShow() {
    console.log("训练页面的onShow()正在被调用");

    timer1 = setInterval(this.run1, 1000);
    timer2 = setInterval(this.run2, picker2seconds * 1000);
    timer3 = setInterval(this.run3, picker2seconds / 100 * 1000);
},
onDestroy() {
    console.log("训练页面的onDestroy()正在被调用");
}
}

```

---

保存所有代码后打开Previewer，单击主页面中的按钮跳转到训练页面。在训练页面中每次吸气或呼气时都在一个小括号中实时显示了

进度百分比。每次吸气或呼气都会显示100次进度。当本次呼吸训练结束时，显示“（100%）”。运行效果如图3-66和图3-67所示。



图3-66 实时显示吸气或呼气的进度百分比



图3-67 呼吸训练结束时显示“（100%）”

## 3.13 任务13：每次吸气或呼气时logo都顺时针转动一周

### 3.13.1 运行效果

该任务实现的运行效果是这样的：单击主页面中的按钮跳转到训练页面。训练页面的上方显示logo。每次吸气或呼气时logo都会顺时针转动一周。本次呼吸训练结束后，logo图片停止转动。运行效果如图3-68所示。



图3-68 logo顺时针转动的训练页面

### 3.13.2 实现思路

通过style属性中的animation-duration样式指定logo图片转动一次的时间。通过style属性中的animation-iteration-count样式指定logo图片转动的周数。

### 3.13.3 代码详解

打开training.html文件。

添加一个组件image，以显示logo的图片。将class属性的值设置为“img”，以通过training.css中名为img的类选择器设置image组件的样式。将src属性的值设置为“/common/hm.png”，以指定logo图片在项目中的位置。

在一次呼吸训练中，logo转动一次的时间和转动的次数是动态变化的，这取决于主页面上两个选择器的值。因此，添加一个style属性，并在style中通过动态数据绑定的方式指定两个样式。第一个样式是animation-duration，它可以用来表示logo转动一次的时间，将其占位符的名称指定为duration；第二个样式是animation-iteration-count，它可以用来表示logo转动的周数，将其占位符的名称指定为count。

上述讲解如代码清单3-38所示。

#### 代码清单3-38 training.html

```
<div class="container">
  <image class="img" src="/common/hm.png" style="animation-
duration:{{duration}};animation-
iteration-count:{{count}};" />
  <text class="txt1">
    {{breath}}({{percent}}%)
  </text>
  <text class="txt2" show="{{isShow}}">
    再坚持 {{seconds}} 秒
  </text>
```

```
        <input type="button" value="单击重新开始" class="btn"
onclick="clickAction" />
</div>
```

---

打开training.css文件。

添加一个名为img的类选择器，以指定logo图片的样式。因为logo图片的宽度和高度都是208px，所以将width和height的值都设置为208px。为了让logo图片和下面的文本框有一定的间距，将margin-bottom的值设置为10px。将animation-name设置为aniname，以指定动画效果的名称。

定义一个名为aniname的动画效果，以表示logo图片顺时针转动了360度。动画效果以@keyframes开头，其后面是动画效果的名称aniname。在花括号中分别指定起始状态from和终止状态to。因为是让图片进行转动，所以from和to中都是transform: rotate();。因为是让图片顺时针转动一圈，所以两个rotate()中的参数分别是0deg和360deg。

上述讲解如代码清单3-39所示。

#### 代码清单3-39 training.css

---

```
.container {
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 454px;
  height: 454px;
}
.img {
  width: 208px;
  height: 208px;
  margin-bottom: 10px;
  animation-name: aniname;
}
@keyframes aniname {
  from {
    transform: rotate(0deg);
  }
}
```

```
to {  
  transform: rotate(360deg);  
}  
}  
..
```

---

打开training.js文件。

在data中将duration和count占位符都初始化为""。

在训练页面的生命周期事件函数onInit()中，对duration和count占位符再次进行初始化。logo顺时针转动一周的时间，就是一次吸气或呼气的时长picker2seconds。但是，对于training.html中的animation-duration样式，要指定其单位“s”，因此要将picker2seconds+“s”赋值给data中的duration。在一次呼吸训练中，logo转动的次数就是吸气和呼气的总次数，即picker1seconds/picker2seconds。因此，调用toString()将其转换为字符串之后，再赋值给data中的count。

上述讲解如代码清单3-40所示。

#### 代码清单3-40 training.js

---

```
.....  
  
export default {  
  data: {  
    seconds: 0,  
    isShow: true,  
    breath: "吸气",  
    percent: "0",  
    duration: "",  
    count: ""  
  },  
  .....  
  onInit() {  
    .....  
  
    this.seconds = picker1seconds;  
  
    this.duration = picker2seconds + "s";
```

```
        this.count = (picker1seconds / picker2seconds).toString();  
    },  
    .....  
}
```

---

保存所有代码后打开Previewer，单击主页面中的按钮跳转到训练页面。训练页面的上方显示出了logo。每次吸气或呼气logo都会顺时针转动一周。本次呼吸训练结束后，logo停止转动。运行效果如图3-69所示。



图3-69 logo顺时针转动的训练页面

## 3.14 任务14：添加倒计时页面并实现由主页面向其跳转

### 3.14.1 运行效果



该任务实现的运行效果是这样的：单击主页面中的按钮后跳转到倒计时页面。页面中从上往下依次显示“请保持静止”“3秒后跟随训练指引”“进行吸气和呼气”。运行效果如图3-70所示。



图3-70 倒计时页面

### 3.14.2 实现思路

在项目中新建一个倒计时页面。当单击主页面中的按钮时，目标页面的uri指定为倒计时页面的uri。

### 3.14.3 代码详解

在项目的pages子目录上单击右键，在弹出的菜单中选中New，然后在弹出的子菜单中单击JS Page，以新建一个JS页面。运行效果如图3-71所示。

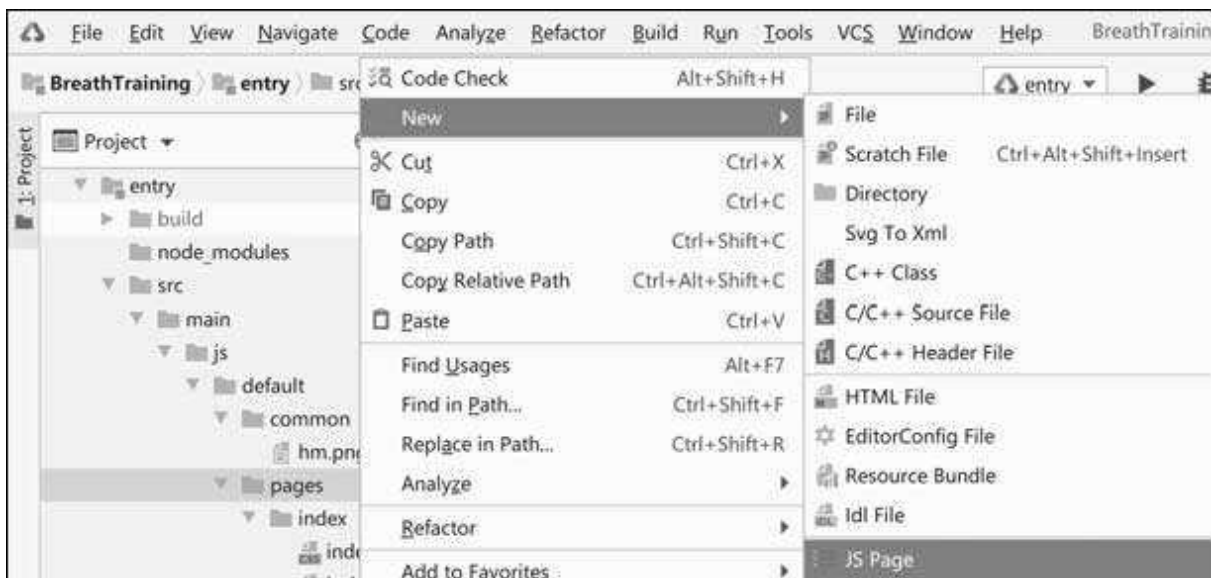


图3-71 新建一个JS页面

在打开文件的窗口中，将JS页面的名称设置为countdown，然后单击Finish按钮，如图3-72所示。

这样，在pages目录下就自动创建了一个名为countdown的子目录。该子目录中自动创建了3个文件：countdown.html、countdown.css和countdown.js，如图3-73所示。这3个文件共同组成了倒计时页面。

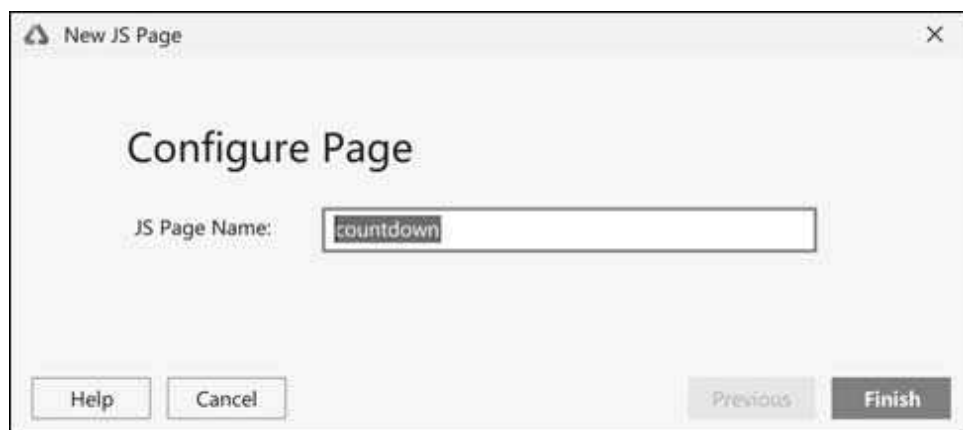


图3-72 配置JS页面的名称

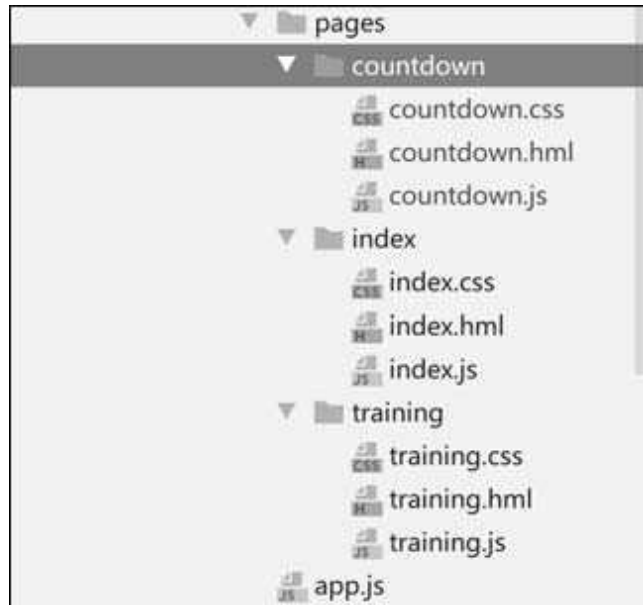


图3-73 自动创建的countdown子目录

打开countdown.html文件。

将text组件的class属性值修改为“txt”，并将显示的文本修改为“请保持静止”。

再添加两个text组件，将class属性的值都设置为“txt”，并将显示的文本分别设置为“3秒后跟随训练指引”“进行吸气和呼气”。

上述讲解如代码清单3-41所示。

#### 代码清单3-41 countdown.html

```
<div class="container">
  <text class="txt">
    请保持静止
  </text>
  <text class="txt">
    3 秒后跟随训练指引
  </text>
  <text class="txt">
    进行吸气和呼气
  </text>
</div>
```

打开countdown.css文件。

在container中添加一个flex-direction样式，将它的值设置为column，以竖向排列div容器内的所有组件。这样，因为无须再使用弹性布局的显示方式，所以就可以删掉样式display了。left和top这两个样式用于定位div容器在页面坐标系中的位置，其默认值都是0px，因此可以将left和top这两个样式都删掉。

将title类选择器的名称修改为txt。将font-size字体大小的值修改为38px。将width的值修改为最大值454px，将height的值修改为50px。为了让每个文本框都与其上面的组件保持一定的间距，添加一个margin-top样式，将它的值设置为10px。

上述讲解如代码清单3-42所示。

#### 代码清单3-42 countdown.css

```
.container {  
  flex-direction: column;  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  left: 0px;  
  top: 0px;  
  width: 454px;  
  height: 454px;  
}  
.txt {  
  font-size: 38px;  
  text-align: center;  
  width: 454px;  
  height: 50px;  
  margin-top: 10px;  
}
```

打开countdown.js文件。

因为在countdown.html中没有使用title占位符，所以删除title及其动态数据绑定的值'World'。

上述讲解如代码清单3-43所示。

#### 代码清单3-43 countdown.js

```
export default {  
  data: {  
    title: 'World'  
  }  
}
```

打开index.js文件。

在clickAction()函数中将目标页面的uri修改为'pages/countdown/countdown'，以实现由主页面向倒计时页面的跳转。

上述讲解如代码清单3-44所示。

#### 代码清单3-44 index.js

```
.....  
  
export default {  
  data: {  
    picker1range: ["1", "2", "3"],  
    picker2range: ["较慢", "舒缓", "较快"]  
  },  
  clickAction() {  
    router.replace({  
      uri: 'pages/countdown/countdown',  
      params: {"data1": picker1value, "data2": picker2value}  
    });  
  },  
  .....  
}
```

保存所有代码后打开Previewer，单击主页面中的按钮后跳转到了倒计时页面。页面中从上往下依次显示“请保持静止”“3秒后跟随训练指引”“进行吸气和呼气”。运行效果如图3-74所示。



图3-74 倒计时页面

## 3.15 任务15：在倒计时页面进行训练指引的3秒倒计时

### 3.15.1 运行效果

该任务实现的运行效果是这样的：单击主页面中的按钮跳转到倒计时页面。在倒计时页面的上方和页面下方的倒数第2行，都会进行训练指引的3秒倒计时。运行效果如图3-75所示。



图3-75 正在3秒倒计时的倒计时页面

### 3.15.2 实现思路

通过3张图片交替显示3秒倒计时。在生命周期事件函数onShow()中调用setInterval()函数创建一个定时器，并在调用时指定定时器要执行的动作以及时间间隔。

### 3.15.3 代码详解

在倒计时页面中，对于上方显示的数字3、2和1，即便使用最大字号的38px，也无法像运行效果中显示得那么大，因此必须使用图片来显示。将数字3、2和1对应的3张图片3.png、2.png和1.png添加到项目的common子目录中。

打开countdown.html文件。

添加一个image组件，以交替显示3秒倒计时的3张图片。将class属性的值设置为“img”，以在countdown.css中根据名为img的类选择器设置image组件的样式。添加一个src属性，通过动态数据绑定的方式指定要显示的图片在项目中的位置。其中，将占位符的名称设置为imgsrc。

在第二个text组件中通过动态数据绑定的方式指定多少秒后跟随训练指引。其中，将占位符的名称设置为seconds。

上述讲解如代码清单3-45所示。

#### 代码清单3-45 countdown.html

```
<div class="container">
  <image class="img" src="{{imgsrc}}"/>
  <text class="txt">
    请保持静止
  </text>
  <text class="txt">
    {{seconds}} 秒后跟随训练指引
  </text>
  <text class="txt">
    进行吸气和呼气
  </text>
</div>
```

打开countdown.css文件。

添加一个名为img的类选择器，以设置image组件的样式。因为用于3秒倒计时的3张图片的宽度和高度都是100px，所以将width和height的值都设置为100px。为了让3张图片都和下面的文本框保持一定的间距，将margin-bottom的值设置为30px。

上述讲解如代码清单3-46所示。

#### 代码清单3-46 countdown.css



```
.container {
  flex-direction: column;
  justify-content: center;
  align-items: center;
  width: 454px;
  height: 454px;
}
.img {
  width: 100px;
  height: 100px;
  margin-bottom: 30px;
}
.txt {
  font-size: 38px;
  text-align: center;
  width: 454px;
  height: 50px;
  margin-top: 10px;
}
```

---

打开countdown.js文件。

在data中将imgsrc和seconds占位符都初始化为""。

声明一个用于计数器的counter全局变量。因为从3开始倒计时，所以将counter初始化为3。

在倒计时页面的生命周期事件函数onInit()中，对imgsrc和seconds占位符再次进行初始化。调用toString()函数将counter转换为字符串，然后拼接为common目录下的图片地址，再将其赋值给data中的imgsrc。调用toString()函数将counter转换为字符串，然后将其赋值给data中的seconds。

为了能够每秒更换一张倒计时图片，可以在倒计时页面正在显示时，也就是在生命周期事件函数onShow()中调用setInterval()函数创建一个定时器timer，在调用时指定定时器要执行的动作以及时间间隔。创建一个全局变量timer，将其初始值设置为null。将setInterval()的第2个实参指定为1000，以指定时间间隔，其单位是毫秒。第1个实参指定定

时器要执行的动作，我们可以自定义一个名为run的函数，然后将setInterval()的第1个实参指定为this.run。

在run函数的函数体中，先将计数器counter自减1，然后对counter进行判断：如果counter没有自减为0，那就更新data中imgsrc和seconds占位符的值，只需要把函数onInit()中的两行代码复制过来就可以了；如果counter自减为0，那就将imgsrc和seconds占位符都设置为""，然后清除timer定时器并将其置为null。

上述讲解如代码清单3-47所示。

#### 代码清单3-47 countdown.js

```
var counter = 3;
var timer = null;

export default {
  data: {
    imgsrc: "",
    seconds: "",
  },
  run() {
    counter = counter - 1;
    if(counter !== 0) {
      this.imgsrc = "/common/" + counter.toString() + ".png";
      this.seconds = counter.toString();
    } else {
      this.imgsrc = "";
      this.seconds = "";

      clearInterval(timer);
      timer = null;
    }
  },
  onInit() {
    this.imgsrc = "/common/" + counter.toString() + ".png";
    this.seconds = counter.toString();
  },
  onShow() {
    timer = setInterval(this.run, 1000);
  }
}
```

---

保存所有代码后打开Previewer，单击主页面中的按钮跳转到了倒计时页面。在倒计时页面的上方和下方的倒数第2行，都会进行训练指引的3秒倒计时。运行效果如图3-76所示。



图3-76 正在3秒倒计时的倒计时页面

## 3.16 任务16：3秒倒计时结束后跳转到训练页面并传递主页面的数据

### 3.16.1 运行效果

该任务实现的运行效果是这样的：单击主页面中的按钮跳转到倒计时页面。3秒倒计时结束后，跳转到训练页面并将主页面的数据传递给训练页面，从而使得训练页面中的所有功能都是我们添加倒计时页面之前所实现的功能。

### 3.16.2 实现思路

在倒计时页面中，将主页面中传递过来的两个选择器的值作为value存放在一个字典中，并且通过params传递给训练页面。在训练页面中，通过key从字典中获取两个选择器的值。

### 3.16.3 代码详解

打开countdown.js文件。

声明两个全局变量pv1和pv2，将其都初始化为null。

在倒计时页面的生命周期事件函数onInit()中，通过this.data1和this.data2获取主页面中传递过来的两个选择器的值，分别将其赋值给pv1和pv2变量。

在函数run()的函数体中，当3秒倒计时结束时，调用router.replace()跳转到训练页面。从 '@system.router' 中导入router，然后通过uri指定目标页面的地址为 'pages/training/training'，并且通过params指定要传递的两个选择器的值。将这两个值都作为value存放在一个字典中，其对应的key分别是key1和key2。

上述讲解如代码清单3-48所示。

#### 代码清单3-48 countdown.js

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

var counter = 3;
var timer = null;

var pv1 = null;
var pv2 = null;
```

```

export default {
  data: {
    imgsrc: "",
    seconds: "",
  },
  run() {
    counter = counter - 1;
    if(counter != 0) {
      this.imgsrc = "/common/" + counter.toString() + ".png";
      this.seconds = counter.toString();
    } else {
      this.imgsrc = "";
      this.seconds = "";

      clearInterval(timer);
      timer = null;

      router.replace({
        uri: 'pages/training/training',
        params: {"key1": pv1, "key2": pv2}
      });
    }
  },
  onInit() {
    pv1 = this.data1;
    pv2 = this.data2;

    this.imgsrc = "/common/" + counter.toString() + ".png";
    this.seconds = counter.toString();
  },
  onShow() {
    timer = setInterval(this.run, 1000);
  }
}

```

---

打开training.js文件。

在onInit()函数中通过this.key1和this.key2获取从倒计时页面传递过来的两个选择器的值，将其分别赋值给picker1value和picker2value变量。

上述讲解如代码清单3-49所示。

**代码清单3-49** training.js

---

```
.....
onInit() {
    console.log("训练页面的onInit()正在被调用");

    console.log("接收到的左边选择器的值：" + this.data1);
    console.log("接收到的右边选择器的值：" + this.data2);

    picker1value = this.key1;
    picker2value = this.key2;

    if(picker1value == "1") {
        picker1seconds = 12;
    } else if(picker1value == "2") {
        picker1seconds = 24;
    } else if(picker1value == "3") {
        picker1seconds = 36;
    }
}
.....
```

---

保存所有代码后打开Previewer，单击主页面中的按钮跳转到倒计时页面。3秒倒计时结束后，跳转到训练页面并将主页面的数据传递给训练页面，从而使得训练页面中的所有功能都是我们添加倒计时页面之前所实现的功能。

## 3.17 任务17：呼吸训练结束后右滑查看训练报告

### 3.17.1 运行效果

该任务实现的运行效果是这样的：呼吸训练结束后，在训练页面中用黄色字体显示文本“右滑查看训练报告”。运行效果如图3-77所示。

当在训练页面中向右滑动时，跳转到第1个训练报告页面。运行效果如图3-78所示。



图3-77 右滑查看训练报告



图3-78 第1个训练报告页面

当在第1个训练报告页面中向左滑动时，跳转到主页面。

### 3.17.2 实现思路

在多个连续的text组件中使用if-elif-else结构，以便从中选择一个text组件进行显示。在页面的最外层div组件中添加onswipe属性，从而在页面触发滑动事件时自动调用指定的自定义函数。

### 3.17.3 代码详解

呼吸训练结束后，在训练页面中不会再显示再坚持的秒数。但是，它在页面中所占用的空间并没有被释放。为了能够释放掉它占用的空间，从而在该空间处显示文本“右滑查看训练报告”，可以在多个连续的text组件中使用if-elif-else结构，以便从中选择一个text组件进行显示。所有不被显示的text组件都不会占用页面空间。

打开training.html文件。

对于再坚持的秒数对应的text组件，将其show属性修改为if。

在该组件的下方添加一个text组件。将class属性的值设置为txt3，并添加一个else属性。将显示的文本设置为“右滑查看训练报告”。

上述讲解如代码清单3-50所示。

#### 代码清单3-50 training.html

```
<div class="container">
    .....
    <text class="txt2" if="{{isShow}}">
        再坚持 {{seconds}} 秒
    </text>
    <text class="txt3" else>
        右滑查看训练报告
    </text>
    <input type="button" value="单击重新开始" class="btn"
onclick="clickAction" />
</div>
```



打开training.css文件。

将txt2类选择器复制一份并重命名为txt3，以设置文本“右滑查看训练报告”所对应的text组件的样式。在txt3类选择器中添加一个color样式，并将其值设置为#ffa500，以设置文本“右滑查看训练报告”的颜色。

上述讲解如代码清单3-51所示。

#### 代码清单3-51 training.css

```
.....  
.txt2 {  
  font-size: 30px;  
  text-align: center;  
  width: 400px;  
  height: 40px;  
}  
.txt3 {  
  font-size: 30px;  
  text-align: center;  
  width: 400px;  
  height: 40px;  
  color: #ffa500;  
}  
.btn {  
  width: 300px;  
  height: 50px;  
  font-size: 38px;  
  background-color: #000000;  
  border-color: #000000;  
  margin-top: 40px;  
}
```

保存所有代码后打开Previewer，呼吸训练结束后，在训练页面中用黄色字体显示出了文本“右滑查看训练报告”。运行效果如图3-79所示。



图3-79 右滑查看训练报告

在项目的pages子目录上单击右键，在弹出的菜单中选中New，然后在弹出的子菜单中单击JS Page，以新建一个名为report1的JS页面。该页面将被作为第1个训练报告页面。

打开report1.html文件。

将text组件中显示的文本修改为“第1个训练报告页面”。

上述讲解如代码清单3-52所示。

#### 代码清单3-52 report1.html

```
<div class="container">
  <text class="title">
    第1个训练报告页面
  </text>
</div>
```

打开report1.js文件。

因为在report1.html中没有使用title占位符，所以在report1.js中删除title及其动态数据绑定的值'World'。

上述讲解如代码清单3-53所示。

#### 代码清单3-53 report1.js

```
export default {  
  data: {  
    title: 'World'  
  }  
}
```

打开report1.css文件。

将title类选择器中width的值修改为最大值454px，以让text组件中的文本“第1个训练报告页面”能够在一行内显示。

上述讲解如代码清单3-54所示。

#### 代码清单3-54 report1.css

```
.....  
.title {  
  font-size: 30px;  
  text-align: center;  
  width: 454px;  
  height: 100px;  
}
```

打开training.js文件。

添加一个名为toReport1Page的自定义函数，并定义一个名为e的形参。在函数体中通过e.direction的值判断滑动的方向。如果e.direction等于字符串“right”，那就跳转到第1个训练报告页面。

上述讲解如代码清单3-55所示。

#### 代码清单3-55 training.js

```
.....
onDestroy() {
  console.log("训练页面的onDestroy()正在被调用");
},
toReport1Page(e) {
  if (e.direction == 'right') {
    router.replace({
      uri: 'pages/report1/report1'
    });
  }
}
}
```

---

打开training.html文件。

在最外层的div组件中将onswipe属性的值设置为自定义的toReport1Page函数。这样，当用户在训练页面中用手指滑动时，就会触发页面的onswipe事件，从而自动调用自定义的toReport1Page函数。如果用户在训练页面中用手指向右滑动了，就会跳转到第1个训练报告页面。

上述讲解如代码清单3-56所示。

#### 代码清单3-56 training.html

---

```
<div class="container" onswipe="toReport1Page">
  .....
</div>
```

---

打开report1.js文件。

添加一个名为toIndexPage的自定义函数，并定义一个名为e的形参。在函数体中通过e.direction的值判断滑动的方向。如果e.direction等于字符串“left”，那就跳转到主页面。

从 '@system.router' 中导入router。

上述讲解如代码清单3-57所示。

### 代码清单3-57 report1.js

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

export default {
  data: {

  },
  toIndexPage(e) {
    if (e.direction == 'left') {
      router.replace({
        uri: 'pages/index/index'
      });
    }
  }
}
```

打开report1.html文件。

在最外层的div组件中将onswipe属性的值设置为自定义的函数名toIndexPage。这样，当用户在第1个训练报告页面中用手指滑动时，就会触发页面的onswipe事件，从而自动调用自定义的toIndexPage函数。如果用户向左滑动了页面，就会跳转到主页面。

上述讲解如代码清单3-58所示。

### 代码清单3-58 report1.html

```
<div class="container" onswipe="toIndexPage">
  <text class="title">
    第1个训练报告页面
  </text>
</div>
```

保存所有代码后打开Previewer，呼吸训练结束后，当在训练页面中向右滑动时，跳转到了第1个训练报告页面。运行效果如图3-80所示。



图3-80 第1个训练报告页面

当在第1个训练报告页面中向左滑动时，跳转回主页面。

## 3.18 任务18：将第1个训练报告页面的标题修改为压力占比

### 3.18.1 运行效果

该任务实现的运行效果是这样的：在App启动后首先显示的是主页面，但是主页面一闪而过，接着显示的是第1个训练报告页面，其标题为“压力占比”。运行效果如图3-81所示。



图3-81 第1个训练报告页面

当在第1个训练报告页面中向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是第1个训练报告页面。

### 3.18.2 实现思路

在主页面的生命周期事件函数onInit()中跳转到某个页面，从而间接地将该页面设置为App在启动后显示的第1个页面。

### 3.18.3 代码详解

为了方便大家学习，在接下来的几个训练报告页面中使用的数据都是随机生成的测试数据。数据的来源和真实性并不太重要，重要的是让大家学会如何对数据进行分析 and 可视化展示。

打开index.js文件。

为了方便测试，我们在主页面的生命周期事件函数onInit()中跳转到第1个训练报告页面。这样，在App启动后我们看到的第1个页面就是第1个训练报告页面。

上述讲解如代码清单3-59所示。

#### 代码清单3-59 index.js

```
.....
onInit() {
    console.log("主页面的onInit()正在被调用");

    router.replace({
        uri: 'pages/report1/report1'
    });
},
.....
```

打开training.js文件。

在onInit()函数中，对于左边的选择器转换后得到的秒数，之前因为测试的方便我们将其改小了，现在我们将其改回实际的转换值：将1分转换为60秒；将2分转换为120秒；将3分转换为180秒。

上述讲解如代码清单3-60所示。

#### 代码清单3-60 training.js

```
.....
onInit() {
    .....

    picker1value = this.key1;
    picker2value = this.key2;

    if(picker1value == "1") {
        picker1seconds = 60;
    } else if(picker1value == "2") {
        picker1seconds = 120;
```



```
    } else if(picker1value == "3") {  
        picker1seconds = 180;  
    }  
  
    .....  
},  
.....
```

---

打开report1.html文件。

将text组件中显示的页面标题修改为“压力占比”。在所有组件中，text组件是唯一不是必须要设置width和height的组件。如果不设置text组件的宽度和高度，就需要在其外层嵌套一个div组件，以便对其样式进行设置。在text组件的外层嵌套一个div组件，并将属性class的值设置为“title-container”。

上述讲解如代码清单3-61所示。

#### 代码清单3-61 report1.html

---

```
<div class="container" onswipe="toIndexPage">  
    <div class="title-container">  
        <text class="title">  
            压力占比  
        </text>  
    </div>  
</div>
```

---

打开report1.css文件。

在title类选择器中删除width、height和text-align样式，将font-size的值修改为38px，并将margin-top的值设置为40px。

添加一个title-container类选择器，以设置text组件的div外层组件的样式。将width和height的值分别设置为300px和130px。将justify-content和align-items都设置为center，从而让容器div内的组件在水平方向和竖直方向都居中对齐。

在container类选择器中删除left、top和display样式，并将flex-direction的值设置为column。将justify-content的值修改为flex-start，从而让容器div内的组件在竖直方向与容器的上边缘对齐。

上述讲解如代码清单3-62所示。

#### 代码清单3-62 report1.css

```
.container {
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
    left: 0px;
    top: 0px;
    width: 454px;
    height: 454px;
}
.title-container {
    width: 300px;
    height: 130px;
    justify-content: center;
    align-items: center;
}
.title {
    font-size: 38px;
    margin-top: 40px;
    text-align: center;
    width: 484px;
    height: 100px;
}
```

保存所有代码后打开Previewer，在App启动后首先显示的是主页面，但是主页面一闪而过，接着显示的是第1个训练报告页面，其标题为“压力占比”。运行效果如图3-82所示。



图3-82 第1个训练报告页面

当在第1个训练报告页面中向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是第1个训练报告页面。

注意：flex-direction样式用于指定容器内所有组件的排列方向，可选值有两个：row和column，分别表示水平方向排列和竖直方向排列。当flex-direction的值设置为row时，水平方向为主轴，竖直方向为副轴；当flex-direction的值设置为column时，竖直方向为主轴，水平方向为副轴。

justify-content样式用于指定容器内所有组件在主轴上的对齐方式，可选值有5个：

flex-start、flex-end、center、space-between和space-around。

align-items样式用于指定容器内所有组件在副轴上的对齐方式，可选值有3个：flex-start、flex-end和center。

组合使用以上3个样式，可以指定容器内所有组件的布局。接下来我们通过多组示例来演示以上3个样式的组合用法。

新建一个智能手表的项目。

打开index.html文件。

在最外层的div组件中嵌套4个div组件，将class属性的值分别设置为subcontainer1、subcontainer2、subcontainer3、subcontainer4。

上述讲解如代码清单3-63所示。

### 代码清单3-63 index.html

```
<div class="container">
  <div class="subcontainer1">
  </div>

  <div class="subcontainer2">
  </div>

  <div class="subcontainer3">
  </div>

  <div class="subcontainer4">
  </div>
</div>
```

打开index.css文件。

添加4个类选择器，以设置4个div内嵌组件的样式。

将第1个div内嵌组件的width和height都设置为40px，并将其背景色设置为蓝色。

将第2个div内嵌组件的width和height都设置为60px，并将其背景色设置为绿色。

将第3个div内嵌组件的width和height都设置为80px，并将其背景色设置为红色。

将第4个div内嵌组件的width和height都设置为100px，并将其背景色设置为黄色。

为了设置4个div内嵌组件的布局，在container类选择器中将flex-direction的值设置为row，以指定水平方向为主轴，从而指定容器内所

有组件的排列方向为水平方向。将justify-content的值设置为flex-start，以指定容器内所有组件在主轴上的对齐方式。将align-items的值设置为center，以指定容器内所有组件在副轴上的对齐方式。

上述讲解如代码清单3-64所示。

#### 代码清单3-64 index.css

```
.container {
    flex-direction: row;
    justify-content: flex-start;
    align-items: center;
    width: 454px;
    height: 454px;
}
.subcontainer1 {
    width: 40px;
    height: 40px;
    background-color: blue;
}
.subcontainer2 {
    width: 60px;
    height: 60px;
    background-color: green;
}
.subcontainer3 {
    width: 80px;
    height: 80px;
    background-color: red;
}
.subcontainer4 {
    width: 100px;
    height: 100px;
    background-color: yellow;
}
```

保存所有代码后打开Previewer，4个div内嵌组件的排列方向为水平方向，在主轴（水平方向）上的对齐方式为左对齐，在副轴（竖直方向）上的对齐方式为居中对齐。运行效果如图3-83所示。

将index.css中主轴上的对齐方式修改为flex-end。

上述讲解如代码清单3-65所示。

#### 代码清单3-65 index.css

```
.container {  
    flex-direction: row;  
    justify-content: flex-end;  
    align-items: center;  
    width: 454px;  
    height: 454px;  
}
```

保存所有代码后打开Previewer，4个div内嵌组件在主轴上的对齐方式为右对齐。运行效果如图3-84所示。



图3-83 主轴上的对齐方式为flex-start



图3-84 主轴上的对齐方式为flex-end

将index.css中主轴上的对齐方式修改为center。

上述讲解如代码清单3-66所示。

#### 代码清单3-66 index.css

```
.container {  
    flex-direction: row;  
    justify-content: center;  
    align-items: center;  
    width: 454px;  
    height: 454px;  
}
```

保存所有代码后打开Previewer，4个div内嵌组件在主轴上的对齐方式为居中对齐。运行效果如图3-85所示。

将index.css中主轴上的对齐方式修改为space-between。

上述讲解如代码清单3-67所示。

#### 代码清单3-67 index.css

---

```
.container {  
    flex-direction: row;  
    justify-content: space-between;  
    align-items: center;  
    width: 454px;  
    height: 454px;  
}
```

---

保存所有代码后打开Previewer，4个div内嵌组件在主轴上的对齐方式为两端对齐。运行效果如图3-86所示。

将index.css中主轴上的对齐方式修改为space-around。

上述讲解如代码清单3-68所示。

#### 代码清单3-68 index.css

---

```
.container {  
    flex-direction: row;  
    justify-content: space-around;  
    align-items: center;  
    width: 454px;  
    height: 454px;  
}
```

---





图3-85 主轴上的对齐方式为center



图3-86 主轴上的对齐方式为space-between

保存所有代码后打开Previewer，4个div内嵌组件在主轴上的对齐方式为分散对齐。运行效果如图3-87所示。

将index.css中副轴上的对齐方式修改为flex-start。  
上述讲解如代码清单3-69所示。

#### 代码清单3-69 index.css

```
.container {  
    flex-direction: row;  
    justify-content: space-around;  
    align-items: flex-start;  
    width: 454px;  
    height: 454px;  
}
```

保存所有代码后打开Previewer，4个div内嵌组件在副轴上的对齐方式为上对齐。运行效果如图3-88所示。



图3-87 主轴上的对齐方式为space-around



图3-88 副轴上的对齐方式为flex-start

将index.css中副轴上的对齐方式修改为flex-end。

上述讲解如代码清单3-70所示。

#### 代码清单3-70 index.css

```
.container {  
    flex-direction: row;  
    justify-content: space-around;  
    align-items: flex-end;  
    width: 454px;  
    height: 454px;  
}
```

保存所有代码后打开Previewer，4个div内嵌组件在副轴上的对齐方式为下对齐。运行效果如图3-89所示。

将index.css中flex-direction的值设置为column，以指定竖直方向为主轴，从而指定容器内所有组件的排列方向为竖直方向。

上述讲解如代码清单3-71所示。

## 代码清单3-71 index.css

```
.container {  
    flex-direction: column;  
    justify-content: space-around;  
    align-items: flex-end;  
    width: 454px;  
    height: 454px;  
}
```

保存所有代码后打开Previewer，4个div内嵌组件在主轴（竖直方向）上的对齐方式为分散对齐，并且在副轴（水平方向）上的对齐方式为右对齐。运行效果如图3-90所示。



图3-89 副轴上的对齐方式为flex-end



图3-90 竖直方向为主轴

通过以上多组示例的演示，相信大家已经掌握了flex-direction、justify-content和align-items这3个样式的组合用法，从而可以轻松地指定容器内所有组件的布局了。

## 3.19 任务19：在压力占比页面的标题下方显示压力分类的列表

### 3.19.1 运行效果

该任务实现的运行效果是这样的：在第1个训练报告页面（后面都称之为压力占比页面）的标题下方显示压力分类的列表。运行效果如图3-91所示。



图3-91 显示压力分类列表的压力占比页面

### 3.19.2 实现思路

联合使用list和list-item组件来展示一个列表中的多个列表项。在list组件中使用for属性并通过动态数据绑定的方式指定要迭代的数组。在list-item组件中使用tem并通过动态数据绑定的方式指定迭代过程中数组中的元素。

### 3.19.3 代码详解

打开report1.html文件。

添加一个list组件以在页面中显示一个列表，将其class属性的值设置为“state-wrapper”。

list和list-item组件要组合使用，其中，list-item用于定义列表中的列表项。

在list组件中添加4个list-item组件，以定义4个列表项，将它们的class属性的值都设置为“state-item”。在每一个list-item组件中都添加一个text组件，将它们的class属性的值都设置为“state”。4个text组件显示的文本分别为“焦虑80-99”“紧张60-79”“正常30-59”“放松1-29”。其中，在“1-29”的前面额外添加了两个空格，以让4个列表项保持右对齐。

上述讲解如代码清单3-72所示。

### 代码清单3-72 report1.html

```
<div class="container" onswipe="toIndexPage">
  <div class="title-container">
    <text class="title">
      压力占比
    </text>
  </div>
  <list class="state-wrapper">
    <list-item class="state-item">
      <text class="state">
        焦虑 80-99
      </text>
    </list-item>
    <list-item class="state-item">
      <text class="state">
        紧张 60-79
      </text>
    </list-item>
    <list-item class="state-item">
      <text class="state">
        正常 30-59
      </text>
    </list-item>
    <list-item class="state-item">
      <text class="state">
        放松 1-29
      </text>
    </list-item>
  </list>
</div>
```

---

打开report1.css文件。

添加一个state-wrapper类选择器以定义list组件的样式，将width和height的值分别设置为320px和220px。

添加一个state-item类选择器以定义4个list-item组件的样式，将width和height的值分别设置为320px和55px。

添加一个state类选择器以定义列表中4个text组件的样式，将font-size的值设置为24px，并将color的值设置为gray。之前我们设置与颜色相关的样式值时，使用的都是十六进制的形式，其实还可以使用颜色对应的英语单词来设置样式值。

上述讲解如代码清单3-73所示。

#### 代码清单3-73 report1.css

---

```
.....
.title {
    font-size: 38px;
    margin-top: 40px;
}
.state-wrapper {
    width: 320px;
    height: 220px;
}
.state-item {
    width: 320px;
    height: 55px;
}
.state {
    font-size: 24px;
    color: gray;
}
```

---

保存所有代码后打开Previewer，在压力占比页面的标题下方显示出了压力分类的列表。运行效果如图3-92所示。





图3-92 显示压力分类列表的压力占比页面

为了把数据和显示进行分离，我们把report1.html中的4个列表项数据提取到report1.js中，然后在report1.html中通过动态数据绑定的方式进行指定。

打开report1.js文件。

在data中添加一个名为states的数组，数组中的元素分别为'焦虑80-99'、'紧张60-79'、'正常30-59'、'放松1-29'。

上述讲解如代码清单3-74所示。

#### 代码清单3-74 report1.js

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

export default {
  data: {
    states: [
      '焦虑 80-99',
      ...
    ]
  }
}
```

```

        '紧张 60-79',
        '正常 30-59',
        '放松 1-29',
    ]
},
toIndexPage(e) {
    if (e.direction == 'left') {
        router.replace({
            uri: 'pages/index/index'
        });
    }
}
}
}

```

打开report1.html文件。

通过动态数据绑定的方式指定4个列表项的数据，它们分别为“{{states[0]}}”“{{states[1]}}”“{{states[2]}}”和“{{states[3]}}”。

上述讲解如代码清单3-75所示。

### 代码清单3-75 report1.html

```

<div class="container" onswipe="toIndexPage">
    <div class="title-container">
        <text class="title">
            压力占比
        </text>
    </div>
    <list class="state-wrapper">
        <list-item class="state-item">
            <text class="state">
                {{states[0]}}
            </text>
        </list-item>
        <list-item class="state-item">
            <text class="state">
                {{states[1]}}
            </text>
        </list-item>
        <list-item class="state-item">
            <text class="state">
                {{states[2]}}
            </text>
        </list-item>
    </list>
</div>

```

```
        </list-item>
        <list-item class="state-item">
            <text class="state">
                {{states[3]}}
            </text>
        </list-item>
    </list>
</div>
```

---

保存所有代码后打开Previewer，运行结果没有发生任何变化。

打开report1.html文件。

仔细观察和对比一下4个list-item组件，它们的区别仅仅在于两个花括号中占位符的索引是不一样的，其余部分全是一样的。因此，要显示这4个列表项，还有更简单的类似for循环的写法。可以把4个list-item组件合并为一个list-item组件，并将for属性的值设置为“{{states}}”。这样，每一个列表项数据都可以用“{{ \$item }}”来表示。系统会自动对“{{states}}”进行迭代，将迭代过程中的每一个数据自动赋值给“{{ \$item }}”。

上述讲解如代码清单3-76所示。

#### 代码清单3-76 report1.html

---

```
<div class="container" onswipe="toIndexPage">
    <div class="title-container">
        <text class="title">
            压力占比
        </text>
    </div>
    <list class="state-wrapper">
        <list-item class="state-item" for="{{states}}">
            <text class="state">
                {{tem}}
            </text>
        </list-item>
    </list>
</div>
```

---

保存所有代码后打开Previewer，运行结果没有发生任何变化。但是，代码较之前简洁了很多。

## 3.20 任务20：在压力分类的右边显示对应的压力占比

### 3.20.1 运行效果

该任务实现的运行效果是这样的：在压力占比页面中，在压力分类的右边显示对应的压力占比。运行效果如图3-93所示。



图3-93 显示压力占比的压力占比页面

## 3.20.2 实现思路

在页面的生命周期事件函数onInit()中，随机生成若干个指定范围内的整数，将其作为所有压力状态的数据。根据随机生成的整数统计每种压力状态所占的百分比，并通过动态数据绑定的方式将其显示在列表中。

## 3.20.3 代码详解

打开report1.js文件。

为了方便地表示每个列表项的所有相关数据，我们将其分别存储在一个字典中。将states数组中的4个列表项数据分别作为value存储在4个字典中。其中，对应的key都为state。

上述讲解如代码清单3-77所示。

### 代码清单3-77 report1.js

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

export default {
  data: {
    states: [
      {
        state: '焦虑 80-99'
      },
      {
        state: '紧张 60-79'
      },
      {
        state: '正常 30-59'
      },
      {
        state: '放松 1-29'
      }
    ]
  }
}
```

```

    }
  ]
},
toIndexPage(e) {
  if (e.direction == 'left') {
    router.replace({
      uri: 'pages/index/index'
    });
  }
}
}
}

```

打开report1.html文件。

对于列表项中text组件显示的文本，在两个花括号中就要通过\$item.state进行表示。其中，\$item.后面的state表示report1.js中字典的key。

上述讲解如代码清单3-78所示。

#### 代码清单3-78 report1.html

```

import router from '@system.router'

<div class="container" onswipe="toIndexPage">
  <div class="title-container">
    <text class="title">
      压力占比
    </text>
  </div>
  <list class="state-wrapper">
    <list-item class="state-item" for="{{states}}">
      <text class="state">
        {{tem.state}}
      </text>
    </list-item>
  </list>
</div>

```

保存所有代码后打开Previewer，与上一个任务相比运行结果没有发生任何变化。

打开report1.js文件。

为了能够在压力分类的右边显示对应的压力占比，在states数组中的每个字典中都添加一个key为percent的元素，并将对应的value都初始化为0。

上述讲解如代码清单3-79所示。

### 代码清单3-79 report1.js

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

export default {
  data: {
    states: [
      {
        state: '焦虑 80-99',
        percent: 0
      },
      {
        state: '紧张 60-79',
        percent: 0
      },
      {
        state: '正常 30-59',
        percent: 0
      },
      {
        state: '放松 1-29',
        percent: 0
      }
    ]
  },
  toIndexPage(e) {
    if (e.direction == 'left') {
      router.replace({
        uri: 'pages/index/index'
      });
    }
  }
}
```

打开report1.html文件。

在text组件的下方添加一个text组件，以显示每个列表项的压力占比。将class属性的值设置为“state”。通过动态数据绑定的方式将显示的文本指定为“{{tem.percent}}%”。

在两个text组件的外部嵌套一个div组件，并将其class属性的值设置为“state-percent”。

上述讲解如代码清单3-80所示。

### 代码清单3-80 report1.html

```
<div class="container" onswipe="toIndexPage">
  <div class="title-container">
    <text class="title">
      压力占比
    </text>
  </div>
  <list class="state-wrapper">
    <list-item class="state-item" for="{{states}}">
      <div class="state-percent">
        <text class="state">
          {{tem.state}}
        </text>
        <text class="state">
          {{tem.percent}}%
        </text>
      </div>
    </list-item>
  </list>
</div>
```

打开report1.css文件。

添加一个名为state-percent的类选择器，以设置report1.html中list-item组件的div内部组件的样式。将width和height的值分别设置为320px和25px。将justify-content的值设置为space-between，从而在主轴（水平方向）上将所有组件的对齐方式指定为两端对齐。

上述讲解如代码清单3-81所示。



### 代码清单3-81 report1.css

```
.....
.state {
    font-size: 24px;
    color: gray;
}
.state-percent {
    width: 320px;
    height: 25px;
    justify-content: space-between;
}
```

打开report1.js文件。

定义一个名为getRandomInt的函数，其两个形参分别为min和max。该函数用于随机生成一个介于min和max之间（包含min和max）的整数。在函数体中，Math.random()用于生成一个介于0和1之间（包含0但不包含1）的随机数；Math.floor(x)用于返回小于等于x的最大整数。

上述讲解如代码清单3-82所示。

### 代码清单3-82 report1.js

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

export default {
  data: {
    .....
  }
  getRandomInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1) ) + min;
  },
  toIndexPage(e) {
    if (e.direction == 'left') {
      router.replace({
        uri: 'pages/index/index'
      });
    }
  }
}
```

```
}  
}
```

在页面的生命周期事件函数onInit()里，首先创建一个空数组并赋值给变量stateData，然后通过for循环执行20次迭代。在每一次迭代中，调用getRandomInt()自定义函数随机生成一个介于1和99之间的整数，并调用push()函数将随机生成的整数添加到stateData数组中。

定义一个名为countStatePercent的函数，其形参为stateData，该函数用于计算每种压力状态所占的百分比。

在onInit()函数的最后，调用countStatePercent()自定义函数，并将stateData作为实参传递给形参stateData。

上述讲解如代码清单3-83所示。

### 代码清单3-83 report1.js

```
import router from '@system.router'  
  
export default {  
  data: {  
    .....  
  },  
  onInit() {  
    let stateData = [];  
    for (let i = 0; i < 20; i++) {  
      stateData.push(this.getRandomInt(1, 99));  
    }  
    this.countStatePercent(stateData);  
  },  
  getRandomInt(min, max) {  
    return Math.floor(Math.random() * (max - min + 1) ) + min;  
  },  
  countStatePercent(stateData) {  
  
  },  
  toIndexPage(e) {  
    if (e.direction == 'left') {  
      router.replace({  
        uri: 'pages/index/index'  
      })  
    }  
  }  
}
```

```
    });  
  }  
}  
}
```

---

在countStatePercent()自定义函数的函数体中，声明4个变量：counter0、counter1、counter2和counter3，将其都初始化为0。这4个变量都作为计数器，分别用于统计每种压力状态的数据个数。通过for循环对stateData数组中的所有压力状态数据进行遍历，并在遍历的过程中判断每个压力状态数据的范围，使用相应的计数器进行个数的统计。

遍历结束后，根据4个计数器的值分别计算每种压力状态所占的百分比，并将其作为value分别赋值给data中states数组里的字典。其中，对应的key都是percent。

上述讲解如代码清单3-84所示。

#### 代码清单3-84 report1.js

---

```
import router from '@system.router'  
  
export default {  
  data: {  
    .....  
    getRandomInt(min, max) {  
      return Math.floor(Math.random() * (max - min + 1) ) + min;  
    },  
    countStatePercent(stateData) {  
      let counter0 = 0;  
      let counter1 = 0;  
      let counter2 = 0;  
      let counter3 = 0;  
  
      for (let index = 0; index < stateData.length; index++) {  
        let currentData = stateData[index];  
  
        if (currentData >= 80 && currentData <= 99) {  
          counter0++;  
        } else if (currentData >= 60 && currentData <= 79) {  
          counter1++;  
        }  
      }  
    }  
  }  
}
```

```

    } else if (currentData >= 30 && currentData <= 59) {
        counter2++;
    } else if (currentData >= 1 && currentData <= 29) {
        counter3++;
    }
}

this.states[0].percent = counter0 / stateData.length * 100;
this.states[1].percent = counter1 / stateData.length * 100;
this.states[2].percent = counter2 / stateData.length * 100;
this.states[3].percent = counter3 / stateData.length * 100;
},
toIndexPage(e) {
    if (e.direction == 'left') {
        router.replace({
            uri: 'pages/index/index'
        });
    }
}
}
}
}

```

保存所有代码后打开Previewer，在压力占比页面中，在压力分类的右边显示出了对应的压力占比。运行效果如图3-94所示。



图3-94 显示压力占比的压力占比页面

## 3.21 任务21：在每个列表项的下方显示压力占比的进度条

### 3.21.1 运行效果

该任务实现的运行效果是这样的：在压力占比页面中，每个列表项的下方都显示压力占比的进度条。

运行效果如图3-95所示。



图3-95 显示进度条的压力占比页面

### 3.21.2 实现思路

通过progress组件显示每个列表项中的进度条。通过style属性和动态数据绑定的方式指定进度条的颜色。

### 3.21.3 代码详解

打开report1.js文件。

为了使用不同的颜色对列表中的进度条进行区分，在states数组的每个字典中都添加一个key-value对。其中，所有的key都设置为color，所有的value分别为'#ffa500'、'#ffff00'、'#00ffff'、和'#0000ff'。

上述讲解如代码清单3-85所示。

#### 代码清单3-85 report1.js

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

export default {
  data: {
    states: [
      {
        state: '焦虑 80-99',
        percent: 0,
        color: '#ffa500'
      },
      {
        state: '紧张 60-79',
        percent: 0,
        color: '#ffff00'
      },
      {
        state: '正常 30-59',
        percent: 0,
        color: '#00ffff'
      },
      {
        state: '放松 1-29',
        percent: 0,
        color: '#0000ff'
      }
    ]
  }
}
```

```
    }  
  ]  
},  
.....  
}
```

---

打开report1.html文件。

在list-item组件中添加一个progress组件，以在每个列表项中都显示一个进度条。将class属性的值设置为“progress-bar”。通过动态数据绑定的方式将percent属性的值指定为“{{ \$item.percent }}”，以使用列表项的压力占比来表示进度条的进度。通过动态数据绑定的方式指定style属性的值。其中，将color样式的值指定为{{ \$item.color }}。

上述讲解如代码清单3-86所示。

#### 代码清单3-86 report1.html

---

```
<div class="container" onswipe="toIndexPage">  
  <div class="title-container">  
    <text class="title">  
      压力占比  
    </text>  
  </div>  
  <list class="state-wrapper">  
    <list-item class="state-item" for="{{states}}">  
      <div class="state-percent">  
        <text class="state">  
          {{tem.state}}  
        </text>  
        <text class="state">  
          {{tem.percent}}%  
        </text>  
      </div>  
      <progress class="progress-bar" percent="{{  
{{ $item.percent }}" style="color: {{ $item.color }}" />  
    </list-item>  
  </list>  
</div>
```

---

打开report1.css文件。

在state-item类选择器中将flex-direction属性的值设置为column，以在竖直方向上排列进度条和其他组件。

添加一个名为progress-bar的类选择器，以定义progress组件的样式。将width和height的值分别设置为320px和5px。将margin-top的值设置为5px，以让进度条和其上面的文本保持一定的间距。

上述讲解如代码清单3-87所示。

#### 代码清单3-87 report1.css

```
.....
state-item {
    width: 320px;
    height: 55px;
    flex-direction: column;
}
.state {
    font-size: 24px;
    color: gray;
}
.state-percent {
    width: 320px;
    height: 25px;
    justify-content: space-between;
}
.progress-bar {
    width: 320px;
    height: 5px;
    margin-top: 5px;
}
```

保存所有代码后打开Previewer，在压力占比页面中，每个列表项的下方都显示出了压力占比的进度条。运行效果如图3-96所示。





图3-96 显示进度条的压力占比页面

## 3.22 任务22：添加第2个训练报告页面并响应滑动事件

### 3.22.1 运行效果

该任务实现的运行效果是这样的：在App启动后首先显示的是主页面，但是主页面一闪而过，接着显示的是第2个训练报告页面。运行效果如图3-97所示。



图3-97 第2个训练报告页面

当在第2个训练报告页面或压力占比页面中向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是第2个训练报告页面。当在第2个训练报告页面中向下滑动时，显示的是压力占比页面。当在压力占比页面中向上滑动时，显示的是第2个训练报告页面。

### 3.22.2 实现思路

在主页面的生命周期事件函数onInit()中跳转到某个页面，从而间接地将该页面设置为App在启动后显示的第1个页面。在页面的div最外层组件中添加onswipe属性，从而在页面触发滑动事件时自动调用指定的自定义函数。

### 3.22.3 代码详解

在项目的pages子目录上单击右键，在弹出的菜单中选中New，然后在弹出的子菜单中单击JS Page，以新建一个名为report2的JS页面。该页面将被作为第2个训练报告页面。

打开report2.html文件。

将text组件中显示的文本修改为“第2个训练报告页面”。

上述讲解如代码清单3-88所示。

#### 代码清单3-88 report2.html

```
<div class="container">
  <text class="title">
    第2个训练报告页面
  </text>
</div>
```

打开report2.js文件。

因为在report2.html中没有使用title占位符，所以在report2.js中删除title及其动态数据绑定的值'World'。

上述讲解如代码清单3-89所示。

#### 代码清单3-89 report2.js

```
export default {
  data: {
    title: 'World'
  }
}
```

打开report2.css文件。

将title类选择器中width的值修改为454px，以让text组件中的文本“第2个训练报告页面”能够在一行内显示。

上述讲解如代码清单3-90所示。

#### 代码清单3-90 report2.css

```
.....  
.title {  
    font-size: 30px;  
    text-align: center;  
    width: 454px;  
    height: 100px;  
}
```

打开report2.js文件。

添加一个名为toNextPage的自定义函数，并定义一个名为e的形参。在函数体中通过e.direction的值判断滑动的方向。如果e.direction等于字符串“left”，那就跳转到主页面；如果e.direction等于字符串“down”，那就跳转到压力占比页面。

从 '@system.router' 中导入router。

上述讲解如代码清单3-91所示。

#### 代码清单3-91 report2.js

```
import router from '@system.router'  
  
export default {  
    data: {  
  
    } ,  
    toNextPage(e) {  
        switch(e.direction) {  
            case 'left':  
                router.replace({  
                    uri: 'pages/index/index'  
                });  
                break;  
            case 'down':  
                router.replace({
```

```
        uri: 'pages/report1/report1'
    });
}
}
}
```

打开report2.html文件。

在最外层的div组件中将onswipe属性的值设置为toNextPage自定义函数。这样，当用户在第2个训练报告页面中用手指滑动时，就会触发页面的onswipe事件从而自动调用自定义函数toNextPage。

上述讲解如代码清单3-92所示。

#### 代码清单3-92 report2.html

```
<div class="container" onswipe="toNextPage">
    <text class="title">
        第2个训练报告页面
    </text>
</div>
```

打开report1.js文件。

将自定义函数toIndexPage重命名为toNextPage。在函数体中通过e.direction的值判断滑动的方向。如果e.direction等于字符串“left”，那就跳转到主页面；如果e.direction等于字符串“up”，那就跳转到第2个训练报告页面。为了让大家熟悉JavaScript的更多语法知识，这里将if语句修改为switch语句。

上述讲解如代码清单3-93所示。

#### 代码清单3-93 report1.js

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

export default {
```

```
data: {  
  
},  
toNextPage(e) {  
  switch(e.direction) {  
    case 'left':  
      router.replace({  
        uri: 'pages/index/index'  
      });  
      break;  
    case 'up':  
      router.replace({  
        uri: 'pages/report2/report2'  
      });  
    }  
  }  
}
```

---

打开report1.html文件。

将onswipe属性的值修改为重命名后的函数名toNextPage。

上述讲解如代码清单3-94所示。

#### 代码清单3-94 report1.html

---

```
<div class="container" onswipe="toNextPage">  
  <div class="title-container">  
    <text class="title">  
      压力占比  
    </text>  
  </div>  
  .....  
</div>
```

---

打开index.js文件。

为了方便测试，我们在主页面的生命周期事件函数onInit()中跳转到第2个训练报告页面。这样，在App启动后我们看到的第1个页面就是第2个训练报告页面。

上述讲解如代码清单3-95所示。

## 代码清单3-95 index.js

```
.....  
onInit() {  
  console.log("主页面的onInit()正在被调用");  
  
  router.replace({  
    uri: 'pages/report2/report2'  
  });  
},  
.....
```

保存所有代码后打开Previewer，首先显示的是主页面，但是主页面一闪而过，接着显示的是第2个训练报告页面。运行效果如图3-98所示。



图3-98 第2个训练报告页面

当在第2个训练报告页面或压力占比页面中向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是第2个训练报告页

面。当在第2个训练报告页面中向下滑动时，显示的是压力占比页面。当在压力占比页面中向上滑动时，显示的是第2个训练报告页面。

## 3.23 任务23：在第2个训练报告页面中显示除心率曲线之外的所有内容

### 3.23.1 运行效果

该任务实现的运行效果是这样的：在第2个训练报告页面（后面都称之为心率曲线页面）显示页面标题、心率最大值及其图标、心率最小值及其图标、心率在每分钟内的平均次数。运行效果如图3-99所示。



图3-99 心率曲线页面



### 3.23.2 实现思路

在页面的生命周期事件函数onInit()中，随机生成若干个指定范围内的整数，以作为所有的心率数据。根据随机生成的整数统计所有心率的`最大值`、`最小值`和`平均值`，并通过`动态数据绑定`的方式将其显示在页面中。

### 3.23.3 代码详解

打开report2.html文件。

将text组件中显示的页面标题修改为“心率曲线”，并在其外层嵌套一个div组件，以便对其样式进行设置。将该div组件的class属性的值设置为“title-container”。

在页面标题的下方添加一个div组件以显示心率曲线图，并将class属性的值设置为“chart”。

在心率曲线图的下方添加一个list组件，以显示心率的`最大值`、`最小值`及其图标，并将class属性的值设置为“list”。

在list组件的内部嵌套一个list-item组件以显示列表中的每个列表项，并将class属性的值设置为“list-item”。通过`动态数据绑定`的方式指定for属性的值为“`{{maxmin}}`”，从而对report2.js中data里面的maxmin进行迭代。

每个列表项都由一张图片和一个文本组成，因此在list-item组件中添加一个image组件和一个text组件。

在image组件中将属性class的值设置为“icon”，并通过`动态数据绑定`的方式将src属性的值设置为“`/common/{{tem.iconName}}.png`”。这

样，report2.js中data里面的maxmin可以是一个字典的数组，数组中的每个字典都包含一个key为iconName的元素。

在text组件中将class属性的值设置为“maxmin”，并通过动态数据绑定的方式将显示的文本设置为“{{tem.mValue}}”。这样，对于report2.js中data里面的maxmin数组，其中的每个字典都包含一个key为mValue的元素。

在列表的下方添加一个div组件以显示心率平均值，并将class属性的值设置为“averagecontainer”。

在div组件中嵌套定义3个text组件，其class属性的值分别为“average”“average-number”和“average”，其显示的文本分别为“平均”“{{average}}”“次/分”。

上述讲解如代码清单3-96所示。

#### 代码清单3-96 report2.html

```
<div class="container" onswipe="toNextPage">
  <div class="title-container">
    <text class="title">
      心率曲线
    </text>
  </div>
  <div class="chart">

  </div>
  <list class="list">
    <list-item class="list-item" for="{{maxmin}}">
      <image class="icon"
src="/common/{{tem.iconName}}.png"/>
      <text class="maxmin">
        {{tem.mValue}}
      </text>
    </list-item>
  </list>
  <div class="average-container">
    <text class="average">
      平均
```

```
        </text>
        <text class="average-number">
            {{average}}
        </text>
        <text class="average">
            次/分
        </text>
    </div>
</div>
```

---

打开report2.css文件。

在container类选择器中删除display、left和top样式。将flex-direction的值设置为column，以在竖直方向上排列容器内的所有组件。将justify-content的值修改为flex-start，以让容器内的所有组件在主轴上向上对齐。

在title类选择器中删除text-align、width和height样式。将font-size的值修改为38px。将margin-top的值设置为40px，以让页面标题与页面的上边缘保持一定的间距。

添加一个名为title-container的类选择器，以设置页面标题的样式。将justify-content和align-items都设置为center，以让容器内的组件在水平方向和竖直方向都居中对齐。将width和height的值分别设置为300px和130px。

添加一个名为chart的类选择器，以设置心率曲线图的样式。将width和height的值分别设置为400px和180px。

添加一个名为list的类选择器，以设置列表的样式。将flex-direction的值设置为row，以在水平方向上排列所有列表项。将width和height的值分别设置为200px和45px。

添加一个名为list-item的类选择器，以设置列表项的样式。将justify-content和align-items都设置为center，以让列表项内的组件在水平方向和竖直方向都居中对齐。将width和height的值分别设置为100px和45px。

添加一个名为icon的类选择器，以设置心率的最大值图标和最小值图标的样式。将width和height的值分别设置为32px和32px。

添加一个名为maxmin的类选择器，以设置心率的最大值文本和最小值文本的样式。将font-size的值设置为24px。将letter-spacing的值设置为0px，以让数字之间的间距更紧凑。

添加一个名为average-container的类选择器，以设置心率平均值的相关文本的样式。将justify-content的值设置为space-between，以让容器内的组件在水平方向上两端对齐。将align-items的值设置为center，以让容器内的组件在竖直方向上居中对齐。将width和height的值分别设置为220px和55px。

添加一个名为average-number的类选择器，以设置心率平均值的样式。将font-size的值设置为38px。将letter-spacing的值设置为0px，以让数字之间的间距更紧凑。

添加一个名为average的类选择器，以设置心率平均值的两边文本的样式。将font-size的值设置为24px。将color的值设置为gray，以将文本显示为灰色。

上述讲解如代码清单3-97所示。

#### 代码清单3-97 report2.css

```
.container {
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
    left: 0px;
    top: 0px;
    width: 454px;
    height: 454px;
}
.title-container {
    justify-content: center;
```

```

        align-items: center;
        width: 300px;
        height: 130px;
    }
    .title {
        margin-top: 40px;
        font-size: 38px;
        text-align: center;
        width: 454px;
        height: 100px;
    }
    .chart {
        width: 400px;
        height: 180px;
    }
    .list {
        flex-direction: row;
        width: 200px;
        height: 45px;
    }
    .list-item {
        justify-content: center;
        align-items: center;
        width: 100px;
        height: 45px;
    }
    .icon {
        width: 32px;
        height: 32px;
    }
    .maxmin {
        font-size: 24px;
        letter-spacing: 0px;
    }
    .average-container {
        justify-content: space-between;
        align-items: center;
        width: 220px;
        height: 55px;
    }
    .average {
        font-size: 24px;
        color: gray;
    }
    .average-number {
        font-size: 38px;
        letter-spacing: 0px;
    }

```

---

把心率最大值图标max.png和心率最小值图标min.png添加到common目录中。

打开report2.js文件。

在data中将maxmin占位符初始化为一个字典数组。该数组中包含两个字典，分别表示心率最大值和心率最小值的相关信息。每个字典中都有两个元素，对应的key都是iconName和mValue，分别表示心率最大的图标名称和心率最值。对于第一个字典，将心率最大的图标名称iconName初始化为“max”，并将心率最大值初始化为0。对于第二个字典，将心率最小值的图标名称iconName初始化为“min”，并将心率最小值初始化为0。

在data中将average占位符初始化为0。

上述讲解如代码清单3-98所示。

### 代码清单3-98 report2.js

---

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

export default {
  data: {
    maxmin: [{
      iconName: 'max',
      mValue: 0
    },
    {
      iconName: 'min',
      mValue: 0
    }
  ],
    average: 0
  },
  toNextPage(e) {
    .....
  }
}
```

---

将report1.js中自定义的名为getRandomInt的函数复制过来，该函数用于随机生成一个介于min和max之间（包含min和max）的整数。

在页面的生命周期事件函数onInit()中，首先创建一个空数组并赋值给变量heartRates，然后通过for循环执行100次迭代。在每一次迭代中，调用getRandomInt()自定义函数随机生成一个介于73和159之间的整数，并调用push()函数将随机生成的整数添加到heartRates数组中。

定义一个名为countMaxMinAverage的函数，其形参为heartRates，该函数用于计算heartRates中所有元素的最大值、最小值和平均值。

在onInit()函数的最后，调用countMaxMinAverage()自定义函数，并将heartRates作为实参传递给形参heartRates。

上述讲解如代码清单3-99所示。

#### 代码清单3-99 report2.js

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

export default {
  data: {
    .....
  },
  onInit() {
    let heartRates = [];
    for (let i = 0; i < 100; i++) {
      heartRates.push(this.getRandomInt(73, 159));
    }
    this.countMaxMinAverage(heartRates);
  },
  getRandomInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1) ) + min;
  },
  countMaxMinAverage(heartRates) {

  },
  toNextPage(e) {
    .....
  }
}
```

---

在自定义函数countMaxMinAverage()的函数体中，分别调用Math.max.apply()和Math.min.apply()计算heartRates数组中的最大值和最小值，然后分别赋值给data中的maxmin[0].mValue和maxmin[1].mValue。通过for循环对heartRates数组中的所有心率数据进行遍历，在遍历的过程中将心率数据累加到变量sum，以计算heartRates数组中所有心率数据的总和。求出总和之后，将其除以所有心率数据的个数就得到了所有心率数据的平均值。调用Math.round()函数返回与心率平均值最接近的整数，并将其赋值给data中的average。

上述讲解如代码清单3-100所示。

#### 代码清单3-100 report2.js

---

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

export default {
  .....
  countMaxMinAverage(heartRates) {
    this.maxmin[0].mValue = Math.max.apply(null, heartRates);
    this.maxmin[1].mValue = Math.min.apply(null, heartRates);

    let sum = 0;
    for (let index = 0; index < heartRates.length; index++) {
      sum += heartRates[index];
    }
    this.average = Math.round(sum / heartRates.length);
  },
  .....
}
```

---

保存所有代码后打开Previewer，在心率曲线页面显示出了页面标题、心率最大值及其图标、心率最小值及其图标、心率在每分钟内的平均次数。运行效果如图3-100所示。





图3-100 心率曲线页面

## 3.24 任务24：在心率曲线页面中显示绘制的心率曲线

### 3.24.1 运行效果

该任务实现的运行效果是这样的：在心率曲线页面中显示绘制的心率曲线。运行效果如图3-101所示。



图3-101 显示心率曲线图的心率曲线页面

### 3.24.2 实现思路

使用chart组件绘制心率曲线图。通过动态数据绑定的方式指定chart组件中options和datasets属性的值，以对图形的参数进行设置。

### 3.24.3 代码详解

打开report2.html文件。

将list组件上方的div组件修改为chart，以绘制一张心率曲线图。在chart组件中，通过动态数据绑定的方式将options和datasets属性的值分别设置为“{{options}}”和“{{datasets}}”。

上述讲解如代码清单3-101所示。

## 代码清单3-101 report2.html

```
<div class="container" onswipe="toNextPage">
  <div class="title-container">
    <text class="title">
      心率曲线
    </text>
  </div>
  <chart class="chart" options="{{options}}" datasets="{{datasets}}" />
  </div>
  <list class="list">
    .....
  </list>
  .....
</div>
```

打开report2.js文件。

在data中将options占位符的值初始化为一个字典，字典中包含两个元素，分别用于设置轴和轴的参数。第一个元素的key是xAxis，对应的value是一个空字典{}，说明不需要对轴的参数进行设置。第二个元素的key是yAxis，对应的value是一个由两个元素组成的字典，分别用于设置轴的最小值和最大值。其中，key分别是min和max，value分别是0和160。

在data中将datasets占位符的值初始化为一个字典的数组。该数组中只包含一个字典，该字典中包含两个元素。第一个元素的key是gradient，对应的value是true，用于表示折线向下填充颜色到轴。第二个元素的key是data，对应的value是一个空数组[]，用于指定心率图中的数据。

在页面的生命周期事件函数onInit()中，在随机生成100个整数之后将所有整数组成的数组赋值给data中的datasets[0].data。

上述讲解如代码清单3-102所示。

## 代码清单3-102 report2.js

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

export default {
  data: {
    .....
    average: 0,
    options: {
      xAxis: {},
      yAxis: {
        min: 0,
        max: 160
      }
    },
    datasets: [{
      gradient: true,
      data: []
    }]
  },
  onInit() {
    let heartRates = [];
    for (let i = 0; i < 100; i++) {
      heartRates.push(this.getRandomInt(73, 159));
    }
    this.datasets[0].data = heartRates;
    this.countMaxMinAverage(heartRates);
  },
  .....
}
```

保存所有代码后打开Previewer，在心率曲线页面中显示出了绘制的心率曲线。运行效果如图3-102所示。



图3-102 显示心率曲线图的心率曲线页面

## 3.25 任务25：添加第3个训练报告页面并响应滑动事件

### 3.25.1 运行效果

该任务实现的运行效果是这样的：在App启动后首先显示的是主页面，但是主页面一闪而过，接着显示的是第3个训练报告页面。运行效果如图3-103所示。

当在第3个训练报告页面中向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是第3个训练报告页面。当在第3个训

练报告页面中向下滑动时，显示的是心率曲线页面。当在心率曲线页面中向上滑动时，显示的是第3个训练报告页面。



图3-103 第3个训练报告页面

### 3.25.2 实现思路

在主页面的生命周期事件函数onInit()中跳转到某个页面，从而间接地将该页面设置为App在启动后显示的第1个页面。在页面的最外层div组件中添加onswipe属性，从而在页面触发滑动事件时自动调用指定的自定义函数。

### 3.25.3 代码详解

在项目的pages子目录上单击右键，在弹出的菜单中选中New，然后在弹出的子菜单中单击JS Page，以新建一个名为report3的JS页面。该页面将被作为第3个训练报告页面。

打开report3.html文件。

将text组件中显示的文本修改为“第3个训练报告页面”。

上述讲解如代码清单3-103所示。

#### 代码清单3-103 report3.html

```
<div class="container">
  <text class="title">
    第3个训练报告页面
  </text>
</div>
```

打开report3.js文件。

因为在report3.html中没有使用title占位符，所以在report3.js中删除title及其动态数据绑定的值'World'。

上述讲解如代码清单3-104所示。

#### 代码清单3-104 report3.js

```
export default {
  data: {
    title: 'World'
  }
}
```

打开report3.css文件。

将title类选择器中width的值修改为454px，以让text组件中的文本“第3个训练报告页面”能够在一行内显示。

上述讲解如代码清单3-105所示。

#### 代码清单3-105 report3.css

```
.....  
.title {  
    font-size: 30px;  
    text-align: center;  
    width: 454px;  
    height: 100px;  
}
```

打开report3.js文件。

添加一个名为toNextPage的自定义函数，并定义一个名为e的形参。在函数体中通过e.direction的值判断滑动的方向。如果e.direction等于字符串“left”，那就跳转到主页面；如果e.direction等于字符串“down”，那就跳转到心率曲线页面。

从 '@system.router' 中导入router。上述讲解如代码清单3-106所示。

#### 代码清单3-106 report3.js

```
import router from '@system.router'  
  
export default {  
    data: {  
  
    } ,  
    toNextPage(e) {  
        switch(e.direction) {  
            case 'left':  
                router.replace({  
                    uri: 'pages/index/index'  
                });  
                break;  
            case 'down':  
                router.replace({  
                    uri: 'pages/report2/report2'  
                });  
        }  
    }  
};
```



```
}  
}  
}
```

---

打开report3.html文件。

在最外层的div组件中将onswipe属性的值设置为自定义的函数名toNextPage。这样，当用户在第3个训练报告页面中用手指滑动时，就会触发页面的onswipe事件从而自动调用自定义的toNextPage函数。

上述讲解如代码清单3-107所示。

#### 代码清单3-107 report3.html

---

```
<div class="container" onswipe="toNextPage">  
  <text class="title">  
    第3个训练报告页面  
  </text>  
</div>
```

---

打开report2.js文件。

在toNextPage()函数中添加一个case分支：如果e.direction等于字符串“up”，那就跳转到第3个训练报告页面。

上述讲解如代码清单3-108所示。

#### 代码清单3-108 report2.js

---

```
import router from '@system.router'  
  
export default {  
  .....  
  toNextPage(e) {  
    switch(e.direction) {  
      case 'left':  
        router.replace({  
          uri: 'pages/index/index'  
        });  
    }  
  }  
};
```

---

```
        break;
    case 'up':
        router.replace({
            uri: 'pages/report3/report3'
        });
        break;
    case 'down':
        router.replace({
            uri: 'pages/report1/report1'
        });
    }
}
}
```

---

打开index.js文件。

为了方便测试，我们在主页面的生命周期事件函数onInit()中跳转到第3个训练报告页面。这样，在App启动后我们看到的第1个页面就是第3个训练报告页面。

上述讲解如代码清单3-109所示。

---

#### 代码清单3-109 index.js

---

```
.....
onInit() {
    console.log("主页面的onInit()正在被调用");

    router.replace({
        uri: 'pages/report3/report3'
    });
},
.....
```

---

保存所有代码后打开Previewer，首先显示的是主页面，但是主页面一闪而过，接着显示的是第3个训练报告页面。运行效果如图3-104所示。



图3-104 第3个训练报告页面

当在第3个训练报告页面中向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是第3个训练报告页面。当在第3个训练报告页面中向下滑动时，显示的是心率曲线页面。当在心率曲线页面中向上滑动时，显示的是第3个训练报告页面。

## 3.26 任务26：在第3个训练报告页面中显示除活动分布图之外的所有内容

### 3.26.1 运行效果

该任务实现的运行效果是这样的：在第3个训练报告页面（后面都称之为今日活动分布页面）显示页面标题、一天内的几个时间点、活

动占比和静止占比的列表。运行效果如图3-105所示。



图3-105 今日活动分布页面

### 3.26.2 实现思路

在页面的生命周期事件函数onInit()中，随机生成若干个0和1，以作为所有的活动分布数据。根据随机生成的整数统计活动所占的比例和静止所占的比例，并通过动态数据绑定的方式将其显示在页面中。

### 3.26.3 代码详解

打开report3.html文件。

将text组件中显示的页面标题修改为“今日活动分布”，并在其外层嵌套一个div组件，以便对其样式进行设置。将该div组件的class属性的

值设置为“title-container”。

在页面标题的下方添加一个div组件，以显示今日活动分布图，并将class属性的值设置为“chart-container”。

在今日活动分布图对应组件的下方添加一个div组件，以显示一天中的几个时间点，并将class属性的值设置为“time-container”。在该div组件的内部嵌套一个text组件，将其class属性的值设置为“time”。通过动态数据绑定的方式指定for属性的值为“{{timeRange}}”，从而对report3.js中data里面的timeRange进行迭代。通过动态数据绑定的方式将text组件中显示的文本指定为“{{tem}}”。

在时间点对应组件的下方添加一个list组件，以显示活动和静止的图标、文本和百分比，并将class属性的值设置为“activities-list”。

在list组件的内部嵌套一个list-item组件，以显示列表中的每个列表项，并将class属性的值设置为“activity”。通过动态数据绑定的方式指定for属性的值为“{{activityData}}”，从而对report3.js中data里面的activityData进行迭代。

每个列表项都由一张图片和两个文本组成，因此在list-item组件中添加一个image组件和两个text组件。

在image组件中将class属性的值设置为“icon”，并通过动态数据绑定的方式将src属性的值设置为“/common/{{tem.iconName}}-circle.png”。这样，report3.js中data里面的activityData可以是一个字典的数组，数组中的每个字典都包含一个key为iconName的元素。

在第一个text组件中将class属性的值设置为“type”，并通过动态数据绑定的方式将显示的文本设置为“{{tem.text}}”。这样，对于report3.js中data里面的数组activityData，其中的每个字典都包含一个key为text的元素。

在第二个text组件中将class属性的值设置为“percent”，并通过动态数据绑定的方式将显示的文本设置为“{{tem.percent}}%”。这样，对于report3.js中data里面的数组activityData，其中的每个字典都包含一个key为percent的元素。

上述讲解如代码清单3-110所示。

#### 代码清单3-110 report3.html

```
<div class="container" onswipe="toNextPage">
  <div class="title-container">
    <text class="title">
      今日活动分布
    </text>
  </div>
  <div class="chart-container">
  </div>
  <div class="time-container">
    <text class="time" for="{{timeRange}}">
      {{tem}}
    </text>
  </div>
  <list class="activities-list">
    <list-item class="activity" for="{{activityData}}">
      <image class="icon" src="/common/{{tem.iconName}}-
circle.png"/>
      <text class="type">
        {{tem.text}}
      </text>
      <text class="percent">
        {{tem.percent}}%
      </text>
    </list-item>
  </list>
</div>
```

打开report3.css文件。

在container类选择器中删除display、left和top样式。将flex-direction的值设置为column，以在竖直方向上排列容器内的所有组件。将

justify-content的值修改为flex-start，以让容器内的所有组件在主轴上向上对齐。

在title类选择器中删除text-align、width和height样式。将font-size的值修改为38px。将margin-top的值设置为40px，以让页面标题与页面的上边缘保持一定的间距。

添加一个名为title-container的类选择器，以设置页面标题的样式。将justify-content和align-items都设置为center，以让容器内的组件在水平方向和竖直方向都居中对齐。将width和height的值分别设置为300px和130px。

添加一个名为chart-container的类选择器，以设置今日活动分布图的样式。将width和height的值分别设置为340px和150px。

添加一个名为time-container的类选择器，以设置时间点的样式。将width和height的值分别设置为340px和25px。将justify-content的值设置为space-between，以让容器内的组件在水平方向都两端对齐。将align-items的值设置为center，以让容器内的组件在竖直方向都居中对齐。

添加一个名为time的类选择器，以设置时间点文本的样式。将font-size的值设置为18px，将color的值设置为gray，并将letter-spacing的值设置为0px。

添加一个名为activities-list的类选择器，以设置活动占比的列表的样式。将width和height的值分别设置为180px和110px。将margin-top的值设置为10px，以让活动占比的列表与其上面的时间点保持一定距离的间隔。

添加一个名为activity的类选择器，以设置活动占比列表项的样式。将width和height的值分别设置为175px和45px。将justify-content的值设置为space-between，以让列表项内的组件在水平方向都两端对齐。

将align-items的值设置为center，以让列表项内的组件在竖直方向都居中对齐。

添加一个名为icon的类选择器，以设置活动图标和静止图标的样式。将width和height的值分别设置为32px和32px。

添加一个名为type的类选择器，以设置活动文本和静止文本的样式。将font-size的值设置为24px。将letter-spacing的值设置为0px，以让数字之间的间距更紧凑。将margin-right的值设置为10px，以让活动文本和静止文本与图标的距离更近一些。

添加一个名为percent的类选择器，以设置活动占比和静止占比的样式。将font-size的值设置为30px，并将letter-spacing的值设置为0px。

上述讲解如代码清单3-111所示。

#### 代码清单3-111 report3.css

```
.container {
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
    left: 0px;
    top: 0px;
    width: 454px;
    height: 454px;
}
.title-container {
    justify-content: center;
    align-items: center;
    width: 300px;
    height: 130px;
}
.title {
    margin-top: 40px;
    font-size: 38px;
    text-align: center;
    width: 454px;
    height: 100px;
}
```



```
.chart-container {
  width: 340px;
  height: 150px;
}
.time-container {
  width: 340px;
  height: 25px;
  justify-content: space-between;
  align-items: center;
}
.time {
  font-size: 18px;
  color: gray;
  letter-spacing: 0px;
}
.activities-list {
  width: 180px;
  height: 110px;
  margin-top: 10px;
}
.activity {
  width: 175px;
  height: 45px;
  justify-content: space-between;
  align-items: center;
}
.icon {
  width: 32px;
  height: 32px;
}
.type {
  font-size: 24px;
  letter-spacing: 0px;
  margin-right: 10px;
}
.percent {
  font-size: 30px;
  letter-spacing: 0px;
}
```

---

把活动图标red-circle.png和静止图标gray-circle.png添加到common目录中。

打开report3.js文件。

在data中将timeRange占位符初始化为一个数组，该数组中的元素分别为“07：00”“12：00”“17：00”“22：00”。

在data中将activityData占位符初始化为一个字典的数组。该数组中包含两个字典，分别表示活动和静止的相关信息。每个字典中都有3个元素，对应的key都是iconName、text和percent，分别表示活动和静止的图标名称、文本和百分比。对于第一个字典，将活动的图标名称iconName初始化为“red”，将文本初始化为“活动”，并将百分比初始化为0。对于第二个字典，将静止的图标名称iconName初始化为“gray”，将文本初始化为“静止”，并将百分比初始化为0。

上述讲解如代码清单3-112所示。

#### 代码清单3-112 report3.js

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

export default {
  data: {
    timeRange: ["07:00", "12:00", "17:00", "22:00"],
    activityData: [
      {
        iconName: "red",
        text: "活动",
        percent: 0
      },
      {
        iconName: "gray",
        text: "静止",
        percent: 0
      }
    ]
  },
  toNextPage(e) {
    .....
  }
}
```

定义一个名为`getRandomZeroOrOne`的函数，该函数用于随机生成一个整数0或1。在函数体中，`Math.random()`用于生成一个介于0和1之间（包含0但不包含1）的随机数；`Math.floor(x)`用于返回小于等于x的最大整数。

在页面的生命周期事件函数`onInit()`里，首先创建一个空数组并赋值给变量`activities`，然后通过`for`循环执行20次迭代。在每一次迭代中，调用自定义函数`getRandomZeroOrOne()`随机生成一个整数0或1，并调用函数`push()`将随机生成的整数添加到数组`activities`中。

定义一个名为`countActivityPercent`的函数，其形参为`activities`。该函数用于计算`activities`中整数1和整数0分别所占的百分比。

在函数`onInit()`的最后，调用自定义函数`countActivityPercent()`，并将`activities`作为实参传递给形参`activities`。

上述讲解如代码清单3-113所示。

### 代码清单3-113 report3.js

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

export default {
  data: {
    .....
  },
  onInit() {
    let activities = [];
    for (let i = 0; i < 20; i++) {
      activities.push(this.getRandomZeroOrOne());
    }

    this.countActivityPercent(activities);
  },
  getRandomZeroOrOne() {
    return Math.floor(Math.random() + 0.5);
  },
  countActivityPercent(activities) {

  },
}
```

```
    toNextPage(e) {  
        .....  
    }  
}
```

---

在自定义函数countActivityPercent()的函数体中，通过for循环对activities数组中的所有活动分布数据进行遍历。在遍历的过程中将数值为1的元素累加到变量count，以计算数组activities中所有数值为1的元素的个数。求出个数之后，将其除以所有活动分布数据的个数就得到了活动所占的百分比，将其乘以100后再调用Math.round()函数返回最接近的整数，并将其赋值给data中的activityData[0].percent。用100减去data中的activityData[0].percent，将计算结果赋值给data中的activityData[1].percent。

上述讲解如代码清单3-114所示。

#### 代码清单3-114 report3.js

---

```
import router from '@system.router'  
  
export default {  
    .....  
    countActivityPercent(activities) {  
        let count = 0;  
        for (let index = 0; index < activities.length; index++) {  
            if (activities[index] == 1) {  
                count++;  
            }  
        }  
  
        this.activityData[0].percent = Math.round(count /  
activities.length * 100);  
        this.activityData[1].percent = 100 -  
this.activityData[0].percent;  
    },  
    .....  
}
```

---

保存所有代码后打开Previewer，在今日活动分布页面中显示出了页面标题、一天内的几个时间点、活动占比和静止占比的列表。运行效果如图3-106所示。



图3-106 今日活动分布页面

## 3.27 任务27：在今日活动分布页面中显示绘制的今日活动分布图

### 3.27.1 运行效果

该任务实现的运行效果是这样的：在今日活动分布页面中显示绘制的今日活动分布图。运行效果如图3-107所示。



图3-107 显示今日活动分布图的今日活动分布页面

### 3.27.2 实现思路

通过将chart组件的type属性设置为“bar”来绘制一张柱状图。通过stack组件来堆叠其中的子组件，从而分别绘制活动柱状图和静止柱状图。

### 3.27.3 代码详解

打开report3.html文件。

将今日活动分布图对应的div组件修改为chart。在chart组件中，通过动态数据绑定的方式将options和datasets属性的值分别设置

为“{{options}}”和“{{datasets}}”。将type属性的值设置为“{{bar}}”，以显示一张柱状图。

上述讲解如代码清单3-115所示。

#### 代码清单3-115 report3.html

```
<div class="container" onswipe="toNextPage">
  <div class="title-container">
    <text class="title">
      今日活动分布
    </text>
  </div>
  <chart class="chart-container" type="bar" options="{{options}}" datasets="{{datasets}}"/>
</div>
<div class="time-container">
  <text class="time" for="{{timeRange}}">
    {{tem}}
  </text>
</div>
.....
</div>
```

打开report3.js文件。

在data中将options占位符的值初始化为一个字典，该字典中包含两个元素，分别用于设置轴和轴的参数。第一个元素的key是xAxis，对应的value是一个字典。该字典中只包含一个元素，对应的key和value分别是axisTick和20，用于设置x轴上的刻度数量。在options对应的字典中，第二个元素的key是yAxis，对应的value是一个由两个元素组成的字典，分别用于设置轴的最大值和刻度数量。其中，两个元素的key分别是max和axisTick，对应的value都是1。

在data中将datasets占位符的值初始化为一个字典的数组，该数组中只包含一个字典。该字典中只包含一个元素，元素的key是data，对应的value是一个空数组[]，用于指定今日活动分布图中的数据。

在页面的生命周期事件函数onInit()中，在随机生成20个整数（0和1）之后将所有整数组成的数组赋值给data中的datasets[0].data。

上述讲解如代码清单3-116所示。

### 代码清单3-116 report3.js

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

export default {
  data: {
    timeRange: ["07:00", "12:00", "17:00", "22:00"],
    activityData: [
      .....
    ],
    options: {
      xAxis: {
        axisTick: 20
      },
      yAxis: {
        max: 1,
        axisTick: 1,
      }
    },
    datasets: [
      {
        data: []
      },
    ]
  },
  onInit() {
    let activities = [];
    for (let i = 0; i < 20; i++) {
      activities.push(this.getRandomZeroOrOne());
    }
    this.datasets[0].data = activities;
    this.countActivityPercent(activities);
  },
  .....
}
```



保存所有代码后打开Previewer，在今日活动分布页面中显示出了活动对应的柱状图。运行效果如图3-108所示。

接下来实现的功能是，在今日活动分布页面中同时显示活动和静止对应的柱状图。运行效果如图3-109所示。



图3-108 显示活动柱状图的今日活动分布页面



图3-109 显示活动和静止柱状图的今日活动分布页面

打开report3.html文件。

在chart组件的外面嵌套一个stack组件，这样其中的子组件会按照顺序依次入栈，从而后一个入栈的子组件会堆叠在前一个入栈的子组件的上面。将stack组件的class属性的值设置为“stack”。

把chart组件复制一份，粘贴在stack组件的内部。对于粘贴之后的副本，将其datasets属性的值修改为“{{datasetsStatic}}”。

上述讲解如代码清单3-117所示。

#### 代码清单3-117 report3.html

```
<div class="container" onswipe="toNextPage">
  <div class="title-container">
    <text class="title">
      今日活动分布
    </text>
  </div>
  <stack class="stack">
    <chart class="chart-container" type="bar" options="
```

```

{{options}}" datasets="{{datasetsStatic}}"/>
    <chart class="chart-container" type="bar" options="
{{options}}" datasets="{{datasets}}"/>
    </stack>
    <div class="time-container">
        <text class="time" for="{{timeRange}}">
            {{tem}}
        </text>
    </div>
    .....
</div>

```

---

打开report3.css文件。

添加一个名为stack的类选择器，以设置report3.html中stack组件的样式。将width和height的值分别设置为340px和150px。

上述讲解如代码清单3-118所示。

#### 代码清单3-118 report3.css

---

```

.....
.title {
    margin-top: 40px;
    font-size: 38px;
}
.stack {
    width: 340px;
    height: 150px;
}
.chart-container {
    width: 340px;
    height: 150px;
}
.....

```

---

打开report3.js文件。

在data中将datasets与其初始值复制一份并粘贴在其上面。对于粘贴之后的副本，将占位符的名称修改为datasetsStatic，并在其对应数组的

第一个字典中添加一个元素，该元素的key和value分别是fillColor和“#696969”，以设置静止柱状图的填充颜色。

在页面的生命周期事件函数onInit()中，创建一个空数组并赋值给变量activitiesStatic。在for循环中，首先将生成的随机整数赋值给变量rand，然后将rand添加到activities数组中，最后对rand进行转换后将其添加到数组activitiesStatic中。其中，转换的规则为Math.abs(rand-1)。这样，如果rand为0，就将其转换为1；如果rand为1，就将其转换为0。在for循环结束后，将activitiesStatic赋值给data中的datasetsStatic[0].data。

上述讲解如代码清单3-119所示。

#### 代码清单3-119 report3.js

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

export default {
  data: {
    .....
    options: {
      .....
    },
    datasetsStatic: [
      {
        fillColor: "#696969",
        data: []
      }
    ],
    datasets: [
      {
        data: []
      },
    ]
  },
  onInit() {
    let activities = [];
    let activitiesStatic = [];
    for (let i = 0; i < 20; i++) {
      let rand = this.getRandomZeroOrOne();
      activities.push(rand);
      activitiesStatic.push(Math.abs(rand - 1));
    }
  }
}
```

```
}  
this.datasets[0].data = activities;  
this.datasetsStatic[0].data = activitiesStatic;  
this.countActivityPercent(activities);  
},  
.....  
}
```

保存所有代码后打开Previewer，在今日活动分布页面中显示出了绘制的今日活动分布图。运行效果如图3-110所示。



图3-110 显示今日活动分布图的今日活动分布页面

## 3.28 任务28：添加第4个训练报告页面并响应滑动事件

### 3.28.1 运行效果

该任务实现的运行效果是这样的：在App启动后首先显示的是主页面，但是主页面一闪而过，接着显示的是第4个训练报告页面。运行效果如图3-111所示。



图3-111 第4个训练报告页面

当在第4个训练报告页面中向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是第4个训练报告页面。当在第4个训练报告页面中向下滑动时，显示的是今日活动分布页面。当在今日活动分布页面中向上滑动时，显示的是第4个训练报告页面。

### 3.28.2 实现思路

在主页面的生命周期事件函数onInit()中跳转到某个页面，从而间接地将该页面设置为App在启动后显示的第1个页面。在页面的最外层div组件中添加onswipe属性，从而在页面触发滑动事件时自动调用指定的自定义函数。

### 3.28.3 代码详解

在项目的pages子目录上单击右键，在弹出的菜单中选中New，然后在弹出的子菜单中单击JS Page，以新建一个名为report4的JS页面。该页面将被作为第4个训练报告页面。

打开report4.html文件。

将text组件中显示的文本修改为“第4个训练报告页面”。

上述讲解如代码清单3-120所示。

#### 代码清单3-120 report4.html

```
<div class="container">
  <text class="title">
    第4个训练报告页面
  </text>
</div>
```

打开report4.js文件。

因为在report4.html中没有使用title占位符，所以在report4.js中删除title及其动态数据绑定的值'World'。

上述讲解如代码清单3-121所示。

#### 代码清单3-121 report4.js

```
export default {
  data: {
    title: 'World'
  }
}
```

打开report4.css文件。

将title类选择器中width的值修改为454px，以让text组件中的文本“第4个训练报告页面”能够在一行内显示。

上述讲解如代码清单3-122所示。

#### 代码清单3-122 report4.css

```
.....  
.title {  
    font-size: 30px;  
    text-align: center;  
    width: 454px;  
    height: 100px;  
}
```

打开report4.js文件。

添加一个名为toNextPage的自定义函数，并定义一个名为e的形参。在函数体中通过e.direction的值判断滑动的方向。如果e.direction等于字符串“left”，那就跳转到主页面；如果e.direction等于字符串“down”，那就跳转到今日活动分布页面。

从 '@system.router' 中导入router。

上述讲解如代码清单3-123所示。

#### 代码清单3-123 report4.js

```
import router from '@system.router'  
  
export default {  
    data: {  
  
    },  
    toNextPage(e) {  
        switch(e.direction) {  
            case 'left':  
                router.replace({  
                    uri: 'pages/index/index'  
                })  
            }  
        }  
    }  
}
```



```
    });  
    break;  
    case 'down':  
        router.replace({  
            uri: 'pages/report3/report3'  
        });  
    }  
}  
}
```

---

打开report4.html文件。

在最外层的div组件中将onswipe属性的值设置为自定义的函数名toNextPage。这样，当用户在第4个训练报告页面中用手指滑动时，就会触发页面的onswipe事件从而自动调用自定义函数toNextPage。

上述讲解如代码清单3-124所示。

#### 代码清单3-124 report4.html

---

```
<div class="container" onswipe="toNextPage">  
    <text class="title">  
        第4个训练报告页面  
    </text>  
</div>
```

---

打开report3.js文件。

在toNextPage()函数中添加一个case分支：如果e.direction等于字符串“up”，那就跳转到第4个训练报告页面。

上述讲解如代码清单3-125所示。

#### 代码清单3-125 report3.js

---

```
import router from '@system.router'  
  
export default {  
    .....  
}
```

```
toNextPage(e) {
  switch(e.direction) {
    case 'left':
      router.replace({
        uri: 'pages/index/index'
      });
      break;
    case 'up':
      router.replace({
        uri: 'pages/report4/report4'
      });
      break;
    case 'down':
      router.replace({
        uri: 'pages/report2/report2'
      });
  }
}
```

---

打开index.js文件。

为了方便测试，我们在主页面的生命周期事件函数onInit()中跳转到第4个训练报告页面。这样，在App启动后我们看到的第1个页面就是第4个训练报告页面。

上述讲解如代码清单3-126所示。

#### 代码清单3-126 index.js

---

```
.....
onInit() {
  console.log("主页面的onInit()正在被调用");

  router.replace({
    uri: 'pages/report4/report4'
  });
},
.....
```

---

保存所有代码后打开Previewer，首先显示的是主页面，但是主页面一闪而过，接着显示的是第4个训练报告页面。运行效果如图3-112所示。

当在第4个训练报告页面中向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是第4个训练报告页面。当在第4个训练报告页面中向下滑动时，显示的是今日活动分布页面。当在今日活动分布页面中向上滑动时，显示的是第4个训练报告页面。



图3-112 第4个训练报告页面

## 3.29 任务29：在第4个训练报告页面中显示除压力分布图之外的所有内容

### 3.29.1 运行效果

该任务实现的运行效果是这样的：在第4个训练报告页面（后面都称之为压力分布页面）显示页面标题、一天内的几个时间点、压力最大值及其图标、压力最小值及其图标。运行效果如图3-113所示。



图3-113 压力分布页面

### 3.29.2 实现思路

在页面的生命周期事件函数onInit()中，随机生成若干个指定范围内的整数，以作为所有的压力数据。根据随机生成的整数统计所有压力的最大值和最小值，并通过动态数据绑定的方式将其显示在页面中。

### 3.29.3 代码详解

打开report4.html文件。

将text组件中显示的页面标题修改为“压力分布”，并在其外层嵌套一个div组件，以便对其样式进行设置。将该div组件的class属性的值设置为“title-container”。

在页面标题的下方添加一个canvas组件，以显示压力分布图，并将class属性的值设置为“canvas”。

在canvas组件的下方添加一个div组件，以显示一天中的几个时间点，并将class属性的值设置为“time-container”。在该div组件的内部嵌套一个text组件，将其class属性的值设置为“time”。通过动态数据绑定的方式指定for属性的值为“{{timeRange}}”，从而对report4.js中data里面的timeRange进行迭代。通过动态数据绑定的方式将text组件中显示的文本指定为“{{tem}}”。

在时间点的下方添加一个list组件，以显示压力的最大值、最小值及其图标，并将class属性的值设置为“list”。

在组件list的内部嵌套一个list-item组件，以显示列表中的每个列表项，并将class属性的值设置为“list-item”。通过动态数据绑定的方式指定for属性的值为“{{maxmin}}”，从而对report4.js中data里面的maxmin进行迭代。

每个列表项都由一张图片和一个文本组成，因此在list-item组件中添加一个image组件和一个text组件。

在image组件中将class属性的值设置为“icon”，并通过动态数据绑定的方式将src属性的值设置为“/common/{{tem.iconName}}.png”。这样，report4.js中data里面的maxmin可以是一个字典的数组，数组中的每个字典都包含一个key为iconName的元素。

在text组件中将class属性的值设置为“maxmin”，并通过动态数据绑定的方式将显示的文本设置为“{{tem.mValue}}”。这样，对于report4.js

中data里面的数组maxmin，其中的每个字典都包含一个key为mValue的元素。

上述讲解如代码清单3-127所示。

### 代码清单3-127 report4.html

```
<div class="container" onswipe="toNextPage">
  <div class="title-container">
    <text class="title">
      压力分布
    </text>
  </div>
  <canvas class="canvas">
  </canvas>
  <div class="time-container">
    <text class="time" for="{{timeRange}}">
      {{tem}}
    </text>
  </div>
  <list class="list">
    <list-item class="list-item" for="{{maxmin}}">
      <image class="icon"
src="/common/{{tem.iconName}}.png"/>
      <text class="maxmin">
        {{tem.mValue}}
      </text>
    </list-item>
  </list>
</div>
```

打开report4.css文件。

在container类选择器中删除display、left和top样式。将flex-direction的值设置为column，以在竖直方向上排列容器内的所有组件。将justify-content的值修改为flex-start，以让容器内的所有组件在主轴上向上对齐。

在title类选择器中删除text-align、width和height样式。将font-size的值修改为38px。将margin-top的值设置为40px，以让页面标题与页面的

上边缘保持一定的间距。

添加一个名为title-container的类选择器，以设置页面标题的样式。将justify-content和align-items都设置为center，以让容器内的组件在水平方向和竖直方向都居中对齐。将width和height的值分别设置为300px和130px。

添加一个名为canvas的类选择器，以设置压力分布图的样式。将width和height的值分别设置为383px和180px。在压力分布图中，总共需要绘制48根柱子。其中，每根柱子的宽度是7px，相邻两根柱子之间的间距是1px。因此，压力分布图的宽度=48×(7px+1px)-1px=383px。

添加一个名为time-container的类选择器，以设置时间点的样式。将width和height的值分别设置为383px和25px。将justify-content的值设置为space-between，以让容器内的组件在水平方向都两端对齐。将align-items的值设置为center，以让容器内的组件在竖直方向都居中对齐。

添加一个名为time的类选择器，以设置时间点文本的样式。将font-size的值设置为18px，将color的值设置为gray，并将letter-spacing的值设置为0px。

添加一个名为list的类选择器，以设置列表的样式。将flex-direction的值设置为row，以在水平方向上排列所有列表项。将width和height的值分别设置为200px和45px。将margin-top的值设置为30px，以让列表与其上面的时间点保持一定的间距。

添加一个名为list-item的类选择器，以设置列表项的样式。将justify-content和align-items都设置为center，以让列表项内的组件在水平方向和竖直方向都居中对齐。将width和height的值分别设置为100px和45px。

添加一个名为icon的类选择器，以设置压力的最大值图标和最小值图标的样式。将width和height的值都设置为32px。

添加一个名为maxmin的类选择器，以设置压力的最大值文本和最小值文本的样式。将font-size的值设置为30px。将letter-spacing的值设置为0px，以让数字之间的间距更紧凑。

上述讲解如代码清单3-128所示。

### 代码清单3-128 report4.css

```
.container {
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
    left: 0px;
    top: 0px;
    width: 454px;
    height: 454px;
}
.title-container {
    justify-content: center;
    align-items: center;
    width: 300px;
    height: 130px;
}
.title {
    margin-top: 40px;
    font-size: 38px;
    text-align: center;
    width: 454px;
    height: 100px;
}
.canvas {
    width: 383px;
    height: 180px;
}
.time-container {
    width: 383px;
    height: 25px;
    justify-content: space-between;
    align-items: center;
}
.time {
    font-size: 18px;
    color: gray;
}
```



```
        letter-spacing: 0px;
    }
    .list {
        flex-direction: row;
        width: 200px;
        height: 45px;
        margin-top: 30px;
    }
    .list-item {
        justify-content: center;
        align-items: center;
        width: 100px;
        height: 45px;
    }
    .icon {
        width: 32px;
        height: 32px;
    }
    .maxmin {
        font-size: 30px;
        letter-spacing: 0px;
    }
}
```

---

根据压力值将压力分为4种等级：焦虑、紧张、正常、放松，分别用4种颜色来表示：橙、黄、青、蓝。压力最大值的图标用一个三角形来表示，压力最小值的图标用一个倒三角形来表示。压力最大值和压力最小值都可能是4种等级中的其中一种，因此，要为压力最大值分别准备4张不同颜色的三角形图标，同时为压力最小值准备4张不同颜色的倒三角形图标。

把压力最大值的4张图标max-orange.png、max-yellow、max-cyan、max-blue添加到common目录中。把压力最小值的4张图标min-orange.png、min-yellow、min-cyan、min-blue也添加到common目录中。

打开report4.js文件。

在data中将timeRange占位符初始化为一个数组，该数组中的元素分别为“00：00”“06：00”“12：00”“18：00”“24：00”。

在data中将maxmin占位符初始化为一个字典数组。该数组中包含两个字典，分别表示压力最大值和压力最小值的相关信息。每个字典中都有两个元素，对应的key都是iconName和mValue，分别表示压力最大的图标名称和压力最值。对于第一个字典，将压力最大值的图标名称iconName初始化为""，并将压力最大值初始化为0。对于第二个字典，将压力最小值的图标名称iconName初始化为""，并将压力最小值初始化为0。

上述讲解如代码清单3-129所示。

#### 代码清单3-129 report4.js

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

export default {
  data: {
    timeRange: ["00:00", "06:00", "12:00", "18:00", "24:00"],
    maxmin: [{
      iconName: "",
      mValue: 0
    },
    {
      iconName: "",
      mValue: 0
    }
  ],
  toNextPage(e) {
    .....
  }
}
```

将report1.js中自定义的名为getRandomInt的函数复制过来，该函数用于随机生成一个介于min和max之间（包含min和max）的整数。

创建一个空数组并赋值给全局作用域变量pressures。

在页面的生命周期事件函数onInit()里通过for循环执行48次迭代。在每一次迭代中，调用自定义函数getRandomInt()随机生成一个介于1

和99之间的整数，并调用push()函数将随机生成的整数添加到数组pressures中。

分别调用Math.max.apply()和Math.min.apply()计算数组pressures中的最大值和最小值，然后分别赋值给data中的maxmin[0].mValue和maxmin[1].mValue。

定义一个名为getColorNameByValue的函数，其形参为value，该函数用于返回指定的压力值对应的颜色名称。

在函数onInit()的最后，调用自定义函数getColorNameByValue()，分别将压力最大值和压力最小值作为实参传递给形参value，将返回的颜色名称分别添加前缀“max-”和“min-”，然后分别赋值给data中的maxmin[0].iconName和maxmin[1].iconName。

上述讲解如代码清单3-130所示。

#### 代码清单3-130 report4.js

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

var pressures = []

export default {
  data: {
    .....
  },
  onInit() {
    for (let i = 0; i < 48; i++) {
      pressures.push(this.getRandomInt(1, 99));
    }

    this.maxmin[0].mValue = Math.max.apply(null, pressures);
    this.maxmin[1].mValue = Math.min.apply(null, pressures);

    this.maxmin[0].iconName = "max-" +
    this.getColorNameByValue(this.maxmin[0].mValue);
    this.maxmin[1].iconName = "min-" +
    this.getColorNameByValue(this.maxmin[1].mValue);
  },
```

```
getRandomInt(min, max) {  
    return Math.floor(Math.random() * (max - min + 1) ) + min;  
},  
getColorNameByValue(value) {  
    if (value >= 80 && value <= 99) {  
        return "orange";  
    } else if (value >= 60 && value <= 79) {  
        return "yellow";  
    } else if (value >= 30 && value <= 59) {  
        return "cyan";  
    } else if (value >= 1 && value <= 29) {  
        return "blue";  
    }  
},  
toNextPage(e) {  
    .....  
}  
}
```

保存所有代码后打开Previewer，在压力分布页面中显示出了页面标题、一天内的几个时间点、压力最大值及其图标、压力最小值及其图标。运行效果如图3-114所示。



图3-114 压力分布页面

## 3.30 任务30：在压力分布页面中显示绘制的压力分布图

### 3.30.1 运行效果

该任务实现的运行效果是这样的：在压力分布页面中显示绘制的压力分布图。运行效果如图3-115所示。



图3-115 显示压力分布图的压力分布页面

### 3.30.2 实现思路

通过canvas组件中的ref属性获得其对应的对象实例，并调用getContext('2d')函数获得2D绘制引擎。在遍历所有压力数据的过程中调

用fillRect()函数对压力分布图中的柱子逐一进行绘制。

### 3.30.3 代码详解

打开report4.html文件。

在canvas组件中将ref属性的值设置为“canvas”，以便能够在report4.js中通过引用获得canvas组件的对象实例。

上述讲解如代码清单3-131所示。

#### 代码清单3-131 report4.html

```
<div class="container" onswipe="toNextPage">
    .....
    <canvas class="canvas" ref="canvas">
    </canvas>
    .....
</div>
```

打开report4.js文件。

定义一个名为getColorHexByValue的函数，其形参为value，该函数用于返回指定的压力值对应的颜色十六进制值。

在页面的生命周期事件函数onShow()中，通过引用this.\$refs.canvas获得report4.html中canvas组件的对象实例，然后调用getContext('2d')函数获得2D绘图引擎，将其赋值给变量context。

接下来调用forEach()函数对pressures数组中的所有压力值进行遍历，其中element表示在遍历过程中数组的当前元素。

在遍历的过程中首先调用自定义函数getColorHexByValue()获取当前的压力值对应的颜色十六进制值，并将该值赋值给context的属性fillStyle。这样，就把当前压力值对应的颜色作为当前柱子的颜色。

绘制柱子要调用context的fillRect()函数。在调用fillRect()函数时需要传入某根柱子的左上角的坐标和坐标，以及该柱子的高度和宽度。以其中一根柱子为例，其在canvas组件中对应的坐标系如图3-116所示。canvas组件对应灰色的大矩形，要绘制的柱子对应蓝色的小矩形，柱子的左上顶点的坐标和坐标分别是leftTopX和leftTopY。

对于要绘制的48根柱子，每根柱子的宽度是7px，相邻两根柱子之间的间距是1px，因此，从左到右所有柱子的leftTopX从0开始依次递增8。在report4.css中将canvas组件的height设置为了180px，而所有压力值的最大值为100，因此在绘制每根柱子时其高度都根据压力值等比例地扩大1.8倍，也就是 $\text{element} \times 1.8$ 。每根柱子的leftTopY等于canvas组件的高度（180）减去柱子的高度 $\text{element} \times 1.8$ 。

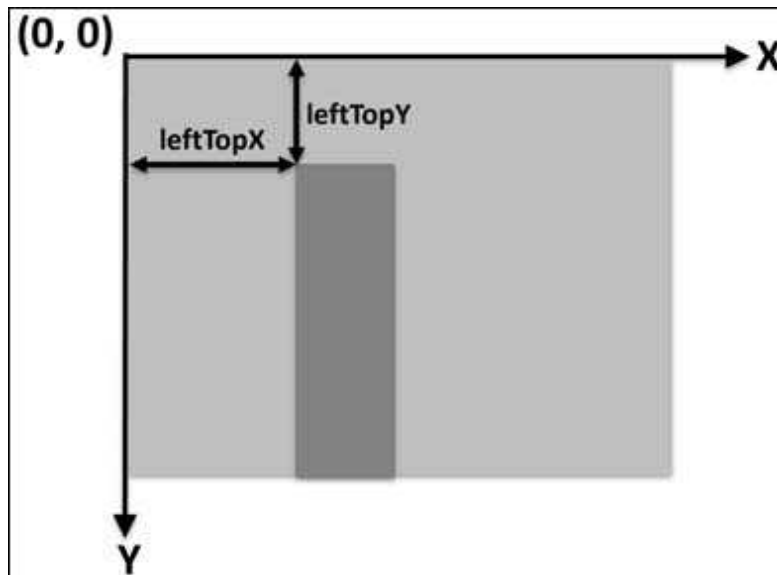


图3-116 绘制的柱子在canvas组件中对应的坐标系

上述讲解如代码清单3-132所示。

#### 代码清单3-132 report4.js

```
.....  
export default {
```

```

.....
getColorNameByValue(value) {
    .....
},
onShow() {
    var context = this.$refs.canvas.getContext('2d');

    let leftTopX = 0;
    pressures.forEach(element => {
        context.fillStyle = this.getColorHexByValue(element);

        let leftTopY = 180 - element * 1.8;
        let width = 7;
        let height = element * 1.8;

        context.fillRect(leftTopX, leftTopY, height, width);

        leftTopX += 8;
    });
},
getColorHexByValue(value) {
    if (value >= 80 && value <= 99) {
        return "#ffa500";
    } else if (value >= 60 && value <= 79) {
        return "#ffff00";
    } else if (value >= 30 && value <= 59) {
        return "#00ffff";
    } else if (value >= 1 && value <= 29) {
        return "#0000ff";
    }
},
.....
}

```

---

保存所有代码后打开Previewer，在压力分布页面中显示出了绘制的压力分布图。运行效果如图3-117所示。





图3-117 显示压力分布图的压力分布页面

### 3.31 任务31：添加第5个训练报告页面并响应滑动事件

#### 3.31.1 运行效果

该任务实现的运行效果是这样的：在App启动后首先显示的是主页面，但是主页面一闪而过，接着显示的是第5个训练报告页面。运行效果如图3-118所示。

当在第5个训练报告页面中向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是第5个训练报告页面。当在第5个训练报告页面中向下滑动时，显示的是压力分布页面。当在压力分布页面中向上滑动时，显示的是第5个训练报告页面。



图3-118 第5个训练报告页面

### 3.31.2 实现思路

在主页面的生命周期事件函数onInit()中跳转到某个页面，从而间接地将该页面设置为App在启动后显示的第1个页面。在页面的最外层div组件中添加onswipe属性，从而在页面触发滑动事件时自动调用指定的自定义函数。

### 3.31.3 代码详解

在项目的pages子目录上单击右键，在弹出的菜单中选中New，然后在弹出的子菜单中单击JS Page，以新建一个名为report5的JS页面。该页面将被作为第5个训练报告页面。

打开report5.html文件。

将text组件中显示的文本修改为“第5个训练报告页面”。

上述讲解如代码清单3-133所示。

### 代码清单3-133 report5.html

```
<div class="container">
  <text class="title">
    第5个训练报告页面
  </text>
</div>
```

打开report5.js文件。

因为在report5.html中没有使用title占位符，所以在report5.js中删除title及其动态数据绑定的值'World'。

上述讲解如代码清单3-134所示。

### 代码清单3-134 report5.js

```
export default {
  data: {
    title: 'World'
  }
}
```

打开report5.css文件。

将title类选择器中width的值修改为454px，以让text组件中的文本“第5个训练报告页面”能够在一行内显示。

上述讲解如代码清单3-135所示。

### 代码清单3-135 report5.css

```
.....  
.title {  
    font-size: 30px;  
    text-align: center;  
    width: 454px;  
    height: 100px;  
}
```

---

打开report5.js文件。

添加一个名为toNextPage的自定义函数，并定义一个名为e的形参。在函数体中通过e.direction的值判断滑动的方向。如果e.direction等于字符串“left”，那就跳转到主页面；如果e.direction等于字符串“down”，那就跳转到压力分布页面。

从'@system.router'中导入router。

上述讲解如代码清单3-136所示。

### 代码清单3-136 report5.js

---

```
import router from '@system.router'  
  
export default {  
    data: {  
  
    } ,  
    toNextPage(e) {  
        switch(e.direction) {  
            case 'left':  
                router.replace({  
                    uri: 'pages/index/index'  
                });  
                break;  
            case 'down':  
                router.replace({  
                    uri: 'pages/report4/report4'  
                });  
            }  
        }  
    }  
}
```

---

打开report5.html文件。

在最外层的div组件中将onswipe属性的值设置为自定义的函数名toNextPage。这样，当用户在第5个训练报告页面中用手指滑动时，就会触发页面的onswipe事件从而自动调用自定义的函数toNextPage。

上述讲解如代码清单3-137所示。

#### 代码清单3-137 report5.html

```
<div class="container" onswipe="toNextPage">
  <text class="title">
    第5个训练报告页面
  </text>
</div>
```

打开report4.js文件。

在toNextPage()函数中添加一个case分支：如果e.direction等于字符串“up”，那就跳转到第5个训练报告页面。

上述讲解如代码清单3-138所示。

#### 代码清单3-138 report4.js

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

export default {
  .....
  toNextPage(e) {
    switch(e.direction) {
      case 'left':
        router.replace({
          uri: 'pages/index/index'
        });
        break;
      case 'up':
        router.replace({
          uri: 'pages/report5/report5'
        });
    }
  }
}
```

```
        break;
      case 'down':
        router.replace({
          uri: 'pages/report3/report3'
        });
      }
    }
  }
}
```

---

打开index.js文件。

为了方便测试，我们在主页面的生命周期事件函数onInit()中跳转到第5个训练报告页面。这样，在App启动后我们看到的第1个页面就是第5个训练报告页面。

上述讲解如代码清单3-139所示。

#### 代码清单3-139 index.js

---

```
.....
onInit() {
  console.log("主页面的onInit()正在被调用");

  router.replace({
    uri: 'pages/report5/report5'
  });
},
.....
```

---

保存所有代码后打开Previewer，首先显示的是主页面，但是主页面一闪而过，接着显示的是第5个训练报告页面。运行效果如图3-119所示。



图3-119 第5个训练报告页面

当在第5个训练报告页面中向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是第5个训练报告页面。当在第5个训练报告页面中向下滑动时，显示的是压力分布页面。当在压力分布页面中向上滑动时，显示的是第5个训练报告页面。

## 3.32 任务32：在第5个训练报告页面中显示除弧形和星号之外的所有内容

### 3.32.1 运行效果

该任务实现的运行效果是这样的：在第5个训练报告页面（后面都称之为最大摄氧量页面）显示页面标题、最大摄氧量及其单位、摄氧

量等级水平。运行效果如图3-120所示。



图3-120 最大摄氧量页面

### 3.32.2 实现思路

通过数值对应的图片来显示具有较大字体的最大摄氧量数值。在image组件中通过if属性的布尔值来判断是否要显示该组件。

### 3.32.3 代码详解

打开report5.html文件。

将text组件中显示的页面标题修改为“最大摄氧量”，并在其外层嵌套一个div组件，以便对其样式进行设置。将该div组件的class属性的值



设置为“title-container”。

在页面标题的下方添加一个div组件，将其class属性的值设置为“number-container”。在该div组件的内部嵌套两个image组件，以显示最大摄氧量的数值对应的图片。将两个image组件的class属性的值都设置为“number-icon”，并通过动态数据绑定的方式将src属性的值分别设置为“/common/{{first}}.png”和“/common/{{second}}.png”，以指定两张图片在项目中的路径。当最大摄氧量小于10时，只需要显示第二个组件image中的数值图片，因此在第一个image组件中通过动态数据绑定的方式将if属性的值设置为“{{isShow}}”。这样，在report5.js中就可以通过isShow占位符的值到底是true还是false，来决定是否显示第一个image组件。

在最大摄氧量的下方添加一个text组件，以显示最大摄氧量的单位。将class属性的值设置为“unit”。将text组件中显示的文本设置为ml/kg/min。

在最大摄氧量的单位下方添加一个text组件，以显示等级水平。将class属性的值设置为“level”。通过动态数据绑定的方式将text组件中显示的文本设置为“{{level}}水平”。

上述讲解如代码清单3-140所示。

#### 代码清单3-140 report5.html

```
<div class="container" onswipe="toNextPage">
  <div class="title-container">
    <text class="title">
      最大摄氧量
    </text>
  </div>
  <div class="number-container">
    <image if="{{isShow}}" class="number-icon"
src="/common/{{first}}.png"/>
    <image class="number-icon" src="/common/{{second}}.png"/>
  </div>
```

```
<text class="unit">
  ml/kg/min
</text>
<text class="level">
  {{level}}水平
</text>
</div>
```

---

注意：不能将image组件中的if属性换成show属性，否则，当占位符{{isShow}}的值为false从而不显示第1个组件image时，第2个组件image并不会在容器中居中。这是因为当if属性的值为false时，其对应的组件不会占用页面中的空间；当show属性为false时，其对应的组件会占用页面中的空间。

打开report5.css文件。

在container类选择器中删除display、left和top样式。将flex-direction的值设置为column，以在竖直方向上排列容器内的所有组件。将justify-content的值修改为flex-start，以让容器内的所有组件在主轴上向上对齐。

在title类选择器中删除text-align、width和height样式。将font-size的值修改为38px。将margin-top的值设置为40px，以让页面标题与页面的上边缘保持一定的间距。

添加一个名为title-container的类选择器，以设置页面标题的样式。将justify-content和align-items都设置为center，以让容器内的组件在水平方向和竖直方向都居中对齐。将width和height的值分别设置为300px和130px。

添加一个名为number-container的类选择器，以设置最大摄氧量的样式。将width和height的值分别设置为160px和180px。将justify-content和align-items的值都设置为center，以让容器内的组件在水平方向和竖直方向都居中对齐。

添加一个名为number-icon的类选择器，以设置最大摄氧量的数值对应图片的样式。将width和height的值分别设置为80px和100px。将margin-top的值设置为40px，以让图片与其上面的页面标题保持一定的间距。

添加一个名为unit的类选择器，以设置最大摄氧量单位的样式。将width和height的值分别设置为200px和30px。将color的值设置为gray，以将文本的颜色显示为灰色。将text-align的值设置为center，以让显示的文本居中对齐。将font-size的值设置为24px。

添加一个名为level的类选择器，以设置等级水平的样式。将width和height的值分别设置为200px和40px。将text-align的值设置为center，以让显示的文本居中对齐。将margin-top的值设置为30px，以让等级水平与其上面的最大摄氧量单位保持一定的间距。

上述讲解如代码清单3-141所示。

#### 代码清单3-141 report5.css

```
.container {
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
    left: 0px;
    top: 0px;
    width: 454px;
    height: 454px;
}
.title-container {
    justify-content: center;
    align-items: center;
    width: 300px;
    height: 130px;
}
.title {
    margin-top: 40px;
    font-size: 38px;
    text-align: center;
```

```
        width: 454px;
        height: 100px;
    }
    .number-container {
        width: 160px;
        height: 180px;
        justify-content: center;
        align-items: center;
    }
    .number-icon {
        width: 80px;
        height: 100px;
        margin-top: 40px;
    }
    .unit {
        width: 200px;
        height: 30px;
        color: gray;
        text-align: center;
        font-size: 24px;
    }
    .level {
        width: 200px;
        height: 40px;
        text-align: center;
        margin-top: 30px;
    }
}
```

---

因为要显示的最大摄氧量数值的字体比较大，所以我们用图片来显示对应的数值。把0~9对应的10张图片添加到common目录中，它们分别是num-0.png、num-1.png、num-2.png、num-3.png、num-4.png、num-5.png、num-6.png、num-7.png、num-8.png、num-9.png。

打开report5.js文件。

在data中将isShow占位符初始化为true，将first、second和level占位符都初始化为""。

将report1.js中自定义的名为getRandomInt的函数复制过来，该函数用于随机生成一个介于min和max之间（包含min和max）的整数。

在页面的生命周期事件函数onInit()中，调用自定义函数getRandomInt()随机生成一个介于1和70之间的整数，将其赋值给变量

vo2max。将随机生成的整数转换为字符串，并赋值给变量vo2max\_str。对于转换后得到的字符串，如果其长度为2，那么将其第1个字符和第2个字符都添加前缀“num-”，之后再分别赋值给data中的first和second；如果其长度为1，那么将其添加前缀“num-”之后赋值给data中的second，然后将data中的isShow设置为false，这样，就不会显示report5.html中的第1个image组件了。

定义一个名为getLevelByValue的函数，其形参为value，该函数用于返回最大摄氧量对应的等级。在函数体中，定义一个由7个元素组成的数组，这些元素分别表示最大摄氧量的7个等级。等级“超低”对应的区间是[1, 10]；等级“低”对应的区间是[11, 20]；等级“较低”对应的区间是[21, 30]；等级“一般”对应的区间是[31, 40]；等级“高”对应的区间是[41, 50]；等级“优秀”对应的区间是[51, 60]；等级“卓越”对应的区间是[61, 70]。通过调用Math.floor((value-1)/10)可以得到最大摄氧量的等级在levels数组中对应的索引。

在onInit()函数的最后，调用自定义函数getLevelByValue()，将最大摄氧量作为实参传递给形参value，以返回最大摄氧量对应的等级。将返回的等级赋值给data中的level。

上述讲解如代码清单3-142所示。

#### 代码清单3-142 report5.js

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

export default {
  data: {
    isShow: true,
    first: "",
    second: "",
    level: "",
  },
  onInit() {
    let vo2max = this.getRandomInt(1, 70);
```

```

    let vo2max_str = vo2max.toString();
    if (vo2max_str.length == 2) {
        this.first = "num-" + vo2max_str[0];
        this.second = "num-" + vo2max_str[1];
    } else {
        this.second = "num-" + vo2max_str;
        this.isShow = false;
    }

    this.level = this.getLevelByValue(vo2max);
},
getRandomInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1) ) + min;
},
getLevelByValue(value) {
    let levels = ["超低", "低", "较低", "一般", "高", "优秀", "卓越"];
    let index = Math.floor((value - 1) / 10);
    return levels[index];
},
toNextPage(e) {
    .....
}
}

```

---

保存所有代码后打开Previewer，在最大摄氧量页面显示出了页面标题、最大摄氧量及其单位、摄氧量等级水平。运行效果如图3-121所示。



图3-121 最大摄氧量页面

### 3.33 任务33：在最大摄氧量页面显示绘制的弧形

#### 3.33.1 运行效果

该任务实现的运行效果是这样的：在最大摄氧量页面中显示7个不同颜色的弧形。运行效果如图3-122所示。



图3-122 7个不同颜色的弧形

### 3.33.2 实现思路

通过将progress组件的type属性设置为“arc”来绘制弧形。通过stack组件来堆叠其中的子组件，从而分别绘制7个不同颜色的弧形。

### 3.33.3 代码详解

打开report5.html文件。

在最外层div组件的外部嵌套一个stack组件，这样其中的子组件会按照顺序依次入栈，从而后一个入栈的子组件会堆叠在前一个入栈的子组件的上面。将最外层div组件的onswipe属性转移到stack组件中。

在stack组件的内部添加一个progress组件，以显示一个进度条。将class属性的值设置为“progress”。将type属性的值设置为“arc”，以显示



一个弧形进度条。将percent属性的值设置为“50”，以设置进度条的进度为50%。添加一个style属性，以设置进度条的样式，其中，将颜色color设置为#ff0000，并将起始角度start-angle设置为220。起始角度的示意图如图3-123所示。

上述讲解如代码清单3-143所示。

#### 代码清单3-143 report5.html

```
<stack class="stack" onswipe="toNextPage">
  <div class="container" onswipe="toNextPage">
    .....
  </div>
  <progress class="progress" type="arc" percent="50"
style="color:#ff0000; start-angle:220;"/>
</stack>
```

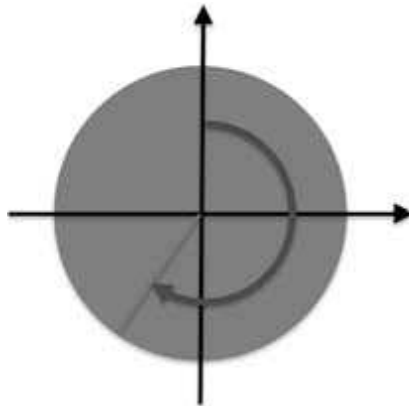


图3-123 起始角度的示意图

打开report5.css文件。

添加一个名为stack的类选择器，以设置report5.html中stack组件的样式。将width和height的值都设置为454px。

添加一个名为progress的类选择器，以设置弧形进度条的样式。将width和height的值都设置为454px。将total-angle的值设置为40deg，以设置弧形进度条的总角度。将stroke-width的值设置为16px，以设置弧

形进度条的宽度。将radius的值设置为227px，以设置弧形进度条所对应圆的半径。将center-x和center-y的值都设置为227px，以设置弧形进度条所对应圆心的坐标和坐标。

上述讲解如代码清单3-144所示。

#### 代码清单3-144 report5.css

```
.stack {
    width: 454px;
    height: 454px;
}
.container {
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
    width: 454px;
    height: 454px;
}
.....
.level {
    width: 200px;
    height: 40px;
    text-align: center;
    margin-top: 30px;
}
.progress {
    width: 454px;
    height: 454px;
    total-angle: 40deg;
    stroke-width: 16px;
    radius: 227px;
    center-x: 227px;
    center-y: 227px;
}
```

保存所有代码后打开Previewer，在最大摄氧量页面中显示出了一个进度为50%的弧形进度条。运行效果如图3-124所示。



图3-124 进度为50%的弧形进度条

打开report5.html文件。

将progress组件中percent属性的值修改为“100”。在progress组件中添加一个for属性，通过动态数据绑定的方式设置其属性值为“{{styles}}”。修改progress组件中style属性的值，将其中color样式的值修改为{{ \$item.color }}，并将其中start-angle样式的值修改为{{ \$item.startAngle }}。

上述讲解如代码清单3-145所示。

#### 代码清单3-145 report5.html

```
<stack class="stack" onswipe="toNextPage">
  <div class="container">
    .....
  </div>
  <progress class="progress" type="arc" percent="100" for="
  {{styles}}" style="color:
  {{ $item.color }};start-angle:{{ $item.startAngle }};"/>
</stack>
```

打开report5.js文件。

在data中将styles占位符初始化为一个数组，该数组中包含7个字典，每个字典中都包含一个key为color的元素和一个key为startAngle的元素，分别用于指定弧形进度条的颜色和起始角度。

上述讲解如代码清单3-146所示。

#### 代码清单3-146 report5.js

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

export default {
  data: {
    isShow: true,
    first: "",
    second: "",
    level: "",
    styles: [
      {
        color: "#ff0000",
        startAngle: 220
      },
      {
        color: "#ffa500",
        startAngle: 260
      },
      {
        color: "#ffd700",
        startAngle: 300
      },
      {
        color: "#ffff00",
        startAngle: 340
      },
      {
        color: "#adff2f",
        startAngle: 20
      },
      {
        color: "#00ffff",
        startAngle: 60
      },
      {
```

```
        color: "#4169e1",
        startAngle: 100
    },
],
},
.....
}
```

---

保存所有代码后打开Previewer，在最大摄氧量页面中显示出了7个不同颜色的弧形。运行效果如图3-125所示。

绘制的7段圆弧与表盘有一定的偏差，因此需要对弧形的圆心坐标和半径做一些微调。

打开report5.css文件。

经过多次测试，将radius的值修改为220px，将center-x的值修改为224px，将center-y的值修改为229px。

上述讲解如代码清单3-147所示。

#### 代码清单3-147 report5.css

---

```
.....
.progress {
    width: 454px;
    height: 454px;
    total-angle: 40deg;
    stroke-width: 16px;
    radius: 220px;
    center-x: 224px;
    center-y: 229px;
}
```

---

保存所有代码后打开Previewer，在最大摄氧量页面中显示出了微调之后的弧形。运行效果如图3-126所示。



图3-125 7个不同颜色的弧形



图3-126 微调之后的弧形

3.34 任务34：在最大摄氧量界面的对应弧形和角度上显示星号

### 3.34.1 运行效果

该任务实现的运行效果是这样的：在最大摄氧量界面的对应弧形和角度上显示星号。运行效果如图3-127所示。



图3-127 显示星号的摄氧量界面

### 3.34.2 实现思路

已知圆心的坐标是  $(x_0, y_0)$ ，圆的半径是  $r$ ，圆上某一点  $P$  的角度是  $\text{angle}$ （以  $Q$  为起点，逆时针时  $\text{angle}$  为负数）。圆的示意图如图3-128所示。

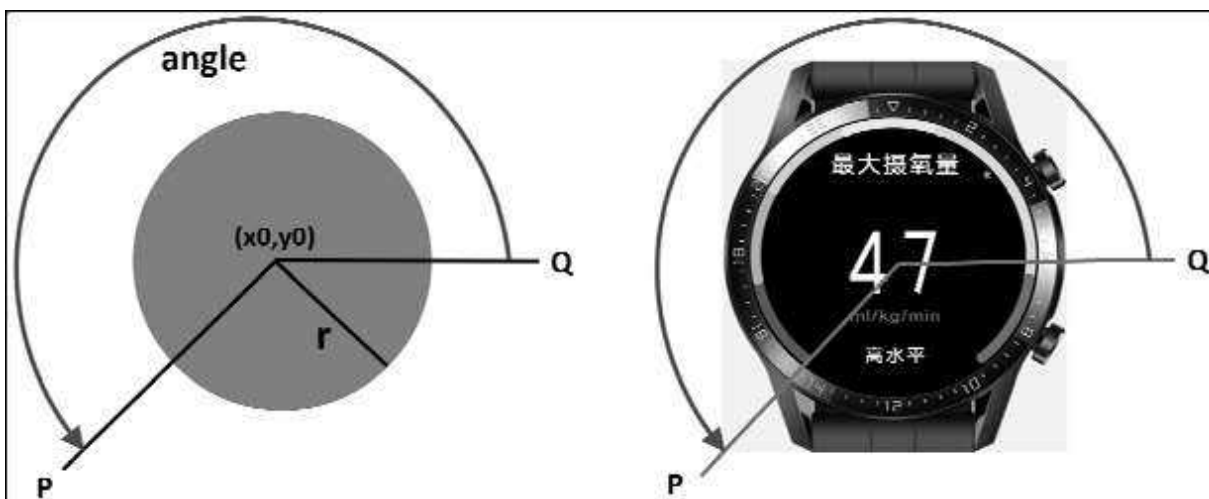


图3-128 圆的示意图

假设点P的坐标是 $(x, y)$ ，那么 $x$ 和 $y$ 的计算公式为：

$$x = x_0 + r \times \cos(\text{angle} \times \text{PI} / 180)$$

$$y = y_0 + r \times \sin(\text{angle} \times \text{PI} / 180)$$

对于我们绘制的红色背景的第1个弧形，如果将该弧形的起始点对应于上图中的P，那么相应的角度angle是 $-230^\circ$ 。我们绘制的7个弧形的总角度是 $280^\circ$ ，最大摄氧量是 $70 \text{ ml}/(\text{kg} \cdot \text{min})$ ，因此每1个单位的摄氧量对应的角度是 $280^\circ / 70 = 4^\circ$ 。当最大摄氧量的值为 $m$ 时，它在弧形上所对应点的角度 $\text{angle} = -230 + m \times 4$ 。

### 3.34.3 代码详解

打开report5.html文件。

在progress组件的下方添加一个canvas组件，以在对应弧形和角度上绘制星号。将class和ref属性的值都设置为“canvas”。

上述讲解如代码清单3-148所示。

代码清单3-148 report5.html



---

```
<stack class="stack" onswipe="toNextPage">
    .....
    <progress class="progress" type="arc" percent="100" for="
{{styles}}" style="color:{{tem.
color}}; start-angle:{{tem.startAngle}};"/>
    <canvas class="canvas" ref="canvas">
    </canvas>
</stack>
```

---

打开report5.css文件。

添加一个名为canvas的类选择器，以设置report5.html中canvas组件的样式。将width和height的值都设置为454px。将background-color的值设置为transparent，以让组件canvas的背景颜色是透明的。

上述讲解如代码清单3-149所示。

#### 代码清单3-149 report5.css

---

```
.....
.progress {
    width: 454px;
    height: 454px;
    total-angle: 40deg;
    stroke-width: 16px;
    radius: 220px;
    center-x: 224px;
    center-y: 229px;
}
.canvas {
    width:454px;
    height:454px;
    background-color: transparent;
}
```

---

打开report5.js文件。

在页面的生命周期事件函数onInit()中，根据实现思路中的计算公式计算出星号的坐标。根据多次测试，为了达到最好的显示效果，推

荐使用的圆心坐标是（218，218），推荐使用的圆半径是193。

在页面的生命周期事件函数onShow()中，通过引用this.\$refs.canvas获得report5.html中canvas组件的对象实例，然后调用getContext('2d')函数获得2D绘图引擎，将其赋值给变量context。将context的fillStyle属性设置为“#ffff00”，以将绘制颜色设置为黄色。调用函数fillText("\*", x, y)在指定的坐标（x，y）处绘制指定的文本“\*”。

上述讲解如代码清单3-150所示。

#### 代码清单3-150 report5.js

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

var x = 0;
var y = 0;

export default {
  .....
  onInit() {
    .....
    this.level = this.getLevelByValue(vo2max);

    let angle = -230 + vo2max * (280 / 70);
    x = Math.round(218 + 193 * Math.cos(angle * Math.PI / 180));
    y = Math.round(218 + 193 * Math.sin(angle * Math.PI / 180));
  },
  onShow() {
    var context = this.$refs.canvas.getContext("2d");
    context.fillStyle = "#ffff00";
    context.fillText("*" , x, y);
  },
  .....
}
```

保存所有代码后打开Previewer，在最大摄氧量界面的对应弧形和角度上显示出了星号。运行效果如图3-129所示。



图3-129 显示星号的最大摄氧量界面

## 3.35 任务35：添加学习交流联系方式页面并响应滑动事件

### 3.35.1 运行效果

该任务实现的运行效果是这样的：在App启动后首先显示的是主页面，但是主页面一闪而过，接着显示的是学习交流联系方式页面。运行效果如图3-130所示。



图3-130 学习交流联系方式页面

当在学习交流联系方式页面向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是学习交流联系方式页面。当在学习交流联系方式页面中向下滑动时，显示的是最大摄氧量页面。当在最大摄氧量页面中向上滑动时，显示的是学习交流联系方式页面。

### 3.35.2 实现思路

在主页面的生命周期事件函数onInit()中跳转到某个页面，从而间接地将该页面设置为App在启动后显示的第1个页面。在页面的最外层div组件中添加onswipe属性，从而在页面触发滑动事件时自动调用指定的自定义函数。

### 3.35.3 代码详解

在项目的pages子目录上单击右键，在弹出的菜单中选中New，然后在弹出的子菜单中单击JS Page，以新建一个名为contact的JS页面。该页面将被作为学习交流联系方式页面。

打开contact.html文件。

将text组件中显示的文本修改为“学习交流联系方式”，以作为页面标题。

上述讲解如代码清单3-151所示。

#### 代码清单3-151 contact.html

```
<div class="container">
  <text class="title">
    学习交流联系方式
  </text>
</div>
```

打开contact.js文件。

因为在contact.html中没有使用title占位符，所以在contact.js中删除title及其动态数据绑定的值'World'。

上述讲解如代码清单3-152所示。

#### 代码清单3-152 contact.js

```
export default {
  data: {
    title: 'World'
  }
}
```

打开contact.css文件。

将title类选择器中width的值修改为454px，以让text组件中的文本“学习交流联系方式”能够在一行内显示。

上述讲解如代码清单3-153所示。

#### 代码清单3-153 contact.css

```
.....  
.title {  
    font-size: 30px;  
    text-align: center;  
    width: 454px;  
    height: 100px;  
}
```

打开contact.js文件。

添加一个名为toNextPage的自定义函数，并定义一个名为e的形参。在函数体中通过e.direction的值判断滑动的方向。如果e.direction等于字符串“left”，那就跳转到主页面；如果e.direction等于字符串“down”，那就跳转到最大摄氧量页面。

从 '@system.router' 中导入router。

上述讲解如代码清单3-154所示。

#### 代码清单3-154 report5.js

```
import router from '@system.router'  
  
export default {  
    data: {  
  
    },  
    toNextPage(e) {  
        switch(e.direction) {  
            case 'left':  
                router.replace({  
                    uri: 'pages/index/index'
```

```
    });  
    break;  
    case 'down':  
        router.replace({  
            uri: 'pages/report5/report5'  
        });  
    }  
}  
}
```

---

打开contact.html文件。

在最外层的div组件中将onswipe属性的值设置为自定义的函数名toNextPage。这样，当用户在学习交流联系方式页面中用手指滑动时，就会触发页面的onswipe事件从而自动调用自定义的函数toNextPage。

上述讲解如代码清单3-155所示。

#### 代码清单3-155 contact.html

---

```
<div class="container" onswipe="toNextPage">  
    <text class="title">  
        学习交流联系方式  
    </text>  
</div>
```

---

打开report5.js文件。

在toNextPage()函数中添加一个case分支：如果e.direction等于字符串“up”，那就跳转到学习交流联系方式页面。

上述讲解如代码清单3-156所示。

#### 代码清单3-156 report5.js

---

```
import router from '@system.router'  
  
export default {  
    .....  
}
```

```
toNextPage(e) {
  switch(e.direction) {
    case 'left':
      router.replace({
        uri: 'pages/index/index'
      });
      break;
    case 'up':
      router.replace({
        uri: 'pages/contact/contact'
      });
      break;
    case 'down':
      router.replace({
        uri: 'pages/report4/report4'
      });
  }
}
```

---

打开index.js文件。

为了方便测试，我们在主页面的生命周期事件函数onInit()中跳转到学习交流联系方式页面。这样，在App启动后我们看到的第1个页面就是学习交流联系方式页面。

上述讲解如代码清单3-157所示。

#### 代码清单3-157 index.js

---

```
.....
onInit() {
  console.log("主页面的onInit()正在被调用");

  router.replace({
    uri: 'pages/contact/contact'
  });
},
.....
```

---



保存所有代码后打开Previewer，在App启动后首先显示的是主页面，但是主页面一闪而过，接着显示的是学习交流联系方式页面。运行效果如图3-131所示。



图3-131 学习交流联系方式页面

当在学习交流联系方式页面中向左滑动时，首先显示的是主页面，但是主页面一闪而过，接着显示的是学习交流联系方式页面。当在学习交流联系方式页面中向下滑动时，显示的是最大摄氧量页面。当在最大摄氧量页面中向上滑动时，显示的是学习交流联系方式页面。

### 3.36 任务36：在学习交流联系方式页面中显示二维码并完成项目收尾工作

### 3.36.1 运行效果

该任务实现的运行效果是这样的：在App启动后首先显示的是主页面。在压力占比页面中显示滑动的提示信息，如图3-132所示。

在学习交流联系方式页面中显示两个二维码和作者信息，如图3-133所示。



图3-132 显示滑动提示信息的压力占比页面



图3-133 显示二维码的学习交流联系方式页面

### 3.36.2 实现思路

使用image组件显示二维码图片。

### 3.36.3 代码详解

打开contact.html文件。

在text组件的外层嵌套一个div组件，以便对其样式进行设置。将该div组件的class属性的值设置为“title-container”。

在页面标题的下方添加一个div组件，以显示两张二维码图片。将class属性的值设置为“qrcode-container”。在该div组件中添加两个image组件，以分别显示两张二维码图片。将class属性的值都设置

为“qrcode”。将src属性的值分别设置

为“/common/qrcode1.png”和“/common/qrcode2.png”，以分别指定两张二维码图片在项目中的位置。

在两张二维码图片的下方添加一个text组件，以显示作者信息。将class属性的值设置为“author”。将显示的文本设置为“作者：张荣超”。

上述讲解如代码清单3-158所示。

### 代码清单3-158 contact.html

```
<div class="container" onswipe="toNextPage">
  <div class="title-container">
    <text class="title">
      学习交流联系方式
    </text>
  </div>
  <div class="qrcode-container">
    <image class="qrcode" src="/common/qrcode1.png"/>
    <image class="qrcode" src="/common/qrcode2.png"/>
  </div>
  <text class="author">
    作者：张荣超
  </text>
</div>
```

打开contact.css文件。

在container类选择器中删除display、left和top样式。将flex-direction的值设置为column，以在竖直方向上排列容器内的所有组件。将justify-content的值修改为flex-start，以让容器内的所有组件在主轴上向上对齐。

在title类选择器中删除text-align、width和height样式。

添加一个名为title-container的类选择器，以设置页面标题的样式。将justify-content的值设置为center，以让容器内的组件在水平方向居中

对齐。将align-items的值设置为flex-end，以让容器内的组件在竖直方向向下对齐。将width和height的值分别设置为400px和100px。

添加一个名为qrcode-container的类选择器，以设置两张二维码图片的样式。将width和height的值分别设置为400px和260px。将justify-content的值设置为space-between，以让容器内的组件在水平方向两端对齐。将align-items的值设置为center，以让容器内的组件在竖直方向居中对齐。

添加一个名为qrcode的类选择器，以设置每张二维码图片的样式。将width和height的值都设置为180px。

添加一个名为author的类选择器，以设置作者信息的样式。将width和height的值分别设置为300px和50px。将font-size的值设置为30px。将text-align的值设置为center。

上述讲解如代码清单3-159所示。

#### 代码清单3-159 contact.css

```
.container {
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
    left: 0px;
    top: 0px;
    width: 454px;
    height: 454px;
}
.title-container {
    justify-content: center;
    align-items: flex-end;
    width: 400px;
    height: 100px;
}
.title {
    font-size: 30px;
    text-align: center;
    width: 454px;
```

```
        height: 100px;
    }
    .qrcode-container {
        width: 400px;
        height: 260px;
        justify-content: space-between;
        align-items: center;
    }
    .qrcode {
        width: 180px;
        height: 180px;
    }
    .author {
        width: 300px;
        height: 50px;
        font-size: 30px;
        text-align: center;
    }
}
```

---

把两张二维码图片qrcode1.png和qrcode2.png添加到common目录中。

保存所有代码后打开Previewer，在学习交流联系方式页面中显示出了两个二维码和作者信息，如图3-134所示。



图3-134 显示二维码的学习交流联系方式页面

接下来，我们在压力占比页面添加滑动的提示信息。

打开report1.html文件。

在list组件的下方添加一个div组件，以显示滑动的提示信息。将class属性的值设置为“tip-container”。在该list组件的内部添加两个text组件，以分别显示两条滑动提示信息。将class属性的值都设置为“tip”。将显示的文本分别设置为相应的滑动提示信息。

上述讲解如代码清单3-160所示。

#### 代码清单3-160 report1.html

```
<div class="container" onswipe="toNextPage">
    .....
    <list class="state-wrapper">
        .....
    </list>
    <div class="tip-container">
        <text class="tip">
            【上滑或下滑】：切换训练报告页面
        </text>
        <text class="tip">
            【左滑】：跳转到主页面
        </text>
    </div>
</div>
```

打开report1.css文件。

添加一个名为tip-container的类选择器，以设置两条滑动提示信息的样式。将width和height的值分别设置为400px和80px。将flex-direction的值设置为column，以在竖直方向上排列容器内的所有组件。将justify-content的值设置为space-around，以让容器内的所有组件在竖直

方向上分散对齐。将align-items的值设置为center，以让容器内的所有组件在水平方向上居中对齐。

添加一个名为tip的类选择器，以设置单条滑动提示信息的样式。将font-size的值设置为18px。将color的值设置为#ffa500。

上述讲解如代码清单3-161所示。

#### 代码清单3-161 report1.css

```
.....
.progress-bar {
  width: 320px;
  height: 5px;
  margin-top: 5px;
}
.tip-container {
  width: 400px;
  height: 80px;
  flex-direction: column;
  justify-content: space-around;
  align-items: center;
}
.tip {
  font-size: 18px;
  color: #ffa500;
}
```

打开index.js文件。

为了方便测试，我们在主页面的生命周期事件函数onInit()中跳转到压力占比页面。这样，在App启动后我们看到的第1个页面就是压力占比页面。上述讲解如代码清单3-162所示。

#### 代码清单3-162 index.js

```
.....

export default {
```



```

.....
onInit() {
  console.log("主页面的onInit()正在被调用");

  router.replace({
    uri: 'pages/report1/report1'
  });
},
.....
}

```

保存所有代码后打开Previewer，在压力占比页面中显示出了滑动的提示信息，如图3-135所示。



图3-135 显示滑动提示信息的压力占比页面

打开index.js文件。

在主页面的生命周期事件函数onInit()中删除页面跳转的代码。这样，在App启动后我们看到的第1个页面是主页面。

上述讲解如代码清单3-163所示。

代码清单3-163 index.js

---

```
.....  
  
export default {  
  .....  
  onInit() {  
    console.log("主页面的onInit()正在被调用");  
  
    router.replace({  
      uri: 'pages/report1/report1'  
    });  
  },  
  .....  
}
```

---

保存所有代码后打开Previewer，在App启动后首先显示的是主页面。

到此，我们就在鸿蒙智能手表上完成了呼吸训练App的全部功能。

亲爱的读者，当你学习到这里的时候，运行在鸿蒙智能手表上的呼吸训练实战项目视频教程已经上线了，快点儿扫码学习吧（二维码见图3-134）！

随着鸿蒙的不断发展成熟，我会在以上二维码的平台中跟大家分享更多的视频教程，包括搭载了鸿蒙系统的手机、智慧屏、汽车等，敬请大家期待！



异步社区WWW.epubit.com

新浪微博@人邮异步社区

投稿/反馈邮箱[contact@epubit.com.cn](mailto:contact@epubit.com.cn)



*Your gateway to knowledge and culture. Accessible for everyone.*



[z-library.se](http://z-library.se)

[singlelogin.re](http://singlelogin.re)

[go-to-zlibrary.se](http://go-to-zlibrary.se)

[single-login.ru](http://single-login.ru)



[Official Telegram channel](#)



[Z-Access](#)



<https://wikipedia.org/wiki/Z-Library>