华 中 科 技 大 学

课 程 实 验 报 告

课程名称：鸿蒙移动应用技术

实验名称：UI 与交互与 Ability 模型-ListGame 小游戏

实验时间：2024 年 5 月 17 日

实验地点：明德楼 B219

指导教师：董枫

专业班级：信安 2102 班

学　　号：U202112003

姓　　名：刘嗣杰

报告日期：2024.5.17

成绩评定

实验完成质量（70 分）	报告撰写质量（30 分）	总成绩
实验步骤清晰、详细、深入，实验记录真实完整等	报告规范、完整、通顺、详实	

实验一：UI 与交互

1. 实验目的

熟悉鸿蒙应用基础的代码框架并学习使用调试工具。

掌握 ArkTS 中声明式 UI 的编写方法与渲染逻辑。

掌握常用组件的使用方法，学习如何添加样式。

2. 实验内容

1. 完成单页 page 中声明式 UI 的编写。

2. 使用样式 API 修改文本颜色与按钮间距，并使用响应式变量使得文本颜色可以在完成目标时变化。

3. 添加”New Game“按钮并完成组件事件的处理逻辑，同时使用条件渲染切换按钮的显示状态。

3. 实验指南

1. 完成单页 page 中声明式 UI 的编写。

从这里你可以了解 ArkTS 的基础概念：基本语法概述。从这里你可以了解声明式 UI 的基础概念：声明式 UI 描述。只有 push 和 pop 你是无法获得 [2,2] 来赢得游戏的，你需要添加 reverse 按钮并通过 list 的 reverse 方法来操作你的 List。阅读已有的代码会很有帮助，参考给出的代码与 Button 组件的文档。

2. 使用样式 API 修改文本颜色与按钮间距，并使用响应式变量使得文本颜色可以在完成目标时变化。

为 Win?/win! 文本添加颜色，并使得其可以在赢得游戏时变为金色。为几个按钮设置合适的间距。阅读已有的代码会很有帮助，参考声明式 UI 的编写方法及相关文档。

3. 添加 New Game 按钮并完成组件事件的处理逻辑，同时使用条件渲染切换按钮的显示状态。

使用条件渲染，当赢得游戏时隐藏 push ， pop ， reverse 按钮，转而显示 New Game 按钮。处理 New Game 按钮的点击事件，生成一个新的可玩儿的 Goal，你可以通过随机进行 push ，pop ， reverse 三个操作来生成合法可玩儿的 Goal。参考条件渲染相关内容：if/else：条件渲染-渲染控制，参考事件处理相关文档。

实验二：应用 Ability 模型

4. 实验目的

学习如何响应来自组件的事件。

学习 ArkTS 路由与页面的运作方法。

学习 UIAbility 模型的运行与数据交互。

5. 实验内容

1. 显示并通过存储 API 保存最高分。

2. 为应用添加其他两个页面：欢迎页面与关于页面，在关于页面添加作者信息。

3. 使用 Dialog 组件实现退出时进行弹窗提醒。

6. 实验指南

预览器 无法完成实验二任务，你需要使用模拟器进行验证。

1. 显示并通过存储 API 保存最高分。

每次点击 New Game 即获得一分。你需要新建一个变量存储最高分，要求在程序退出并重新启动后正常保存最高分（重新安装后会消失）。

参考 PersistentStorage：持久化存储 UI 状态-管理应用拥有的状态-状态管理。布局概述-开发布局-基于 ArkTS 的声明式开发范式

2. 为应用添加其他两个页面：欢迎页面与关于页面，在关于页面添加作者信息。

你需要添加页面 Welcome 与 About 。你需要右键点击“pages”文件夹，选择“New > Page”，此时无需手动配置相关页面路由。否则请参考手动配置路由的方法。

Welcome ：从 Welcome 页面可以进入到主游戏界面 Index 与 About 。你需要使用按钮以及路由 API 完成这个功能。

About ：在该页面添加你的作者信息，使用图像 Image 标签引入你喜欢的头像。由于默认的启动页面是 Index ，你需要修改入口 Ability 的代码使其启动时进入 Welcome 页面。

参考：页面路由（router）-设置页面路由和组件导航

显示图片（Image）-显示图形

3. 使用 Dialog 组件实现退出时进行弹窗提醒。

参考：Dialog 组件的 API

UIAbility 组件与 UI 的数据同步-UIAbility 组件

UIAbilityContext-application-terminateSelf

7. 实验过程

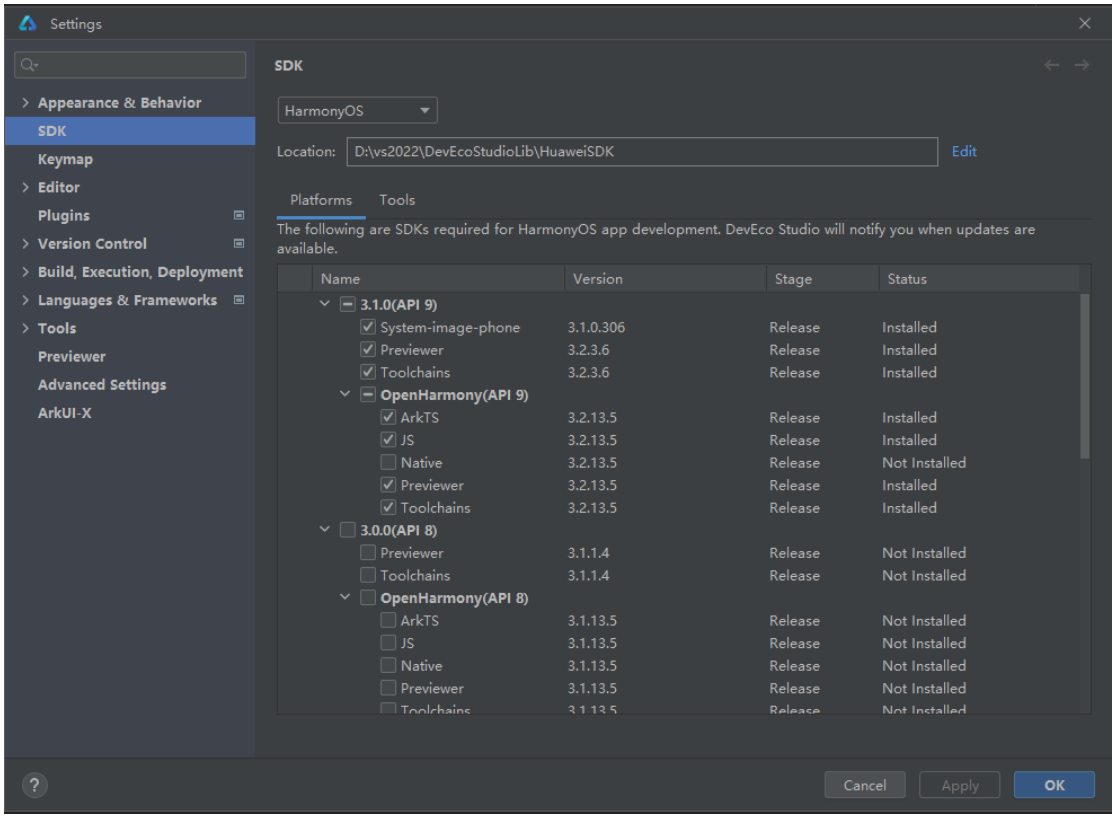
实验环境

安装鸿蒙 devecostudio。点击链接前往官网，找到版本 DevEco Studio 3.1.1 Release 并下载： SDK 下载和升级 | HarmonyOS 开发者。

点击安装后，程序会进行引导，按照提示安装开发需要的工具：Node.js 16，ohpm，HarmonyOS SDK。

安装后的工具链可以在 IDE 设置界面中查看，当 node.js, ohpm, SDK 安装完毕后，我们可以从鸿蒙官网拉去示例项目观察能否正常编译运行，并在预览区域显示来验证是否配置成功。

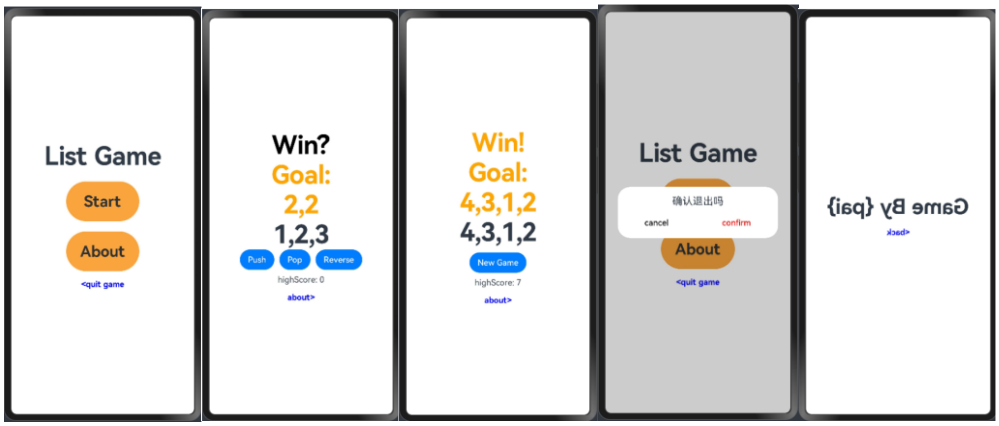
除了基本的开发工具链，我们还需要安装模拟器：单击 File > Settings > SDK（macOS 系统为 DevEco Studio > Preferences > SDK），下拉框选 择 HarmonyOS，勾选并下载 Platforms 下的 System-image 和 Tools 下的 Emulator 资源。



实验步骤与结果

实验结果：

本人实验结果已上传至在线仓库：[thethepai/OpenHarmonyArkTSPushPopGame: 华中科技大学鸿蒙移动应用技术实验 (github.com)](https://github.com/thethepai/OpenHarmonyArkTSPushPopGame)：



实验步骤：

1. 参考[构建第一个 ArkTS 应用]构建基础页面间的路由逻辑，完成各个功能页面的框架
2. 完成游戏功能，反转按钮和新游戏按钮的逻辑

use gameList's reverse function

```
Button('Reverse')
              .onClick(
                () => {
                  this.gameList.reverse()
                })
```

use Math.random() to generate new game list

```
Button('New Game')
              .onClick(
                () => {
                  ...
                  let test = Math.random()
                  ...
                })
```

3. 参考 [PersistentStorage：持久化存储 UI 状态-管理应用拥有的状态-状态管理-学习 ArkTS 语言-入门 | 华为开发者联盟 (huawei.com)]使用持久化存储保存最高分

4. 参考 UIAbility 组件与 UI 的数据同步-UIAbility 组件-Stage 模型应用组件-Stage 模型开发指导-应用模型-开发 | 华为开发者联盟 (huawei.com)完成退出确认弹窗功能

define dialog window

```
@CustomDialog
struct CustomDialogExample {
  controller: CustomDialogController
  cancel: () => void
  confirm: () => void
  ...
}
```

complete call back function in @Entry @Component

```
dialogController: CustomDialogController = new CustomDialogController({
    builder: CustomDialogExample({
      cancel: this.onCancel,
      confirm: this.onAccept,
    }),
  })

  onCancel() {
    ...
  }

  onAccept() {
    ...
  }
```

实验一源代码：

1. 完成单页 page 中声明式 UI 的编写。

游戏核心功能按钮组件，简单使用了 onclick 方法与一些间距控制：

```
TypeScript
if (!this.isWin()) {
  Row() {
    Button('Push')
      .onClick(
        () => {
          this.gameList.push(this.gameList.length + 1)
        })
      .margin({
        right: 10
      })
    Button('Pop')
      .onClick(
        () => {
          this.gameList.pop()
        })
```

```
      .margin({
        right: 10
      })
    Button('Reverse')
      .onClick(
        () => {
          this.gameList.reverse()
        })
  }
}
```

2. 使用样式 API 修改文本颜色与按钮间距，并使用响应式变量使得文本颜色可以在完成目标时变化。

通过三目运算符来使得文本颜色根据游戏目标是否完成来变化：

```
TypeScript
Text(this.isWin() ? 'Win!' : 'Win?')
  .fontSize(50)
  .fontWeight(FontWeight.Bold)
  .fontColor(this.isWin() ? Color.Orange : Color.Black)
```

3. 添加”New Game“按钮并完成组件事件的处理逻辑，同时使用条件渲染切换按钮的显示状态。

用 if 进行条件渲染的控制，在这里我选择使用自带的 Math 工具类来进行新游戏的随机，Math 不需要导入任何包可以直接使用较为方便。

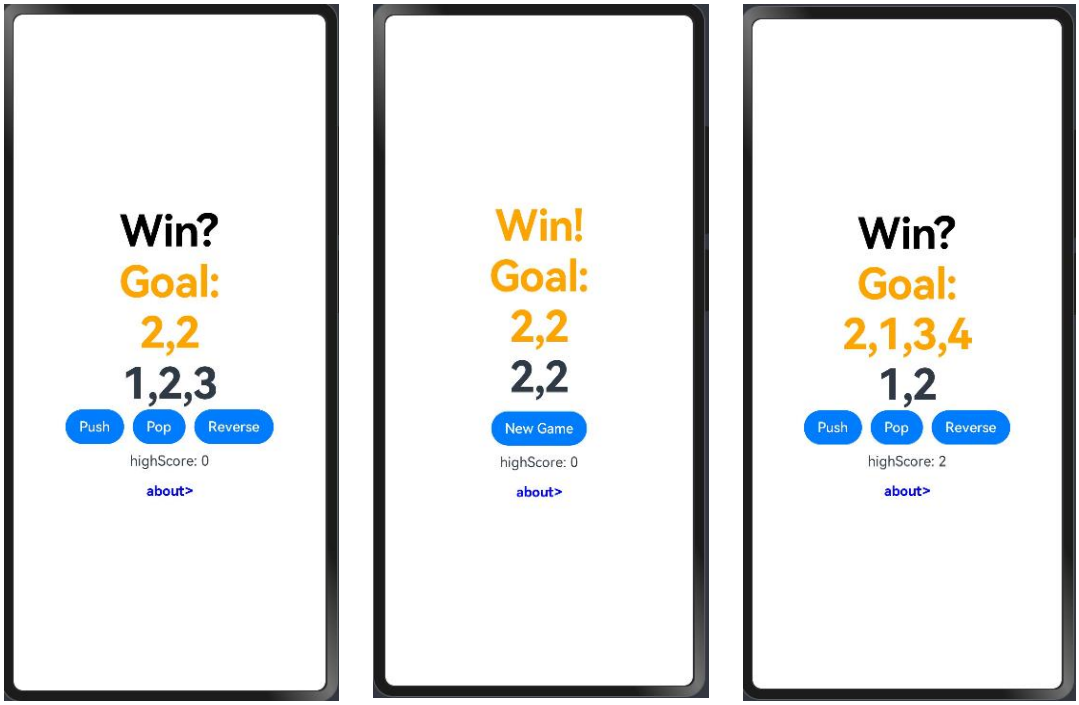
为了保证生成新游戏是合法的，直接进行若干次 push，pop，reverse 操作即可，为了保证能生成较长的序列，稍微调高了 push 的概率：

```
TypeScript
if (this.isWin()) {
  Row() {
    Button('New Game')
      .onClick(
        () => {
          // high score:
          this.highScore ++

          for (let count = 0; count < 12; count++) {
            this.gameGoal.pop()
          }
          do {
            for (let count = 0; count < 14; count++) {
```

```
              let test = Math.random()
              if (test < 0.35) {
                this.gameGoal.push(this.gameGoal.length + 1)
              } else if (test > 0.75) {
                this.gameGoal.pop()
              } else {
                this.gameGoal.reverse()
              }
            }
          } while (this.gameGoal.length == 0)
        })
      .margin({
        top: 10
      })
  }
}
```

实验一截图：



实验二源代码：

1. 显示并通过存储 API 保存最高分。

首先我们需要声明这样一个持久化变量，这样当应用打开时，会先检索是否已经保存了该持久化值，如果没有则创建这样一个持久化值，下一次打开时检索到已经创建过的值就可以直接使用了，由此实现持久化。

接着我们需要在 `struct index` 中获得这个变量，我们需要使用一个`@StorageLink` 语句来获得这个持久化变量，这样我们就可以像普通变量一样使用它：

```
TypeScript
PersistentStorage.PersistProp('highScore', 0);
...
@Entry
@Component
struct index{
@StorageLink('highScore') highScore: number =
AppStorage.Get('highScore')
...
}
```

2. 为应用添加其他两个页面：欢迎页面与关于页面，在关于页面添加作者信息。

首先需要在 `main_pages.json` 中添加页面路由表：

```
TypeScript
{
  "src": [
    "pages/Index",
    "pages/about",
    "pages/Welcome"
  ]
}
```

接着在 `EntryAbility.ts` 中修改进入页面：

```
TypeScript
windowStage.loadContent('pages/Welcome', (err, data) => {
  if (err.code) {
    hilog.error(0x0000, 'testTag', 'Failed to load the content.
Cause: %{public}s', JSON.stringify(err) ?? '');
    return;
  }
  hilog.info(0x0000, 'testTag', 'Succeeded in loading the content.
Data: %{public}s', JSON.stringify(data) ?? '');
});
```

接着我们就可以添加页面文件并且写入想要的内容了，页面之间的跳转按钮逻辑如下：

```
TypeScript
...
.onClick(() => {
  console.info(`Succeeded in clicking the 'Next' button.`)
  // 跳转到第二页
  router.pushUrl({ url: 'pages/about' }).then(() => {
    console.info('Succeeded in jumping to the second page.')
  }).catch((err) => {
    console.error(`Failed to jump to the second page.Code is
${err.code}, message is ${err.message}`)
  })
})
```

3. 使用 Dialog 组件实现退出时进行弹窗提醒。

首先在 welcome 页面声明弹窗组件：

```
TypeScript
@CustomDialog
struct CustomDialogExample {
  controller: CustomDialogController
  cancel: () => void
  confirm: () => void

  build() {
    Column() {
      Text('确认退出吗').fontSize(20).margin({ top: 16, bottom:
10 })
      Flex({ justifyContent: FlexAlign.SpaceAround }) {
        Button('cancel')
          .onClick(() => {
            this.controller.close()
            this.cancel()
          })
          .backgroundColor(0xffffff).fontColor(Color.Black)
        Button('confirm')
          .onClick(() => {
            this.controller.close()
            this.confirm()
          })
          .backgroundColor(0xffffff).fontColor(Color.Red)
```

```
      }
      .margin({
        bottom: 10
      })
    }
  }
}
```

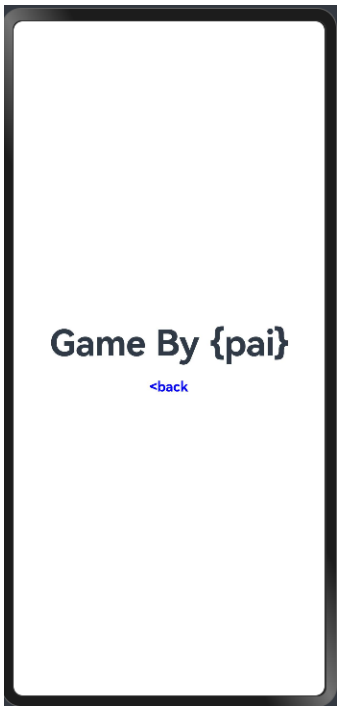
完善其中按钮的功能逻辑，使用 `CustomDialogController` 对象，接着定义 `onCancel` 和 `onAccept` 函数。在 `onAccept` 函数中，使用 `this.context.terminateSelf` 完成程序的退出，注意，使用这个工具类需要在文件头导入 `common` 包。在 `onBackPress` 函数中打开我们刚刚编写的自定义 dialog，注意返回值需要是 true，才能执行用户自定义的逻辑。

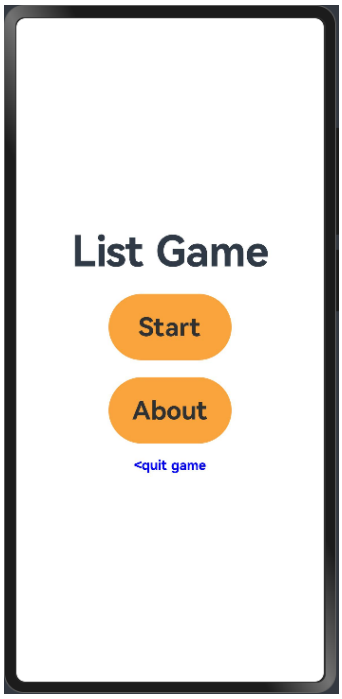
```
TypeScript
import common from '@ohos.app.ability.common';
...
dialogController: CustomDialogController = new
CustomDialogController({
  builder: CustomDialogExample({
    cancel: this.onCancel,
    confirm: this.onAccept,
  }),
  alignment: DialogAlignment.Default, // 可设置 dialog 的对齐方式，设定显示在底部或中间等，默认为底部显示
})
...
onCancel() {
  console.info('Callback when the first button is clicked')
}

onAccept() {
  console.info('Quit App')
  this.context.terminateSelf()
}

onBackPress() {
  console.info('onBackPress')
  this.dialogController.open()
  // 必须要 return true
  return true
}
```

实验二截图：


Game By {pai}
<back


List Game
Start
About
<quit game


List Game
确认退出吗
cancel
confirm
About
<quit game

8. 课程小作业-鸿蒙系统之我见

华为开发鸿蒙系统的初衷是面对美国制裁和谷歌服务禁用的挑战，实现中国科技产业自主可控。鸿蒙系统诞生至今，一直饱受“换壳安卓”的舆论困扰。请以本课程学习的知识为基础，书写一篇小作业，字数不限。描述你对鸿蒙系统的深入见解。

作业推荐包括以下内容：

（1）背景：鸿蒙是否重要？换壳安卓？卡脖子的利器；

（2）现状：评价移动操作系统的关键指标，现有安卓和 ios 系统表现如何？

（3）挑战：当前移动操作系统有哪些不足的地方？

（4）鸿蒙核心技术：新特性，原理

（5）未来：终端操作系统的基座，未来的希望，越来越多的设备上运行

小作业评分维度：

（1）客观性：保持中立，基于事实和数据进行分析。

（2）深度：深入探讨技术细节，提供专业的见解。

（3）广度：覆盖系统的多方面影响，包括技术、市场和社会层面。

（4）创新性：鼓励提出独到的见解和前瞻性的思考。

课程作业：

鸿蒙系统：超越“换壳安卓”的自主探索之旅

背景：鸿蒙的重要性与舆论困境

在中美技术竞争加剧、国际环境不确定性增大的背景下，华为开发鸿蒙系统不仅是对美国制裁和谷歌服务禁用挑战的直接回应，更是中国科技产业追求自主可控战略目标的重要一步。鸿蒙系统的诞生，旨在构建一套能够跨平台运行的操作系统，打破操作系统领域的垄断格局，实现从手机到物联网设备的无缝连接。尽管“换壳安卓”的标签一度成为舆论焦点，但这并未全面反映鸿蒙系统的创新价值和长远布局。

现状：移动操作系统的评价指标与市场表现

评价移动操作系统通常涉及性能、安全性、生态兼容性、用户体验及开发者支持等多个维度。当前，Android 凭借其开放性和广泛的设备覆盖占据市场主导地位，而 iOS 则以其封闭的生态系统、高度优化的用户体验和强大的安全性能著称。两者在应用生态丰富度、系统稳定性方面表现卓越，但同时也面临着隐私保护、系统碎片化（Android 尤为显著）、以及对新兴技术（如 AI、物联网）整合能力的持续挑战。

挑战：移动操作系统面临的不足

1. 生态壁垒：建立一个健康、活跃的应用开发者生态，对于任何操作系统都是巨大的挑战，尤其是打破现有两大生态的用户习惯和开发者忠诚度。

2. 安全性与隐私：随着数据泄露事件频发，如何在保障用户体验的同时加强数据保护，成为操作系统亟需解决的问题。

3. 系统碎片化与更新不均：Android 系统因设备多样性导致的碎片化问题，影响了用户体验和安全更新的普及率。

4. 跨平台互联：随着物联网的发展，操作系统需要更好地支持不同设备间的互联互通，这要求更高的灵活性和兼容性。

鸿蒙核心技术：新特性与原理

鸿蒙的核心竞争力在于其分布式架构，该架构设计使得系统能够根据不同的硬件资源灵活部署，实现“一次开发，多端部署”。这意味着开发者编写的应用可以在手机、平板、智能穿戴设备甚至智能家居产品上无缝运行，极大地提升了用户体验的一致性和便捷性。此外，鸿蒙强调微内核设计，相较于传统的宏内核，它能提供更强的安全性和更低的资源消耗，尤其适合物联网设备。

未来：鸿蒙的无限可能

鸿蒙不仅仅定位为手机操作系统，而是作为未来万物互联时代的终端操作系统基座，承载着连接百亿级智能设备的愿景。随着鸿蒙生态的不断成熟，我们有望看到更多基于鸿蒙系统的智能设备出现，不仅限于消费电子领域，还将拓展到工业、交通、医疗等更广泛的行业应用中。鸿蒙系统通过不断的技术迭代和生态建设，正逐步摆脱“换壳安卓”的标签，展示其在跨平台协同、低延迟通信、超级终端等领域的创新优势，为中国乃至全球的科技自立自强贡献力量。

结论

鸿蒙系统的开发不仅是对当前操作系统市场格局的挑战，更是对未来技术趋势的前瞻性布局。虽然前路不乏挑战，但其分布式设计、微内核架构以及对物联网时代的深刻理解，为鸿蒙开辟了一条独特的发展路径。随着技术的不断演进和生态的日益完善，鸿蒙有潜力成为推动全球数字生态迈向更加开放、互联、安全的新一代操作系统。