HUAWEI

< HDC.Together >

HUAWEI DEVELOPER CONFERENCE 2021

Copyright © Huawei Device Co., Ltd. 2021. All right reserved.
Redistribution or public display not permitted without written permission from Huawei.

HarmonyOS 3.0开发者预览版
全新发布

HarmonyOS 3.0开发者预览版

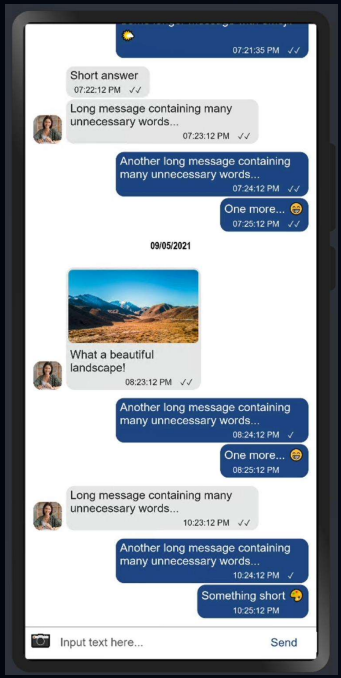
总计6000+能力集，包括新一代声明式UI开发框架，JS后台服务/多线程/远程调用，WebGL，窗口以及各类图片媒体数据库等能力

子系统	能力集说明
UI编程框架	**提供新一代声明式UI开发范式，支持Typescript，链式调用，装饰器，条件渲染，多态组件，自定义组件，多维状态管理等**
用户程序框架	应用信息获取，安装、卸载，系统和窗口状态获取
图形图像	WebGL能力，屏幕信息，窗口信息
软总线基础通讯	基于JS的RPC通信能力
媒体	图片解码， pixelmap/audio/recorder能力
元能力	JS PA开发能力，迁移能力，卡片能力
分布式数据管理	RDB，KVStore数据库能力
全球化	时区，语言获取
语言基础库/其它	多线程机制，Parcel， URL，上传下载，文件基础库，分布式设备列表获取，系统以及应用账号管理等

新一代
UI编程框架

- UI编程框架概述
- HarmonyOS UI编程框架演进
- 关键技术以及典型示例
- 下一步

UI编程框架概述



用户视角：
- 视觉
- 交互
- 体验

开发者视角：
- 编程语言
- 开发界面
- 业务逻辑

UI编程框架

系统视角：
- 运行环境
- 图形显示

永恒的需求

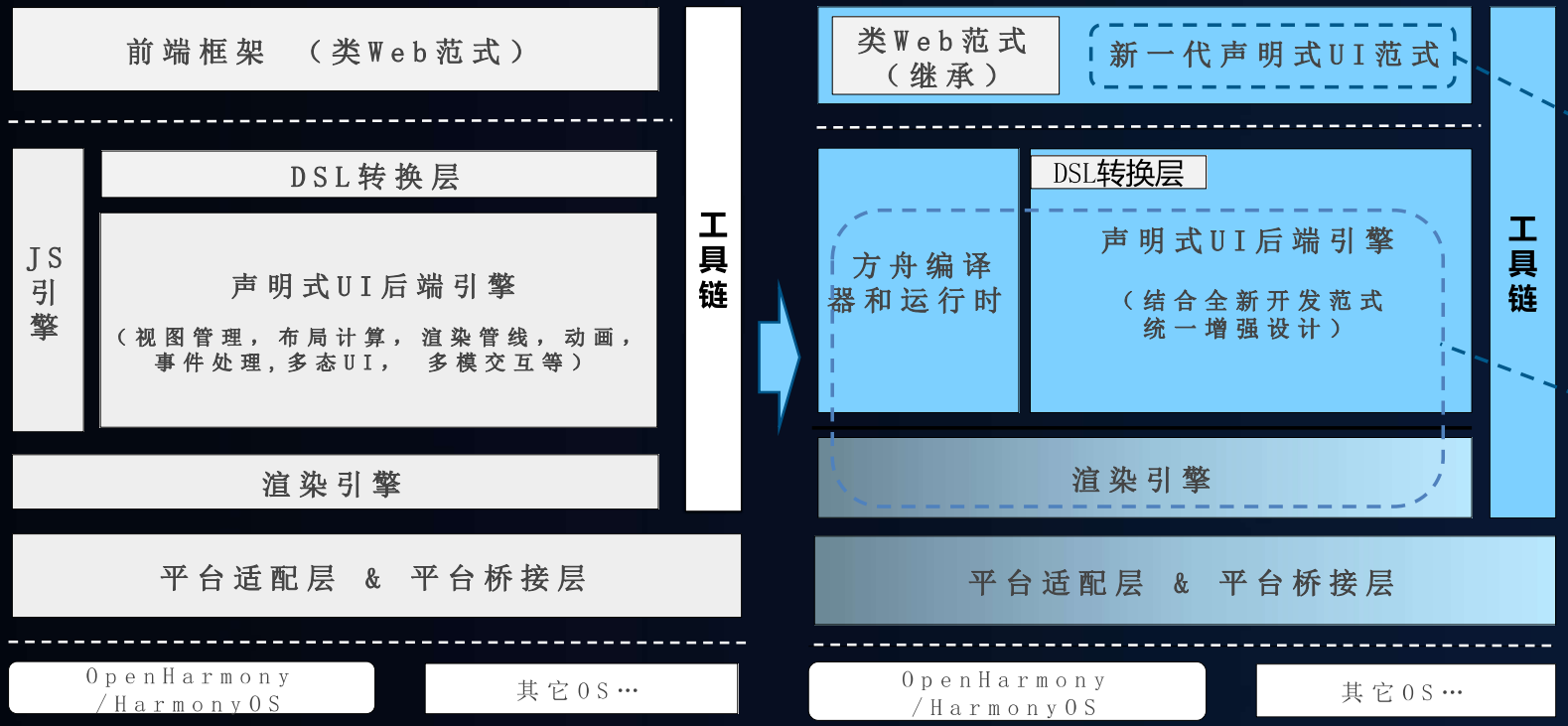
- **开发效率**：代码量、学习曲线、工具、社区、三方库 …
- **性能体验**：启动速度、帧率、响应时延、酷炫动画、资源占用 …

超级终端下的需求

- **设备形态差异**：屏幕形状、尺寸、分辨率，交互模式
- 设备能力差异：内存、CPU、GPU

综合UI渲染&
语言&运行时
构建新一代UI编程框架

围绕极简开发、高性能、跨设备跨平台演进



开发效率

- UI和描述的自然映射
- 所见即所得，实时预览

性能体验

- 启动秒开
- 高帧率渲染

类Web范式

```
/* mycomponent.css */
.column {
  flex-direction: column;
  justify-content: center;
}
.title {
  font-size: 36px;
  color: "red";
}
```

```
<!--mycomponent.hml-->
<div class="column">
  <text class="title">
    Hello HarmonyOS
  </text>
</div>
```

```
<!--mycomponent2.hml-->
<div class="container">
  <canvas id="canvas"></canvas>
</div>
```

```
export default {
  onInit() {
    var canvas =
      this.$element("canvas")
    var context =
      canvas.getContext("2d")
    context.beginPath()
    context.lineTo(10, 0)
    context.lineTo(10, 10)
    context.lineTo(0, 0)
    context.fillStyle = "red"
    context.fill()
  }
}
```

新一代声明式UI范式

```
struct MyComponent {
  build() {
    Column {
      Text("Hello HarmonyOS")
        .fontSize(36)
        .color(Color.red)
    }.center()
  }
}
```

```
struct MyComponent2 {
  build(){
    path()
      .commands('M0 0 L10 0 L10
                 10 L0 0 Z')
  }.fill(Color.red)
}
```

网格　列表　导航

...

UI逻辑（声明式UI范式）

类自然语言的UI描述和组合

极简语法
多态组件
开箱即用

应用逻辑

开放语言

状态管理
（组件内、组件间、跨设备）

基于TypeScript扩展，
增强声明式UI描述能力

声明式UI范式代码示例和概念介绍

一个“Hello World”的文本示例，当点击“Click me”按钮后，将显示“Hello ACE”

```
@Entry
@Component
struct Hello {
    @State myText: string = 'World'
    build(){
        Column(){
            Text('Hello')
                .fontSize(100)
            Text(this.myText)
                .fontSize(100)
            Divider()
            Button(){
                Text('Click me')
            }.onClick(() => {
                this.myText = 'ACE'
            })
            .width(500)
            .height(200)
            .color(Color.Red)
        }
    }
}
```

声明式范式基本概念

装饰器：用来装饰类、结构体、方法以及变量，赋予其特殊的含义，如上述示例中@Entry、@Component、@State都是装饰器

自定义组件：可复用的UI单元，可组合其它组件，如上述被@Component装饰的struct Hello

UI描述：声明式的方式来描述UI的结构，如上述build()方法内部的代码块

内置组件：框架中默认内置的基础和布局组件，可直接被开发者调用，如Column、Text、Divider、Button

属性方法：用于组件属性的配置，统一通过属性方法进行设置，如fontSize()、width()、height()、color()等

事件方法：用于添加组件对事件的响应逻辑，统一通过事件方法进行设置，如跟随在Button后面的onClick()

多维状态管理-组件间，多层组件，全局，跨设备

整体渲染流程以及关键技术概览

关键技术1：
类自然语言描述的声明式UI编程范式

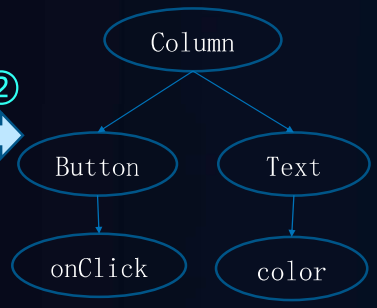
```
@Entry
@Component
struct MyComponent {
  @State count: number = 0
  build() {
    Column() {
      Button('Click me')
        .onClick(() => {
          this.count += 1
        })
      Text(`Click times ${this.count}`)
        .color(Color.Red)
    }
  }
}
```

源代码

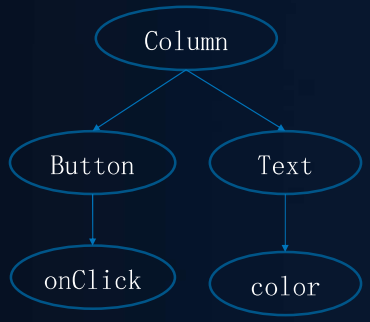
①

带有变量类型标识的目标文件

②



③



关键技术2：
TS语言运行时和跨语言（TS->C++）调用低开销设计

方舟强类型语言运行时

Component树

⑦

Element树

⑨

@State装饰器

setter
getter

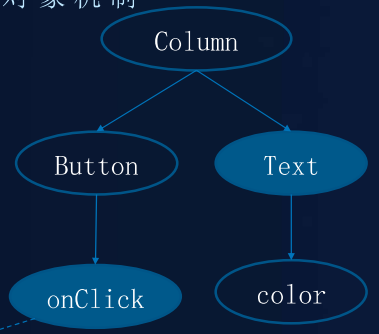
⑧

响应式状态管理

关键技术3：
扁平化UI渲染管线，轻量化对象机制

④

⑩



Render树

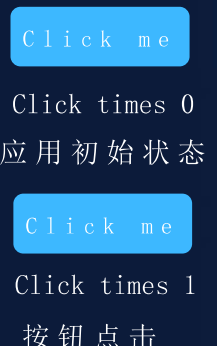
关键技术4：
数据驱动视图自动&高效更新的状态管理机制

⑥

点击事件输入

⑤

⑪



应用初始流程：
①②③④⑤

按钮点击流程：
⑥⑦⑧⑨⑩⑪

编译 | 运行构建或视图更新 | 布局和渲染 | 显示

懒加载机制，
提升页面加载速度，减少UI重建时间

- 数据按需加载
- UI界面动态创建
- 数据更新通知

一个综合应用示例 - 健康生活

绘制和动效组件

```
Shape() {
  Path().commands(this.pathCommands1)
  Path().commands(this.pathCommands2)
  Path().commands(this.pathCommands3)
}
Animator('logo')
  .state(this.stateValue)
  .onFinish(() => { setTimeout(() => { router.replace({ uri: "pages/FoodCategoryList" })}, 1000) })
  .onFrame((value: number) => {
    this.opacityValue = this.curve1.interpolate(value)
    this.scaleValue = this.curve1.interpolate(value)
  })
```

列表和网格布局

```
struct FoodList {
  private foodItems: FoodData[]
  build() {
      List() {
        ForEach(this.foodItems, item => {
          ListItem() {
            FoodListItem({ foodItem: item })
          }
        }, item => item.id.toString())
      }
  }
}
```

```
@Component
struct FoodGrid {
  private foodItems: FoodData[]
  build() {
    Grid() {
      ForEach(this.foodItems, (item: FoodData) => {
        GridItem() {
          FoodGridItem({ foodItem: item })
        }
      }, (item: FoodData) => item.id.toString())
    }
    .rowsTemplate(this.gridRowTemplate)
    .columnsTemplate('1fr 1fr')
  }
}
```

基础布局

```
@Component
struct FoodDetail {
  private foodItem: FoodData = router.getParams().foodId
  build() {
    Column() {
      Stack({ alignContent: Alignment.TopStart }) {
        FoodImageDisplay({ foodItem: this.foodItem })
      }
      Swiper() {
        ContentTable({ foodItem: this.foodItem })
        CaloryProgress({ foodItem: this.foodItem })
      }
      Button('Record',  { type: ButtonType.Capsule, stateEffect: true })
    }
  }
}
```

高级组件示例 - 图表


```
LineChart({
    chartHeight: this.getChartHeight(),
    segmentWidth: this.getSegmentWidth(),
    focusedIndex: this.$focusedIndex,
    data: this.data,
    color: this.color
})
```

```
@Component
export struct LineChart{
  @Link focusedIndex: number
  data: number[]
  build(){
    Stack() {
      Shape() {
        Path().commands(this.buildPath()).stroke(this.color)
      }
      if(this.focusedIndex !== undefined){
        Circle({width:15, height: 15}).fill(this.color).position(this.circlePosition)
          .animation({duration: 100, curve: Curve.EaseIn})
      }
    }
  }
}
```

高级组件示例 - 瀑布流布局


```
WaterfallGrid({
    columns: this.$columns,
    columnWidth: this.$columnWidth,
    items: this.$items
})
```

```
@Component
struct WaterfallGrid {
    @Watch("onItemChange")  @Link items: Item[]
    onItemChange(){
        this.calculatePositions()
    }
    build() {
        Column() {
            Scroll() {
                Stack() {
                    ForEach(this.items, (item: Item) => {
                        Stack() {
                            if (item.source) {
                                Image(item.source)
                            }
                            //...
                        }
                        .onClick(() => this.items = this.items.filter(el => el.id !== item.id))
                        .animation({ duration: 1000, curve: Curve.EaseInOut })
                    }, item => item.id)
                }.alignContent(Alignment.TopStart)
                .width("100%")
```

一次开发多端部署
通过多维度解决方案， 让多设备开发更简单

页面信息结构

内容类首页：
- 搜索+内容+[页签]
- 搜索+子页签+内容+[页签]

效率类首页：
- 标题+[子页签]+内容+[页签/工具栏]
- 标题+搜索+内容+[Fab]+[页签]

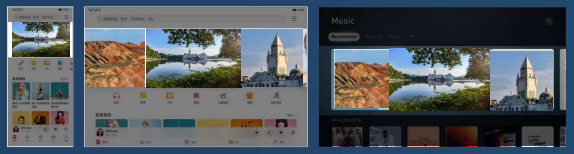
工具类首页：上图下文

多选页：Title+列表/宫格+Toolbar

页面组合模版

典型示例代码

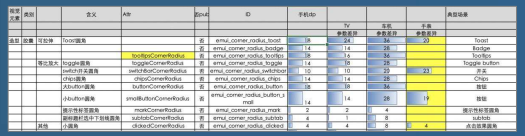
组合组件



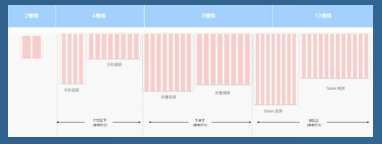
零部件：组件

多态控件 + 统一交互

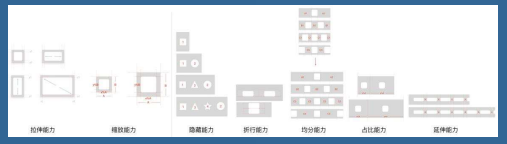
基本参数（色彩、字号、圆角、公共间隔）



栅格定义



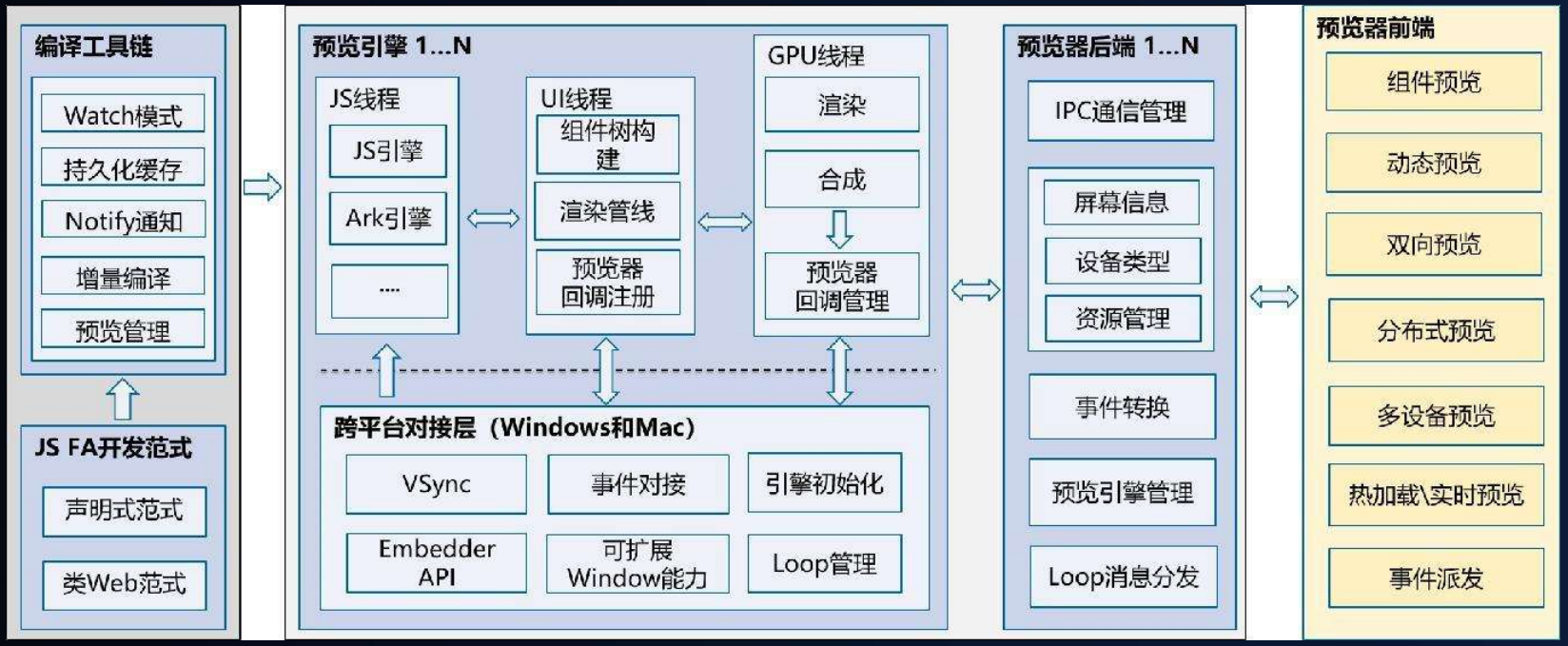
原子能力



基础能力

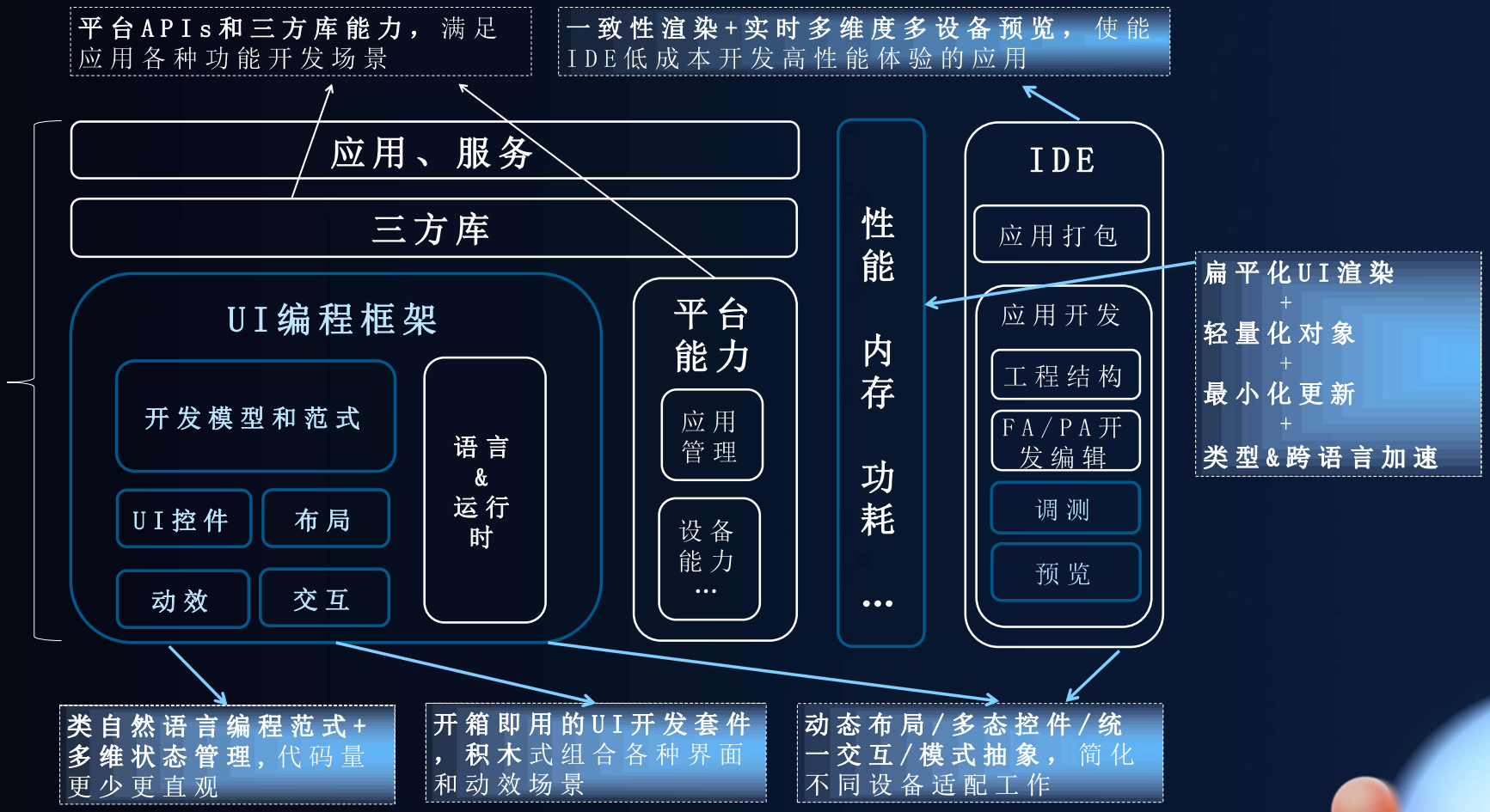
UI编程框架预览架构总览

一致性渲染，双向预览，多维度预览（页面级、组件级，多设备），实时编写显示…



UI编程框架在应用开发整体视图中的概览

应用开发整体视图



下一步

- 三方生态拓展（覆盖全场景设备）
  - 高级UI能力 - 地图、游戏等
  - Web能力
  - …
- 围绕跨设备、性能体验持续创新