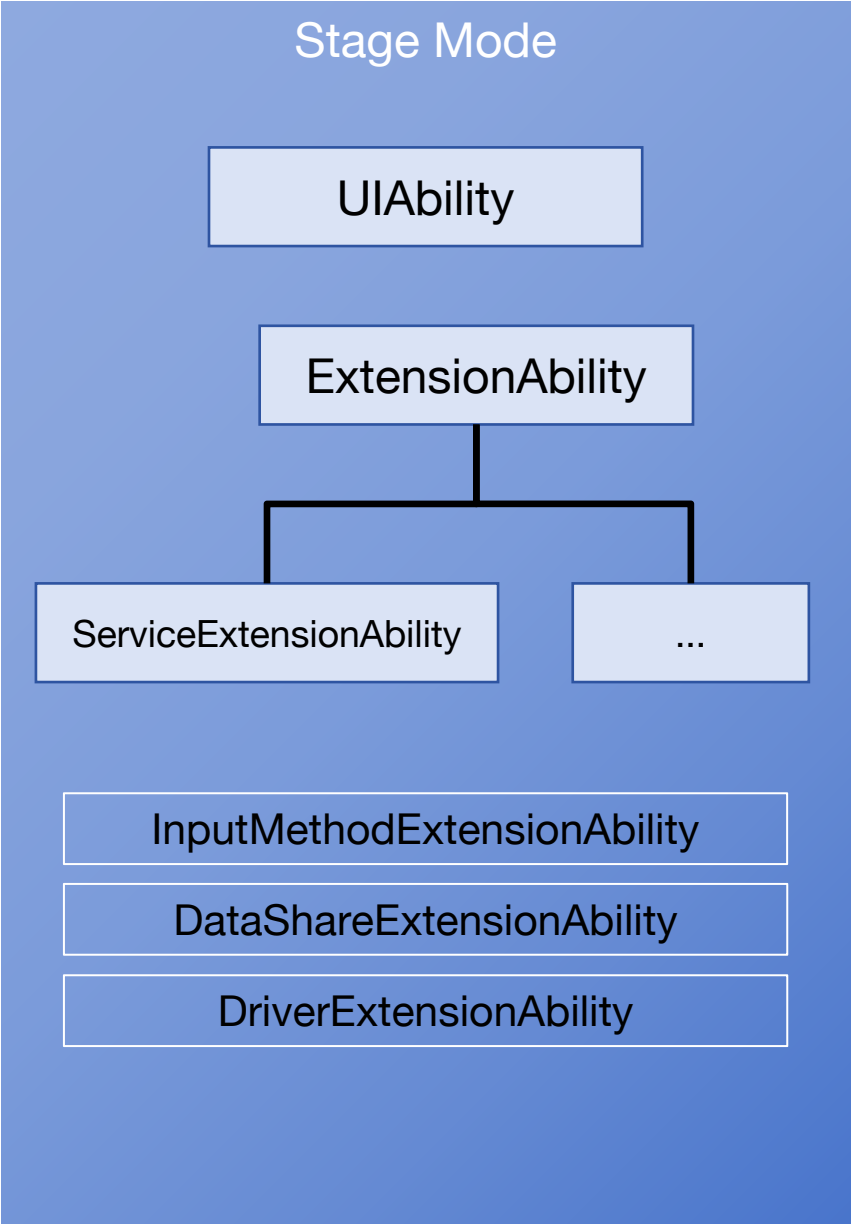


高性能鸿蒙应用开发

分享人：鸿蒙ArkUI框架技术专家 —— 王雷

鸿蒙应用构成

构成	Stage模型
应用组件	<div>1. 组件分类</div> <div>stage-model-component - UIAbility组件: 包含UI, 提供展示UI的能力, 主要用于和用户交互。详细介绍请参见UIAbility组件概述。</div> <div>- ExtensionAbility组件: 提供特定场景 (如卡片、输入法) 的扩展能力, 满足更多的使用场景。详细介绍请参见ExtensionAbility组件概述。</div> <div>2. 开发方式</div> <div>采用面向对象的方式, 将应用组件以类接口的形式开放给开发者, 可以进行派生, 利于扩展能力。</div>
进程模型	<div>1. 主进程</div> <div>2. ExtensionAbility进程</div> <div>3. 渲染进程</div>
线程模型	<div>1. ArkTS引擎实例的创建</div> <div>一个进程可以运行多个应用组件实例, 所有应用组件实例共享一个ArkTS引擎实例。</div> <div>2. 线程模型</div> <div>ArkTS引擎实例在主线程上创建。</div> <div>3. 进程内对象共享: 支持。</div>
任务管理模型	<div>每个UIAbility组件实例创建一个任务。</div> <div>- 任务会持久化存储, 直到超过最大任务个数 (根据产品配置自定义) 或者用户主动删除任务。</div> <div>- UIAbility组件之间不会形成栈的结构。</div>
配置文件	使用app.json5描述应用信息, module.json5描述HAP信息、应用组件信息。



性能影响纬度

- ◆ 语言纬度：ArkTS
- ◆ 线程模型：Worker & TaskPool
- ◆ UI构成：ArkUI框架
- ◆ 图形渲染：ArkGraphics渲染引擎
- ◆ 分析工具

ArkTS语言特性

高性能容器类库

线性容器	非线性容器
ArrayList	HashMap
Vector	HashSet
List	TreeMap
LinkedList	TreeSet
Deque	LightWeightMap
Queue	LightWeightSet
Stack	PlainArray

```
import ArrayList from '@ohos.util.XXX';
```

ArkTS语言使用过程中性能提升方法

- 热点循环中常量提取，减少属性访问次数
- 避免频繁使用delete
- 数值计算避免溢出
- 避免使用稀疏数组
- 使用字面量进行对象创建
- 避免动态添加属性
- 避免使用type类型标注
- 声明参数要和实际的参数一致
- 避免动态声明function与class

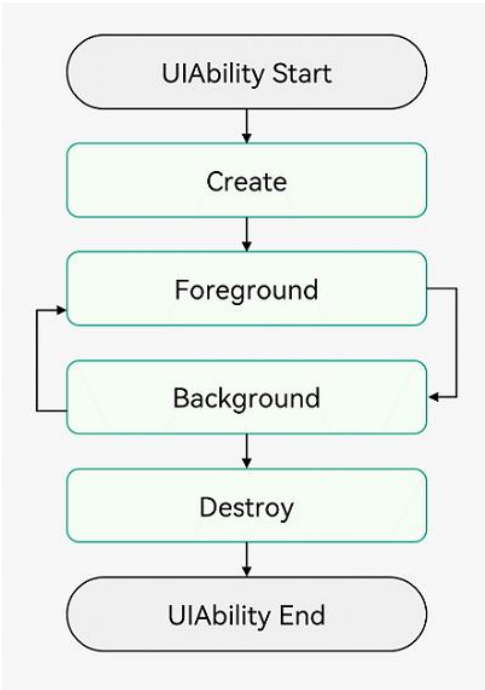
了解启动流程各阶段执行内容，合理配置应用结构，提升启动速度



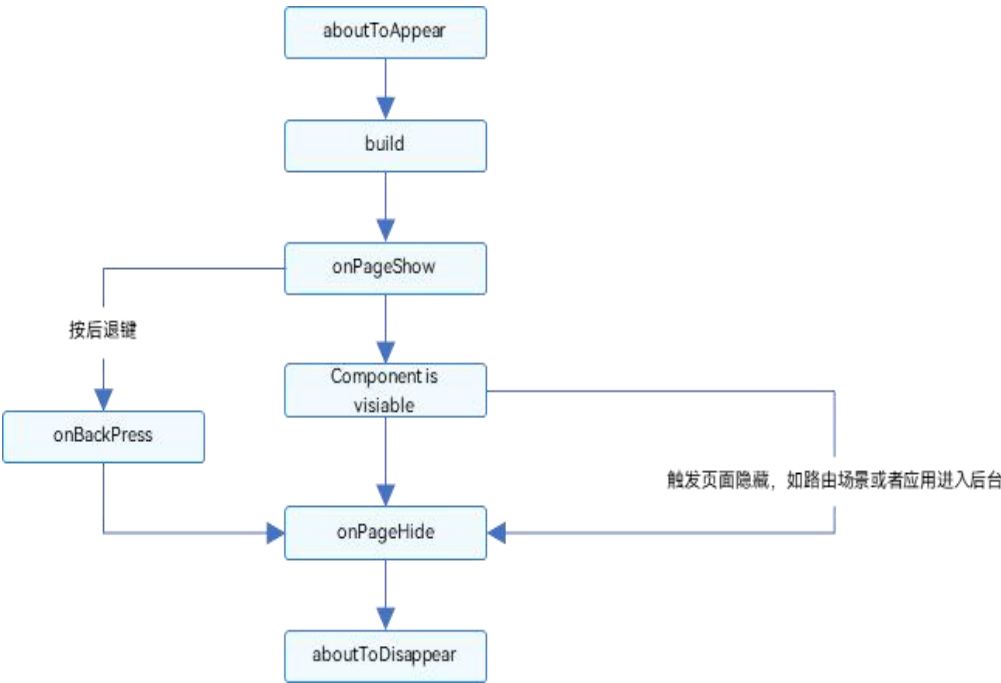
合理配置启动页面图标尺寸，减少解码开销

```
"abilities": [  
  {  
    "name": "EntryAbility",  
    "srcEntrance": "./ets/entryability/EntryAbility.ts",  
    "description": "$string:EntryAbility_desc",  
    "icon": "$media:icon",  
    "label": "$string:EntryAbility_label",  
    "startWindowIcon": "$media:startWindowIcon", // 在这里修改启动页图标，建议不要超过256像素x256像素  
    "startWindowBackground":  
      "$color:start_window_background",  
    "visible": true,  
    "skills": [  
      {  
        "entities": [  
          "entity.system.home"  
        ],  
        "actions": [  
          "action.system.home"  
        ]  
      }  
    ]  
  }  
]
```

按需引入模块，合理使用生命周期，避免耗时操作

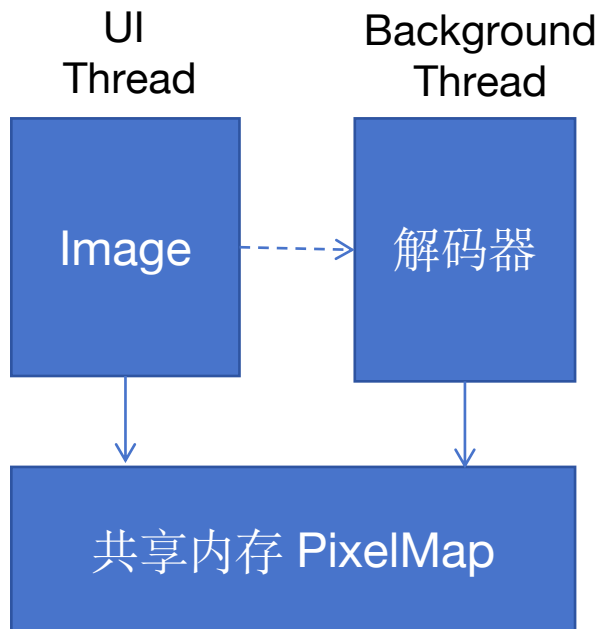


首页组件按需创建，页面生命周期避免耗时操作

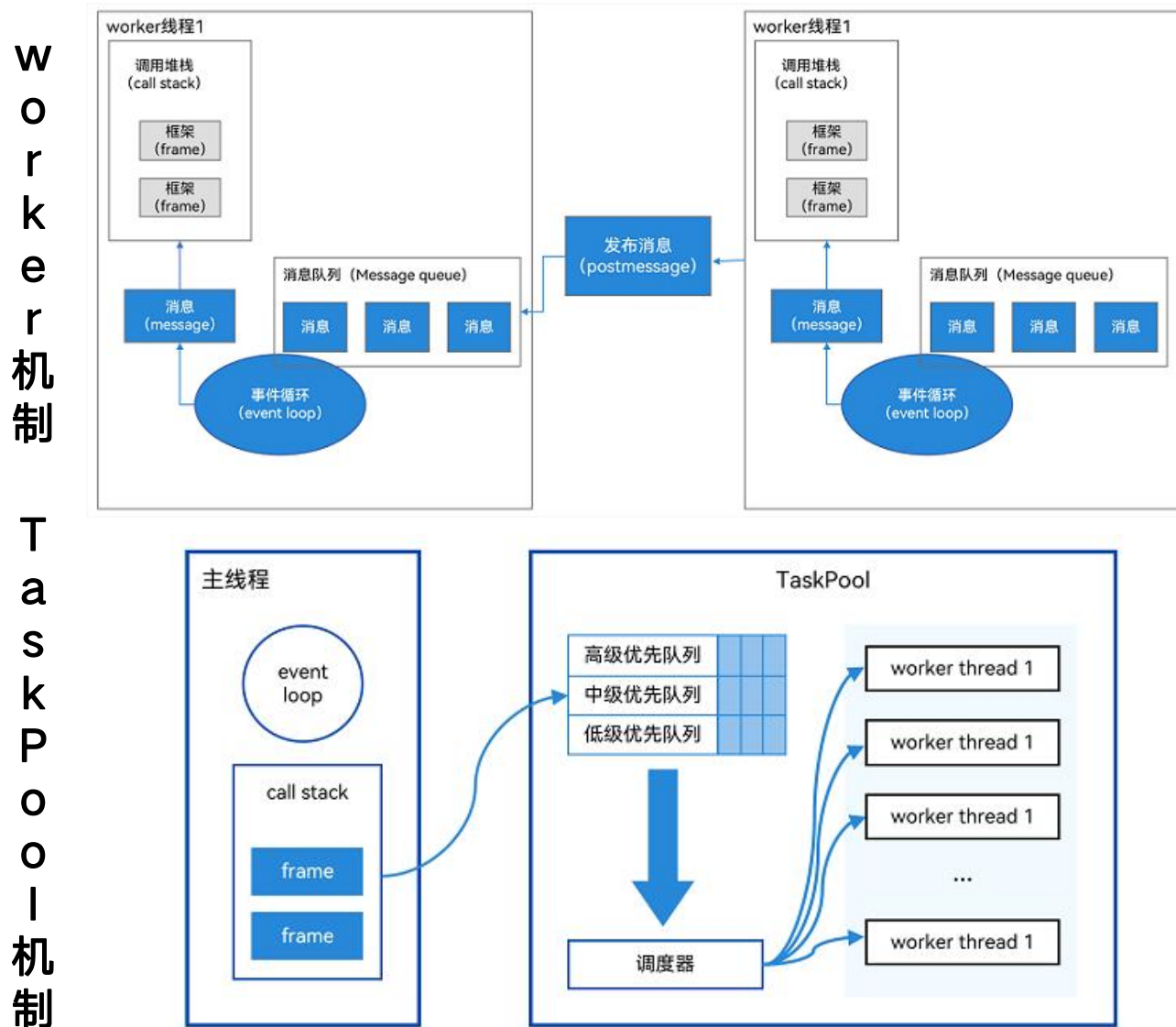


使用异步机制，降低UI线程同步操作负载

组件中的异步逻辑

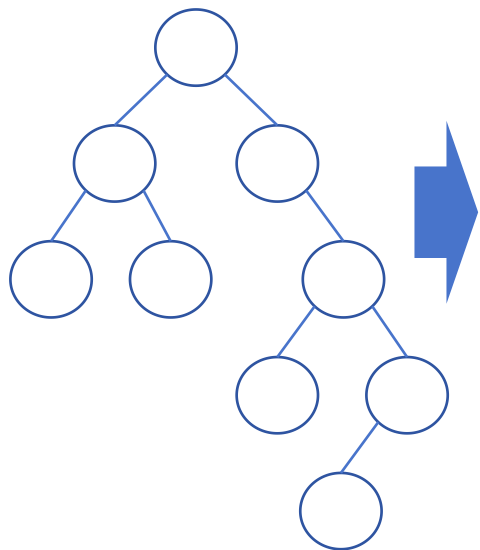


应用异步机制



UI框架运行流程与性能影响点

应用组件树



阶段1: 数据处理



脏节点
列表

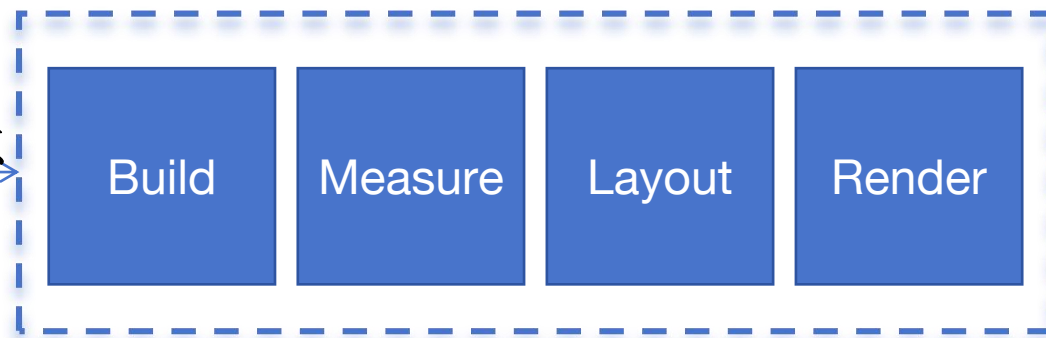
性能影响点:

- 状态数据更新数量
- 更新脏组件数量

开发约束:

- 布局约束
- 状态更新约束
- 懒加载更新约束

阶段2: UI更新



性能影响点:

- 组件创建数量
- 属性更新数量
- 布局更新范围
- 绘制指令数量

开发约束:

- 布局合理使用
- 组件复用, 避免重复创建

UI框架运行流程：数据处理流程

开发者视角

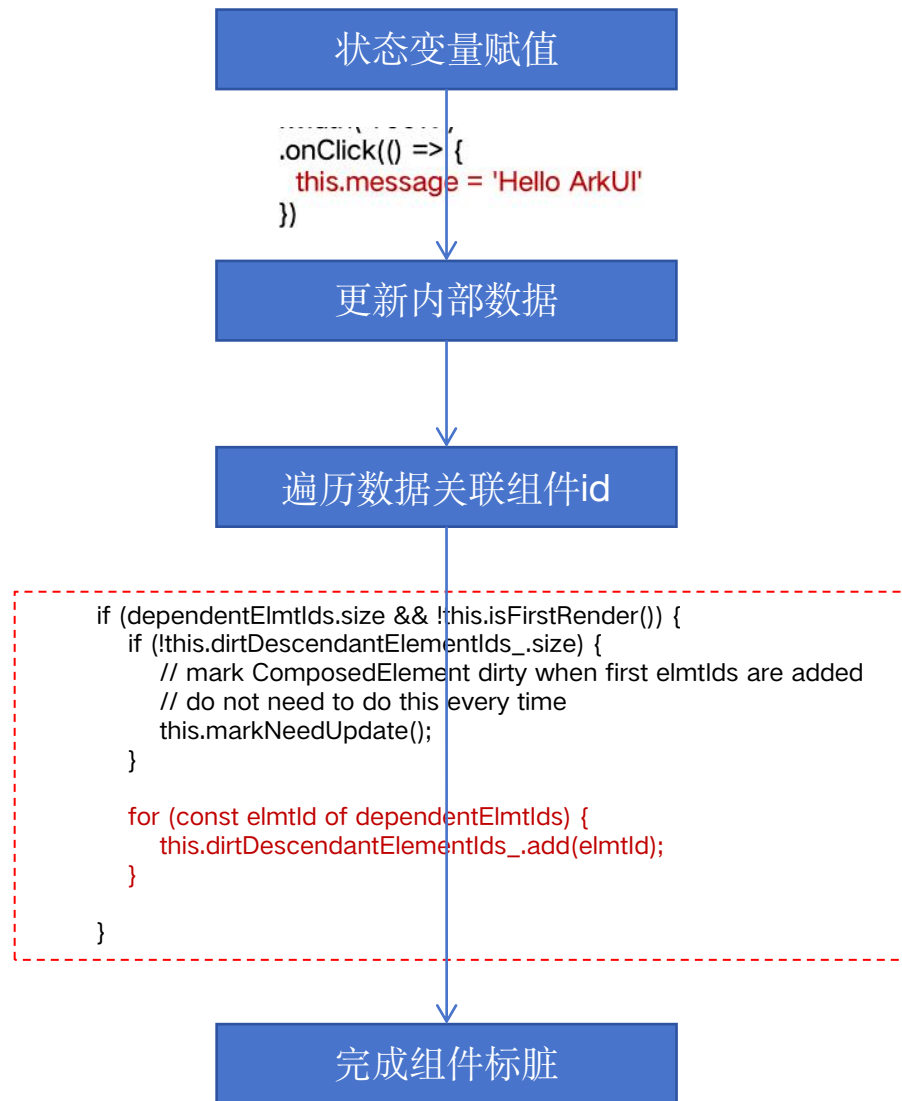
```
@Entry
@Component
struct Index {
  @State message: string = 'Hello World'

  build() {
    Row() {
      Column() {
        Text(this.message)
          .fontSize(50)
          .fontWeight(FontWeight.Bold)
      }
      .width('100%')
      .onClick(() => {
        this.message = 'Hello ArkUI'
      })
    }
    .height('100%')
  }
}
```

依赖收集过程

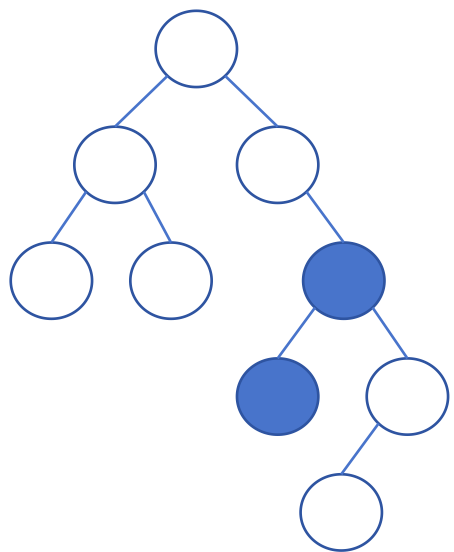


组件标脏过程



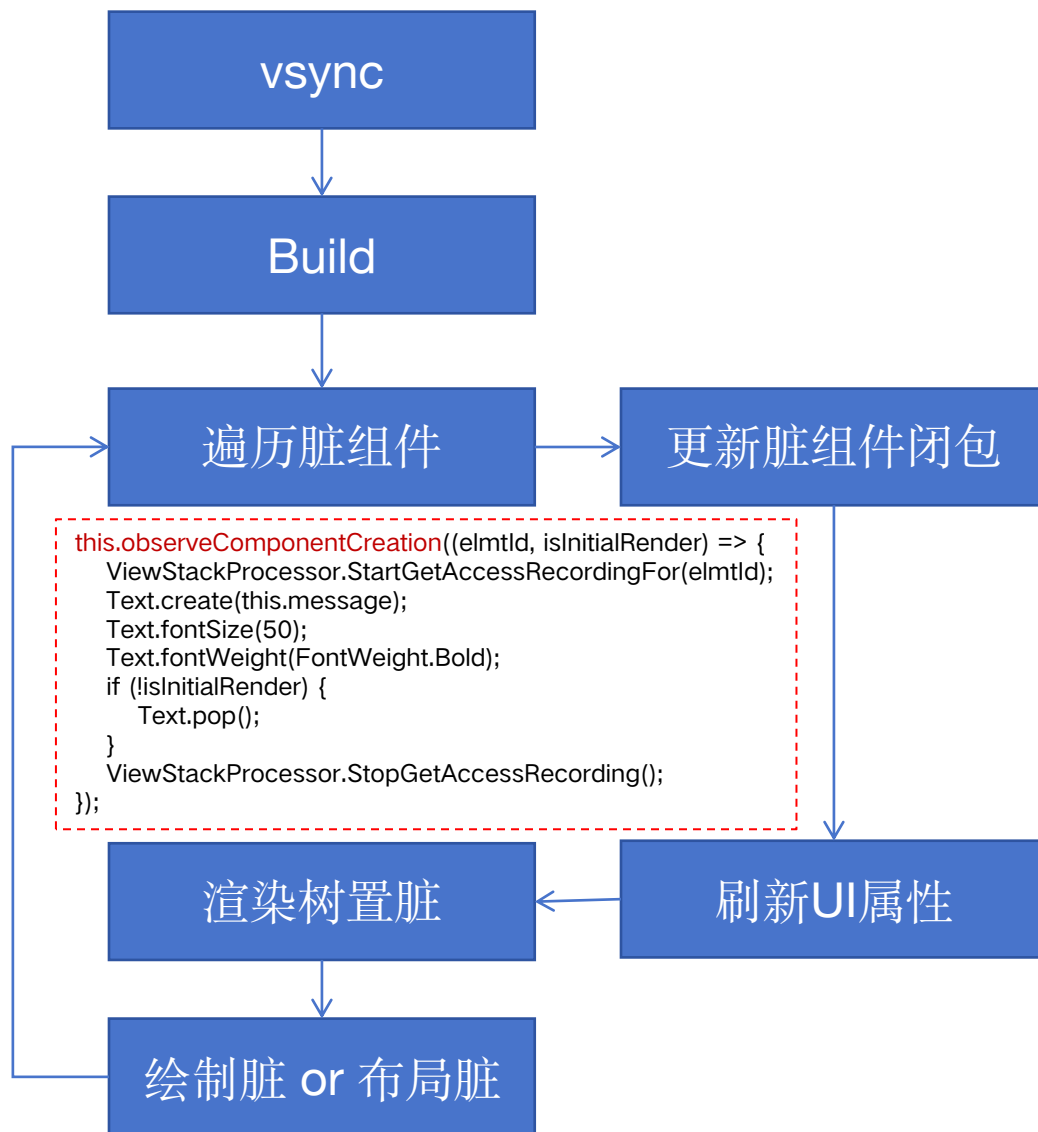
UI框架运行流程：UI更新流程

UITree

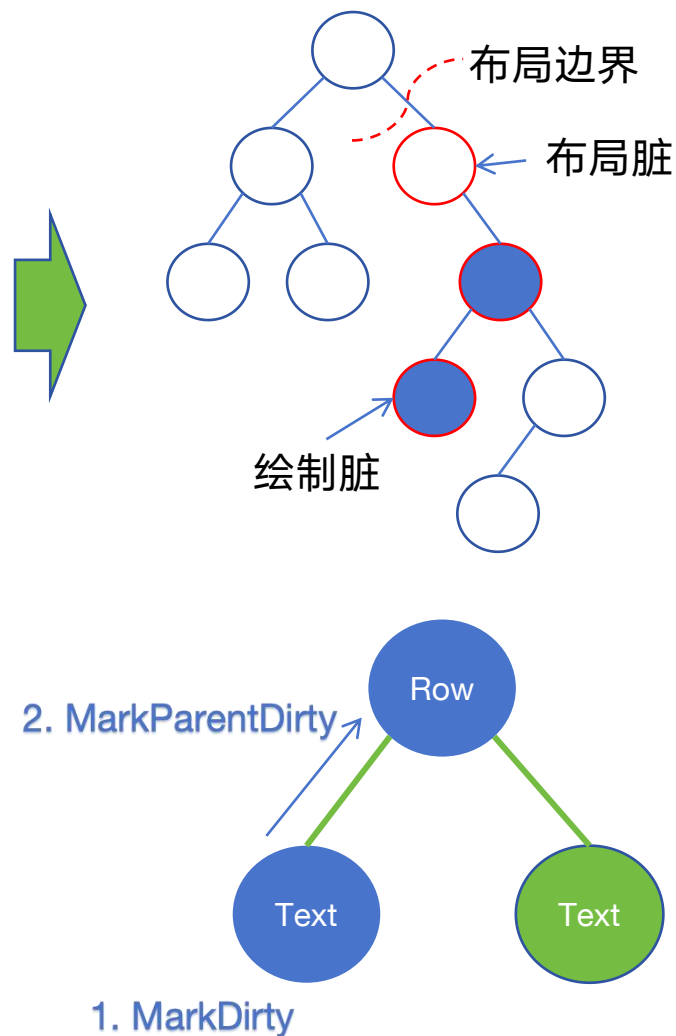


脏节点数组：保存脏节点id

Build过程

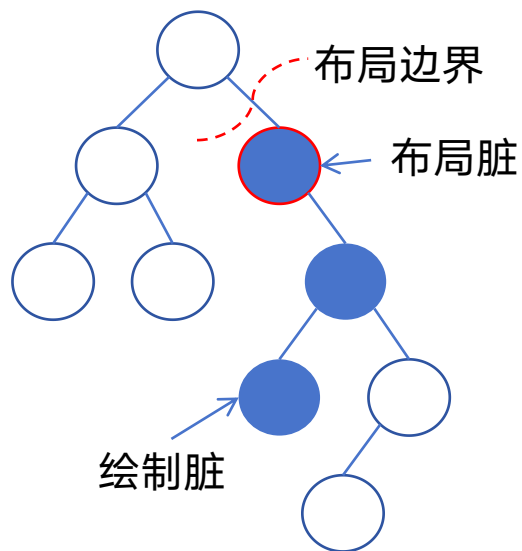


FrameNodeTree



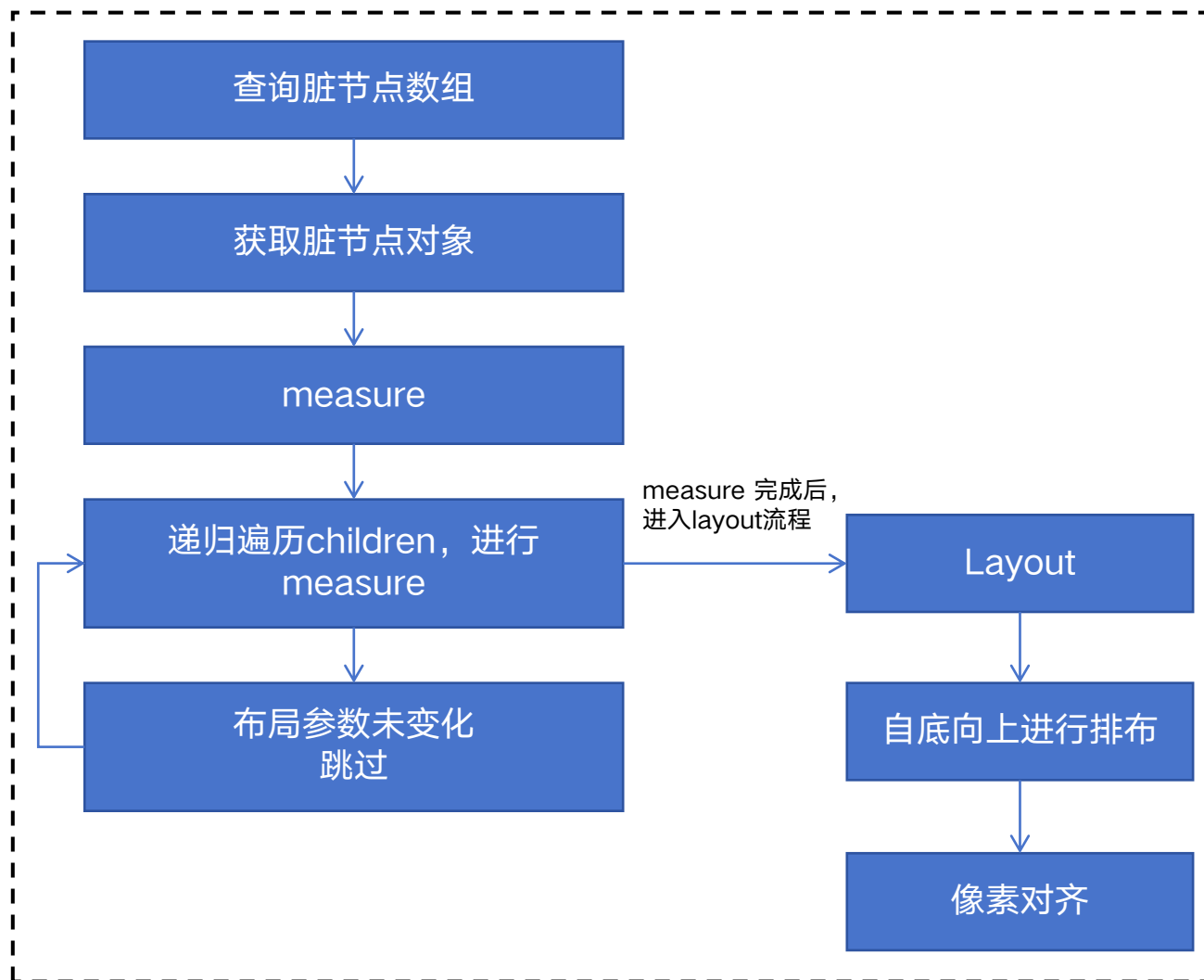
UI框架运行流程：UI更新流程

FrameNodeTree

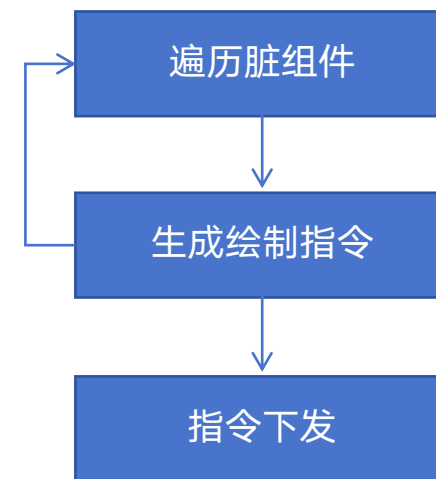


绘制脏不一定触发布局更新，
布局脏一定触发布局更新和
绘制更新

布局过程



绘制过程



状态管理合理使用，精准控制组件更新范围

常见冗余刷新常见：大数据对象关联组件过多，造成刷新范围大

@Observed

```
class UIStyle {  
  translateX: number = 0;  
  translateY: number = 0;  
  scaleX: number = 0.3;  
  scaleY: number = 0.3;  
  width: number = 336;  
  height: number = 178;  
  posX: number = 10;  
  posY: number = 50;  
  alpha: number = 0.5;  
  borderRadius: number = 24;  
  imageWidth: number = 78;  
  imageHeight: number = 78;  
  translateImageX: number = 0;  
  translateImageY: number = 0;  
  fontSize: number = 20;  
}
```

Component1

UIStyle.translateX
UIStyle.translateY

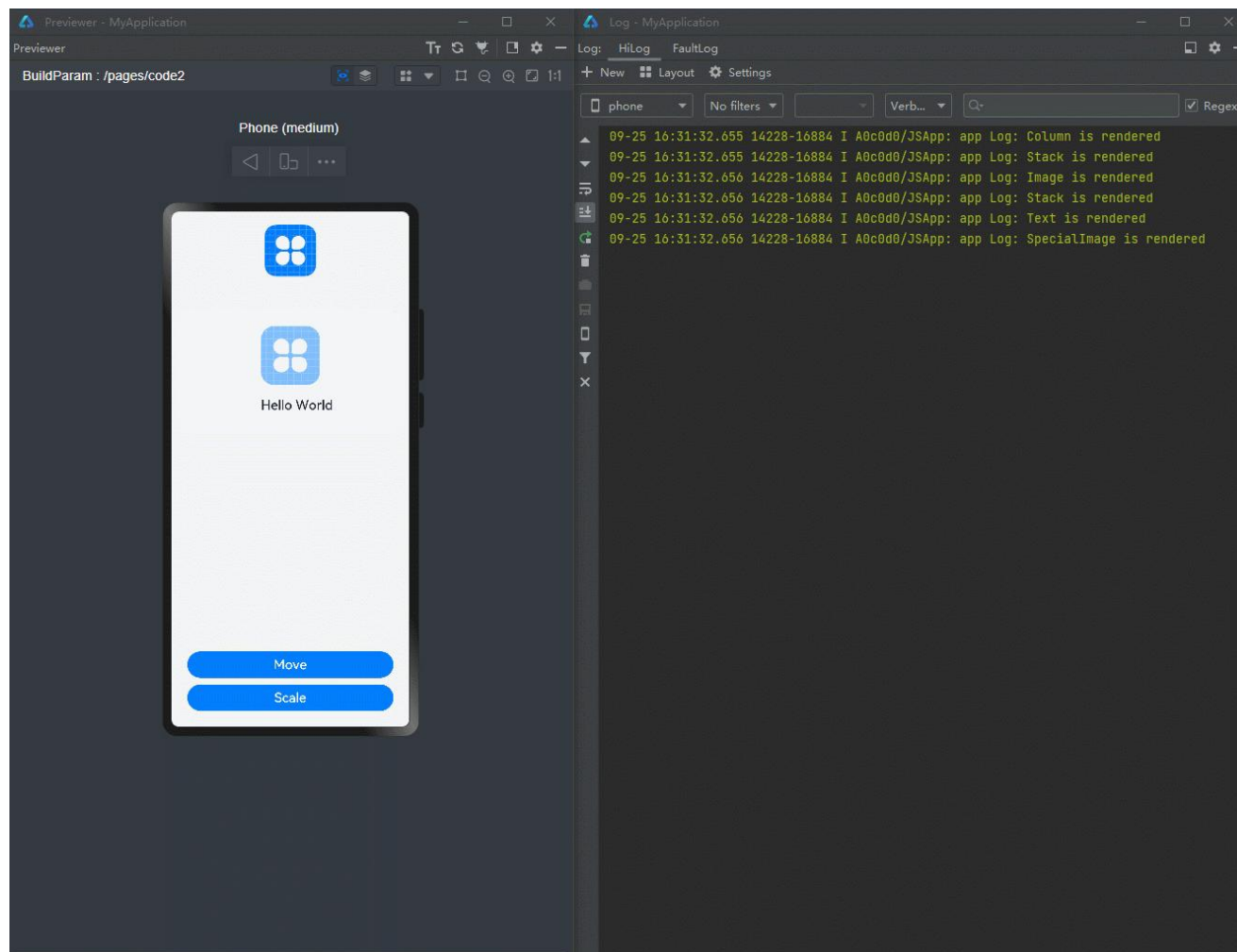
Component2

UIStyle.scaleX
UIStyle.scaleY

Component3

UIStyle.fontSize

同一数据对象中，不同属性被多个组件所使用，
当数据对象任一属性值发生变化时，所有绑定
该对象的组件都被标记为脏，发生刷新



状态管理合理使用，精准控制组件更新范围

通过@Observed装饰器，进行数据合理拆分，实现更新范围最小化

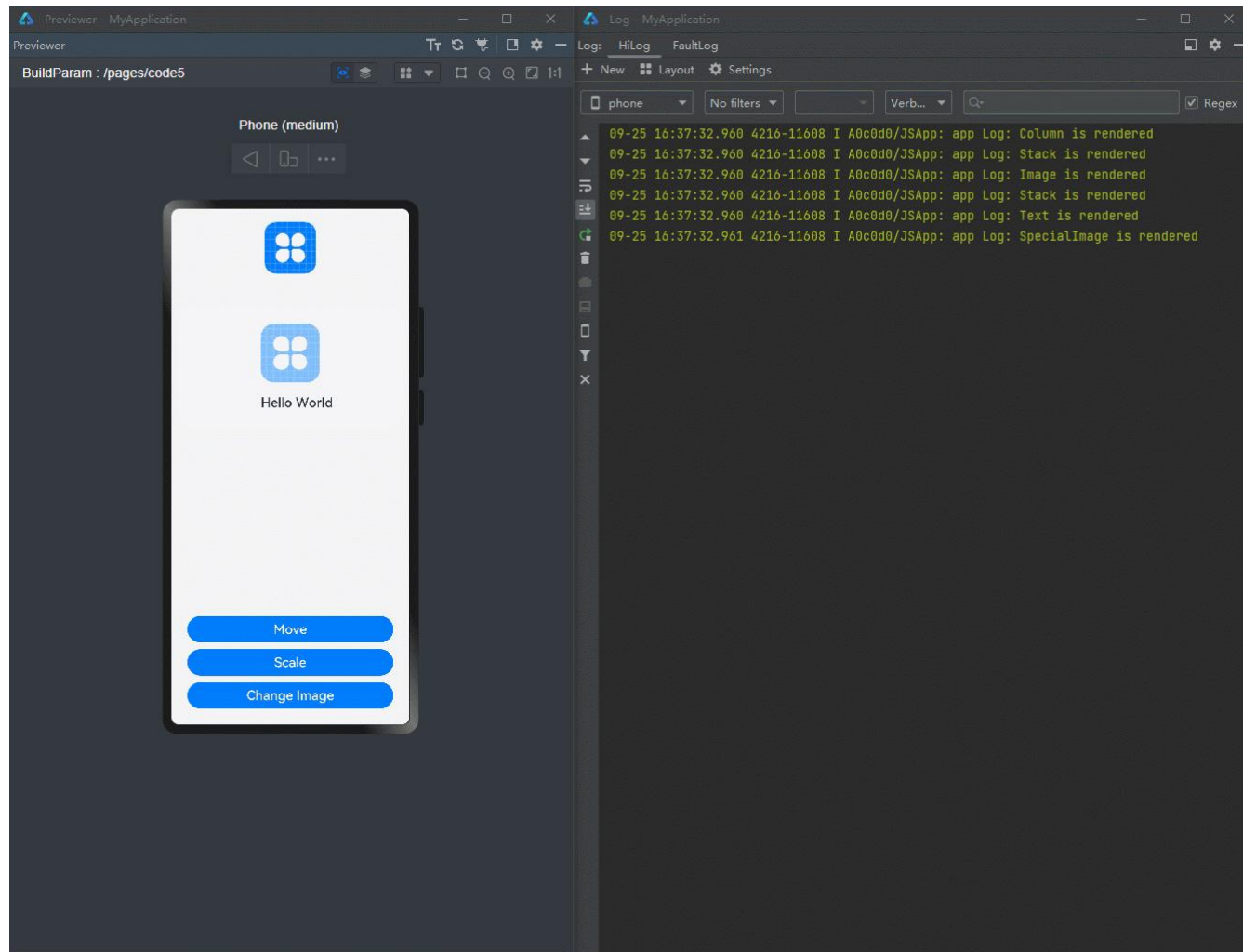
```
@Observed
class NeedRenderTranslate {
    public translateImageX: number = 0;
    public translateImageY: number = 0;
}

@Observed
class NeedRenderScale {
    public scaleX: number = 0.3;
    public scaleY: number = 0.3;
}

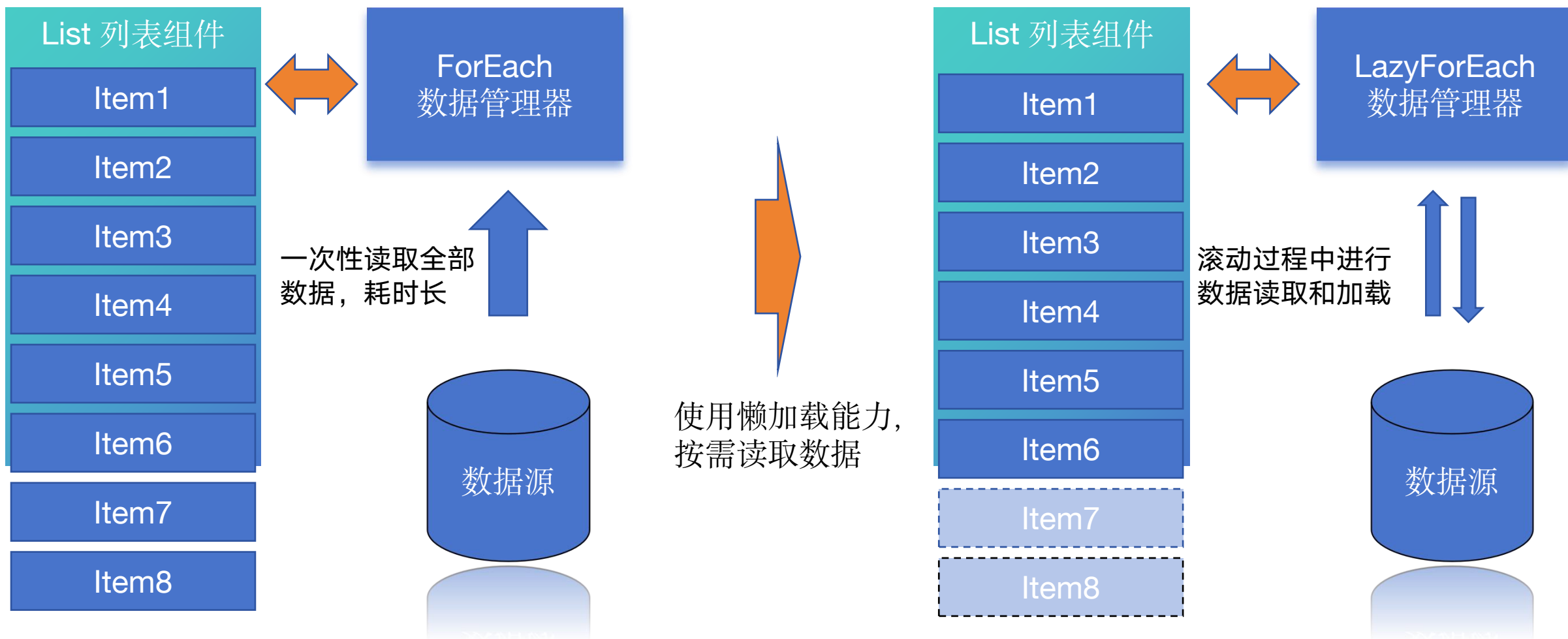
@Observed
class NeedRenderFontSize {
    public fontSize: number = 20;
}

@Observed
class NeedRenderOthers { // properties
    usually used together can be divided
    into the same new divided class
    // . . .
}

@Observed
class UIStyle {
    needRenderTranslate:
    NeedRenderTranslate = new
    NeedRenderTranslate();
    needRenderFontSize:
    NeedRenderFontSize = new
    NeedRenderFontSize();
    needRenderScale: NeedRenderScale
    = new NeedRenderScale();
    // . . .
}
```

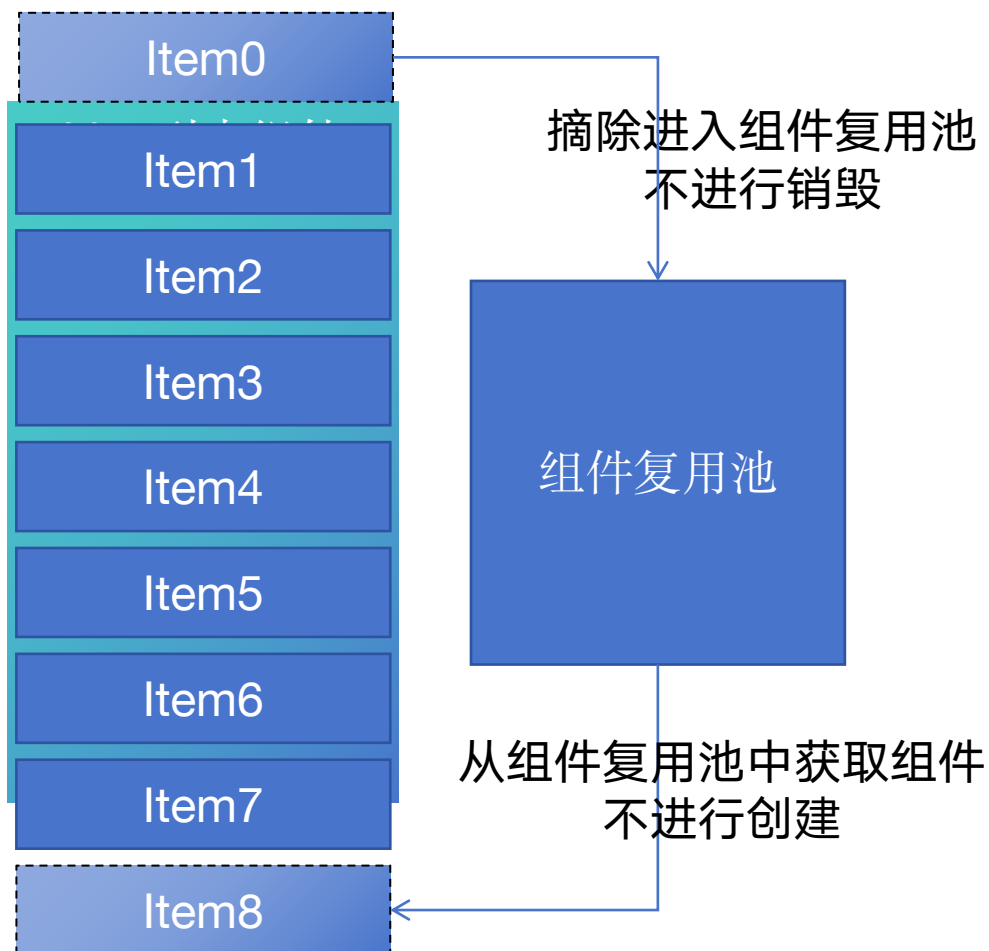


使用懒加载技术，控制列表更新范围，避免超长列表更新阻塞



适用于列表项数量不多或只有1屏数据场景，当列表项数量大时，会造成加载耗时阻塞

组件复用机制，减少组件创建



```
LazyForEach(this.GoodDataOne, (item, index) => {  
  GridItem() {  
    GoodItems({  
      img:item.data.img,  
      webimg:item.data.webimg,  
      hei:item.data.hei,  
    })  
    .reuseId(this.CombineStr(item.type))  
  }  
}, (item) => JSON.stringify(item))
```

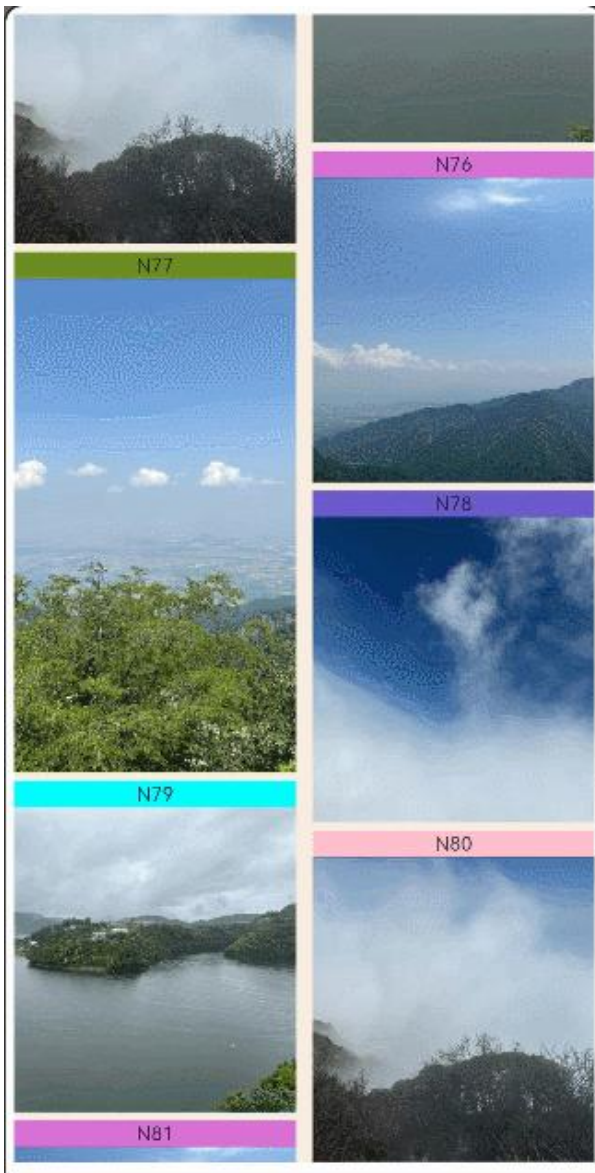
```
@Reusable  
@Component  
struct GoodItems {  
  @State img: Resource = $r("app.media.photo61")  
  @State webimg?: string = ""  
  @State hei: number = 0  
  @LocalStorageLink('storageSimpleProp') simpleVarName:  
string = ""  
  boo: boolean = true  
  index: number = 0  
  controllerVideo: VideoController = new VideoController();
```

```
  aboutToReuse(params)  
  {  
    this.webimg = params.webimg  
    this.img = params.img  
    this.hei = params.hei  
  }  
}
```

```
  build() {  
    // ...  
  }  
}
```


懒加载机制与组件复用机制在组件中的使用——瀑布流布局

效果展示



懒加载能力与 WaterFlow组合使用

```
build() {  
  Column({ space: 2 }) {  
    WaterFlow() {  
      LazyForEach(this.datasource, (item: number) => {  
        FlowItem() {  
          Column() {  
            Text("N" + item).fontSize(12).height('16')  
            Image('res/waterFlowTest (' + item % 5 + ').jpg')  
              .objectFit(ImageFit.Fill)  
              .width('100%')  
              .layoutWeight(1)  
          }  
        }  
        .width('100%')  
        // 提前设定FlowItem高度，避免自适应图片高度  
        .height(this.itemHeightArray[item])  
        .backgroundColor(this.colors[item % 5])  
      }, (item: string) => item)  
    }  
    .columnsTemplate("1fr 1fr")  
    .columnsGap(10)  
    .rowsGap(5)  
    .backgroundColor(0xFAEEE0)  
    .width('100%')  
    .height('80%')  
  }  
}
```

懒加载能力实现无限滚动

```
build() {  
  Column({ space: 2 }) {  
    WaterFlow() {  
      LazyForEach(this.datasource, (item: number) => {  
        FlowItem() {  
          Column() {  
            Text("N" + item).fontSize(12).height('16')  
            Image('res/waterFlowTest (' + item % 5 + ').jpg')  
              .objectFit(ImageFit.Fill)  
              .width('100%')  
              .layoutWeight(1)  
          }  
        }  
        .onAppear() => {  
          // 即将触底时提前增加数据  
          if (item + 20 == this.datasource.totalCount()) {  
            for (let i = 0; i < 100; i++) {  
              this.datasource.AddLastItem()  
            }  
          }  
        }  
        .width('100%')  
        .height(this.itemHeightArray[item % 100])  
        .backgroundColor(this.colors[item % 5])  
      }, (item: string) => item)  
    }  
  }  
}
```

懒加载机制与组件复用机制在组件中的使用——瀑布流布局

懒加载能力结合组件复用机制，达到最佳滚动性能体验

```
build() {
  Column({ space: 2 }) {
    WaterFlow() {
      LazyForEach(this.datasource, (item: number) => {
        FlowItem() {
          // 使用可复用自定义组件
          ResuableFlowItem({ item: item })
        }
      }).onAppear() => {
        // 即将触底时提前增加数据
        if (item + 20 == this.datasource.totalCount()) {
          for (let i = 0; i < 100; i++) {
            this.datasource.AddLastItem()
          }
        }
      })
      .width('100%')
      .height(this.itemHeightArray[item % 100])
      .backgroundColor(this.colors[item % 5])
    }, (item: string) => item)
  }
  .columnsTemplate("1fr 1fr")
  .columnsGap(10)
  .rowsGap(5)
  .backgroundColor(0xFAEEEE)
  .width('100%')
  .height('80%')
}
```

```
@Reusable
@Component
struct ResuableFlowItem {
  @State item: number = 0
```

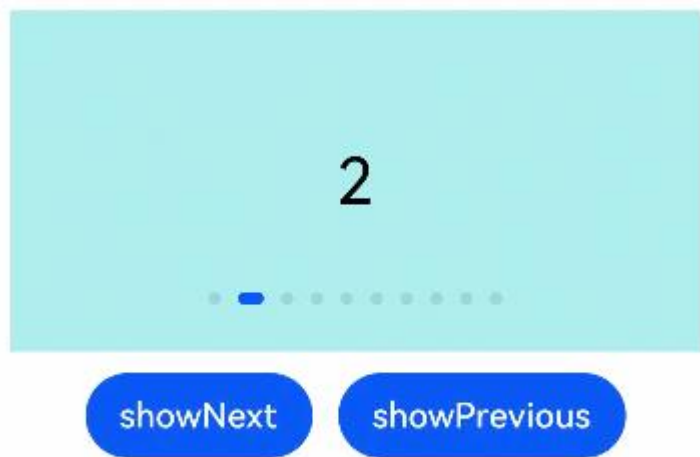
// 从复用缓存中加入到组件树之前调用，可在此处更新组件的状态变量以展示正确内容

```
  aboutToReuse(params) {
    this.item = params.item;
  }
```

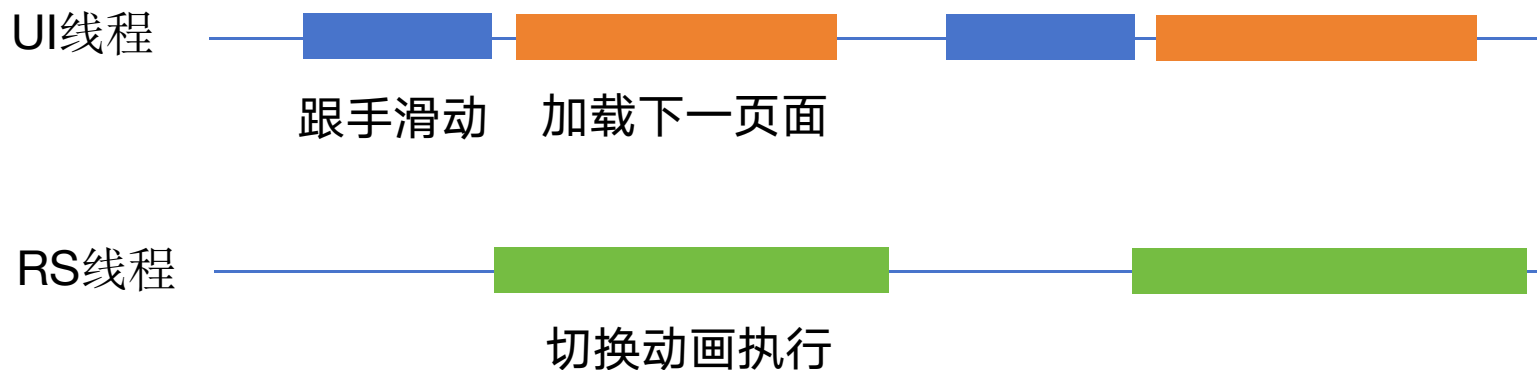
```
build() {
  Column() {
    Text("N" + this.item).fontSize(12).height('16')
    Image('res/waterFlowTest (' + this.item % 5 + ').jpg')
      .objectFit(ImageFit.Fill)
      .width('100%')
      .layoutWeight(1)
  }
}
```


懒加载机制与组件预加载机制在组件中的使用——页面滑动

翻页效果展示



滑动动画&页面创建并行

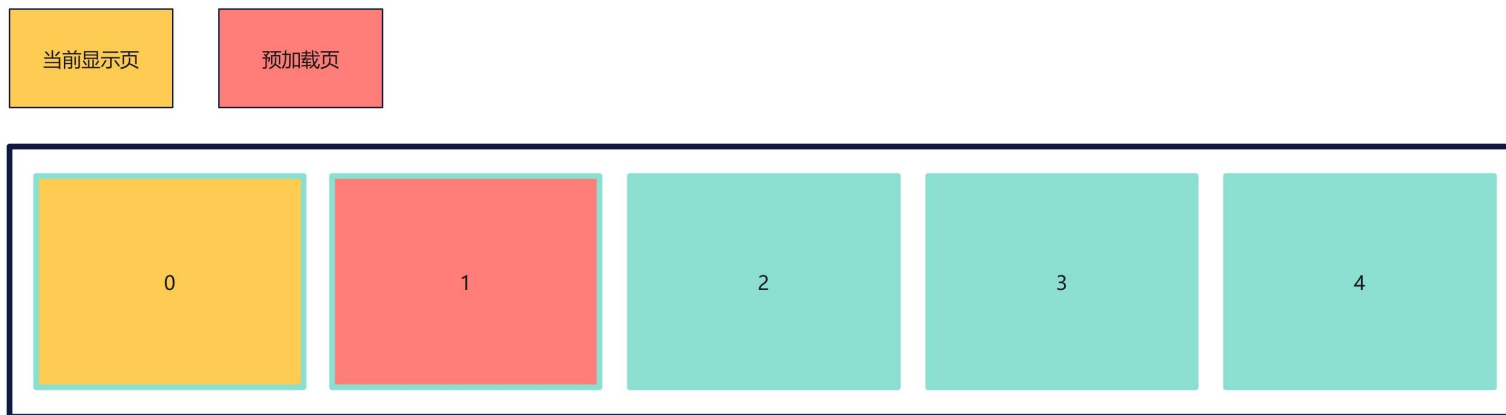


当页面较多数量大时，一次性创建全部页面显然并不划算

优化机制：

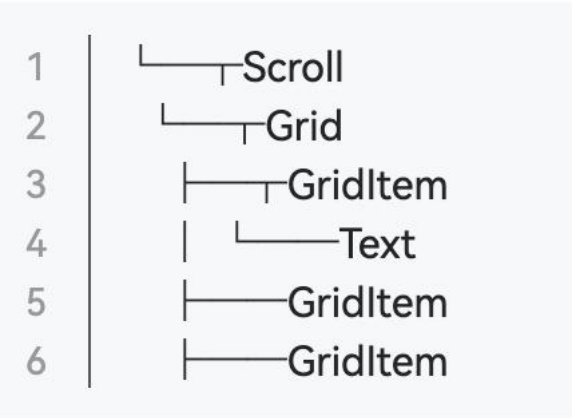
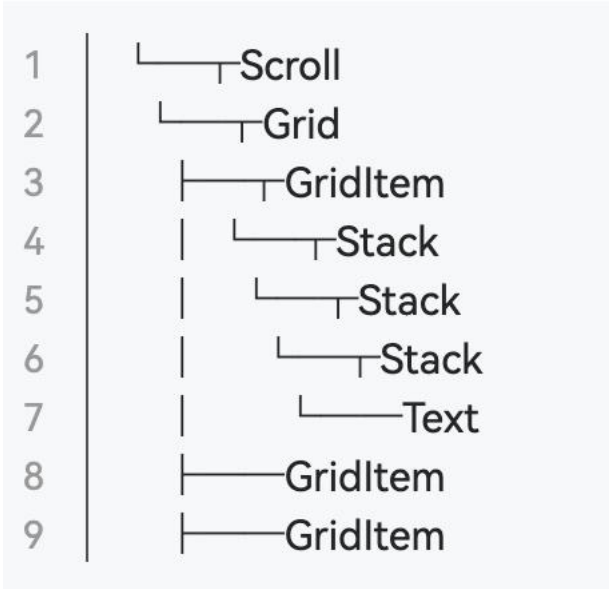
- 利用滑动过程动画时间，并行进行页面创建
- 控制加载的页面数量

通过cacheCount设置预加载页面数量



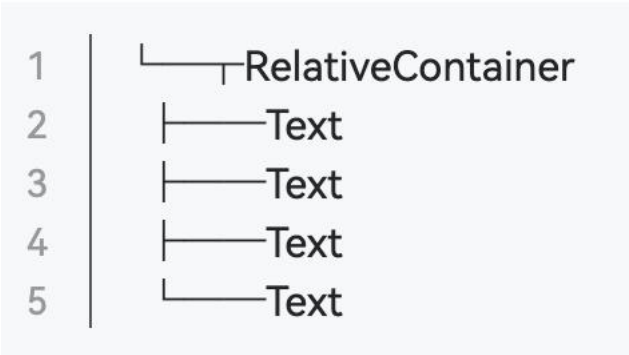
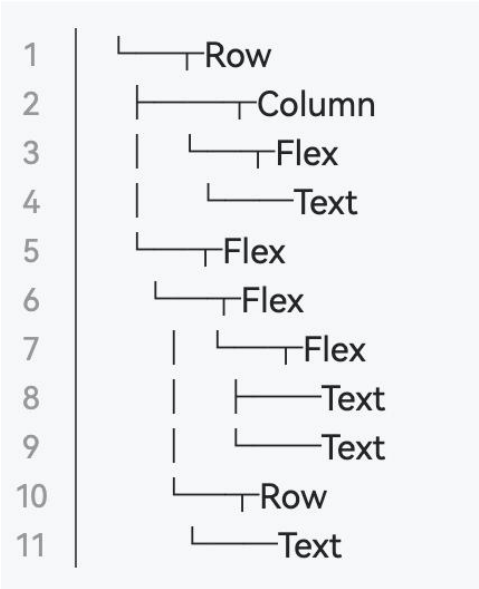
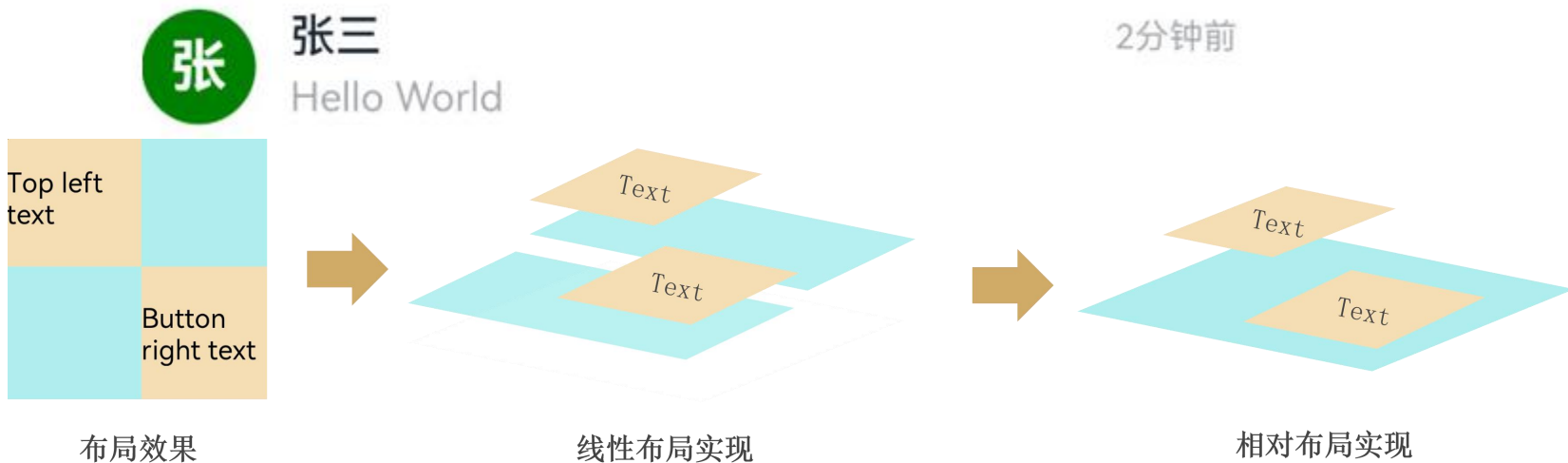
优化页面布局，高效组织UI结构

合理使用布局，降低UI层级



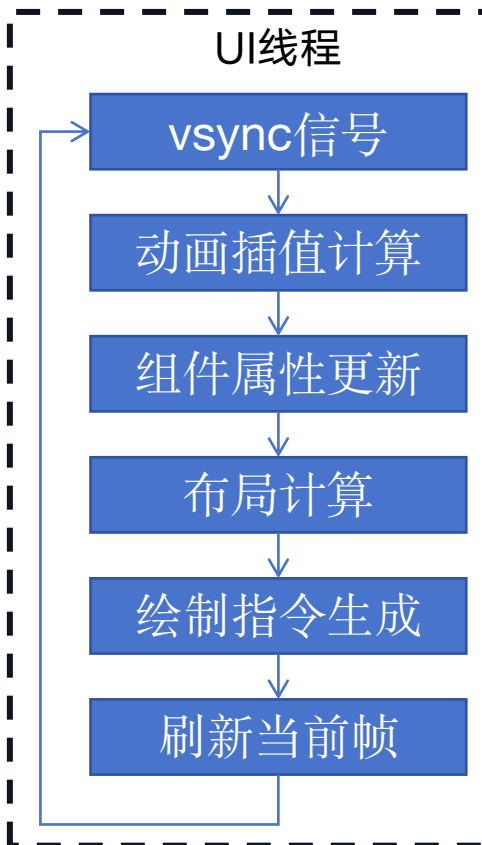
归并相同类型布局，避免冗余组件使用

选用扁平化布局，降低UI层级

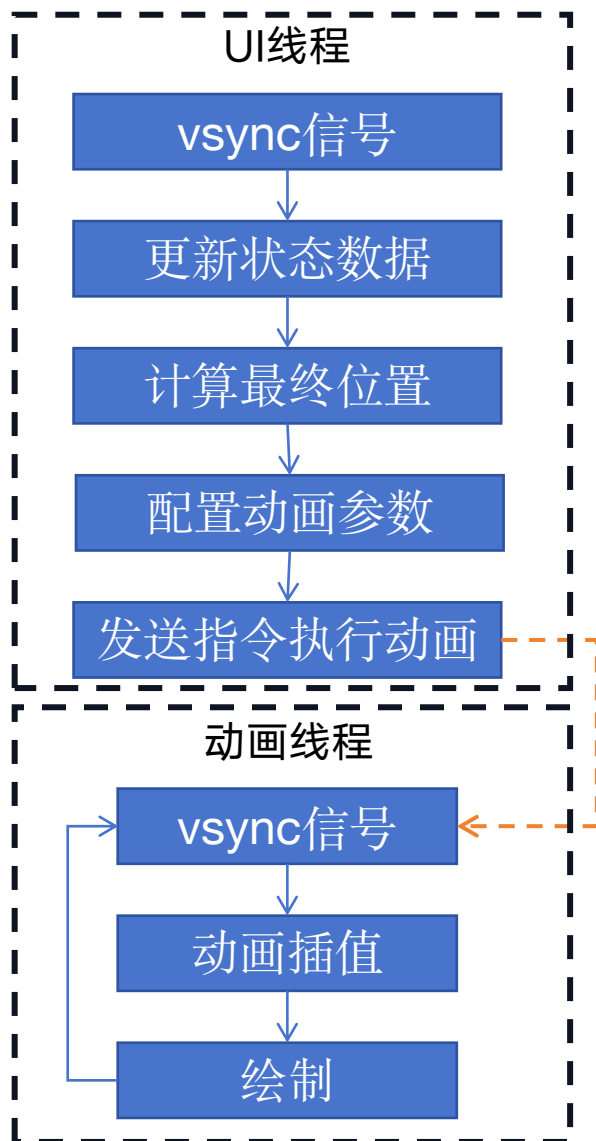


使用属性动画能力，避免帧动画占用UI线程

传统动画处理过程



鸿蒙动画处理过程



动画使用示例

```
@State textScaleX: number = 1;  
@State textScaleY: number = 1;
```

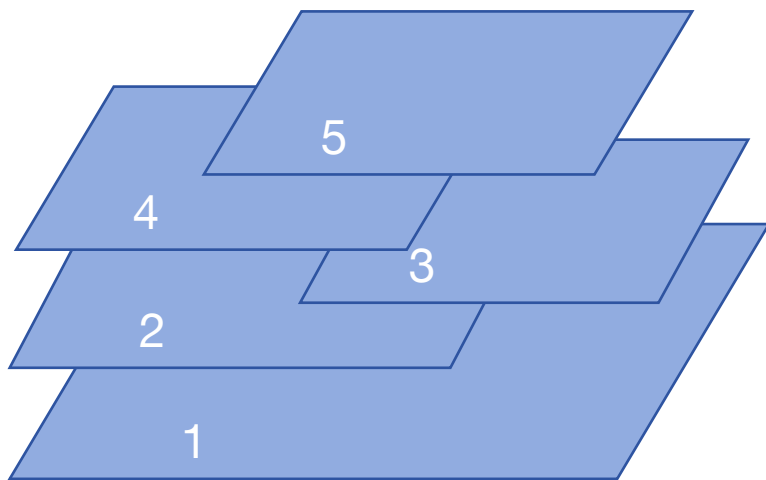
```
Text()  
  .backgroundColor(Color.Blue)  
  .fontColor(Color.White)  
  .fontSize(20)  
  .width(10)  
  .height(10)  
  .scale({ x: this.textScaleX, y: this.textScaleY })  
  .margin({ top: 100 })
```

```
Button('图形变换属性')  
  .onClick(() => {  
    animateTo({ duration: 1000 }, () => {  
      this.textScaleX = 10;  
      this.textScaleY = 10;  
    })  
  })
```



通过节点组缓存（renderGroup），提升渲染性能

组件渲染顺序



按Z序自底向上逐层绘制

节点组缓存，一次性绘制



首次绘制：按Z序自底向上逐层绘制

缓存绘制纹理



发生重绘

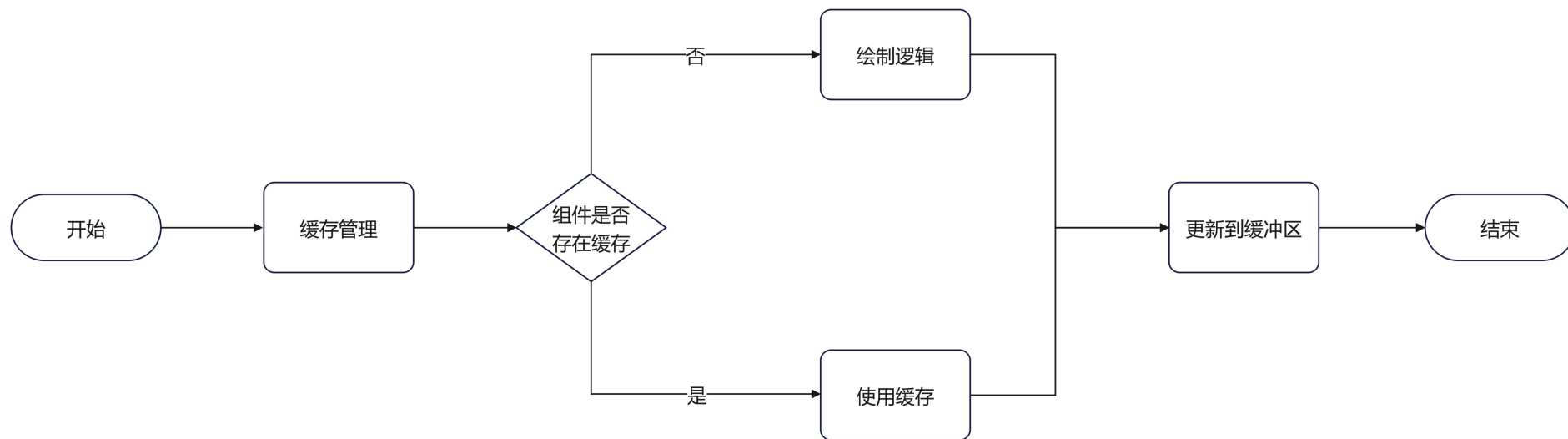


读取纹理，直接绘制



通过节点组缓存（renderGroup），提升渲染性能

组件渲染流程



合理使用节点组缓存，平衡性能&内存

适用场景

缓存更新条件：

- ① 组件在当前组件树上
- ② 组件renderGroup被标记为true
- ③ 组件内容被标脏

缓存清理条件：

- ① 组件不存在于组件树上
- ② 组件renderGroup被标记为false

✓ 组件内容固定不变&内部无动效

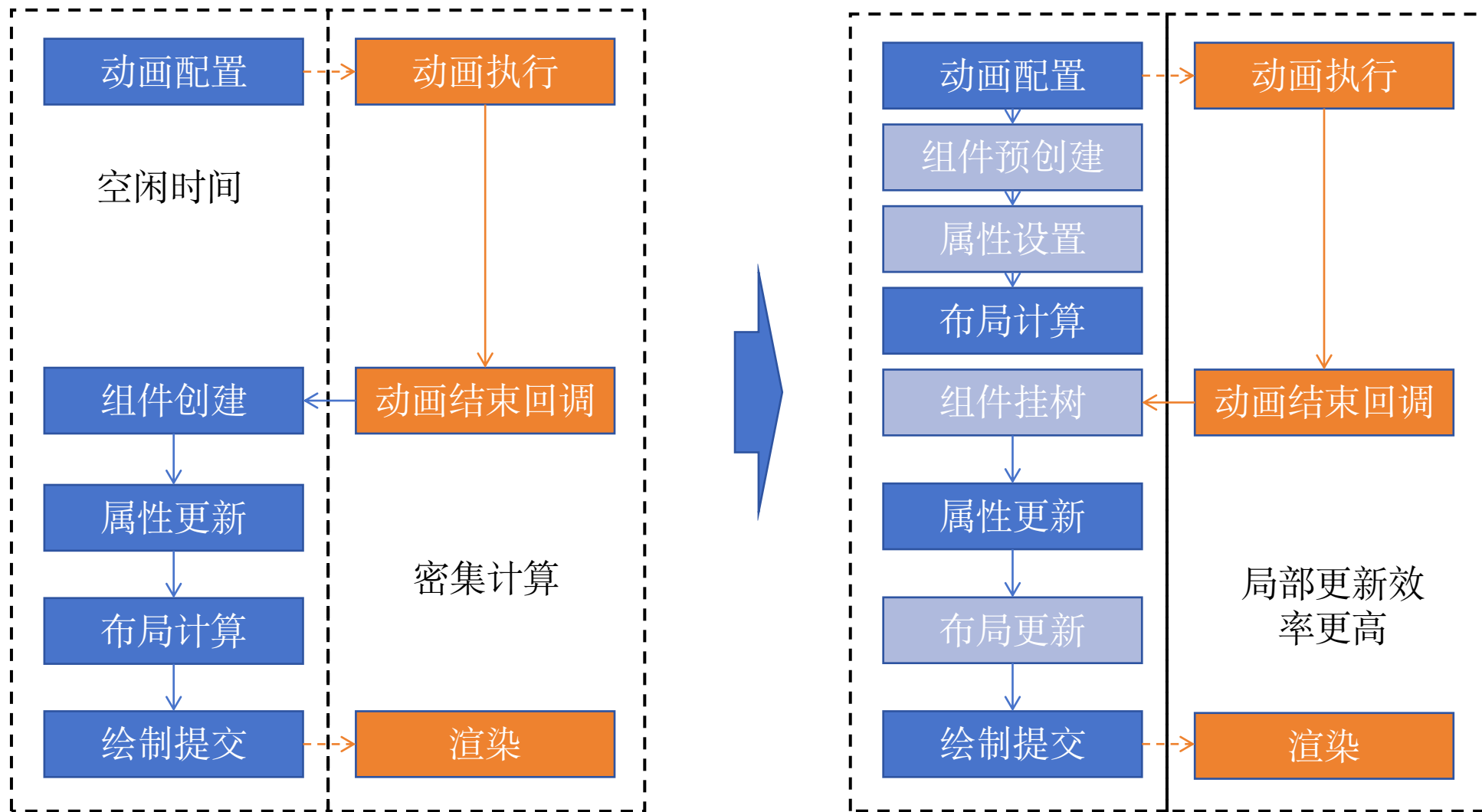
组件内容固定不变或更新频率低，会加大缓存的使用率，减少缓存更新增加的开销

✓ 执行动画的过程

执行动画过程中内容通常没有变化，此时在动画执行前将动画目标组件标记为节点组，可提升动画帧率，动画结束后，恢复原始状态

探索中的性能提升思路

组件预创建能力
利用UI线程空闲时间，提前准备数据



应用性能分析工具CPU Profiler：查看ArkTS耗时分析

The screenshot displays the CPU Profiler interface within an IDE. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, and Help. The main window is titled 'MyApplication' and shows a project structure on the left with 'EntryAbility.ts' selected. The 'Profiler' tab is active, showing a timeline of ArkTS calls. A red circle '1' highlights the 'Profiler' tab. A red circle '2' highlights the 'Select device' dropdown. A red circle '3' highlights the 'Select app or process' dropdown. A red circle '4' highlights the 'Time' button in the bottom left. A red circle '5' highlights the 'Create Session' button. The timeline shows a 3.8s duration. The 'Details' table lists the following data:

Symbol Name	Weight	%	Self	%	Category
(root)	3s 888ms 889μs	100.0%	0ns	0.0%	System
work1 entry/src/main/ets/pages/Index.ets	3s 586ms 258μs	92.2%	76μs	0.0%	ArkTS
work2 entry/src/main/ets/pages/Index.ets	2s 891ms 320μs	74.3%	49μs	0.0%	ArkTS
work3 entry/src/main/ets/pages/Index.ets	1s 696ms 210μs	43.6%	94μs	0.0%	ArkTS
sleep entry/src/main/ets/pages/Index.ets	1s 479ms 213μs	38.0%	660ms 529μs	17.0%	ArkTS
add entry/src/main/ets/pages/Index.ets	197ms 72μs	5.1%	3ms 471μs	0.1%	NAPI
Add(napi_env_*, napi_callback_info_*) /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js	193ms 601μs	5.0%	0ns	0.0%	Native
func10 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js	193ms 601μs	5.0%	0ns	0.0%	Native
func20 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js	193ms 601μs	5.0%	4ms 989μs	0.1%	Native
func30 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js	188ms 612μs	4.9%	13ms 941μs	0.4%	Native
loopFunction0 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js	174ms 671μs	4.5%	174ms 671μs	4.5%	Native
sleep(GC) entry/src/main/ets/pages/Index.ets	17ms 66μs	0.4%	17ms 66μs	0.4%	ArkTS
info entry/src/main/ets/pages/Index.ets	2ms 765μs	0.1%	2ms 765μs	0.1%	NAPI
sleep entry/src/main/ets/pages/Index.ets	987ms 949μs	25.4%	435ms 323μs	11.2%	ArkTS
add entry/src/main/ets/pages/Index.ets	195ms 373μs	5.0%	4ms 636μs	0.1%	NAPI
Add(napi_env_*, napi_callback_info_*) /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js	190ms 737μs	4.9%	0ns	0.0%	Native
func10 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js	190ms 737μs	4.9%	0ns	0.0%	Native
func20 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js	190ms 737μs	4.9%	0ns	0.0%	Native
func30 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js	190ms 737μs	4.9%	6ms 28μs	0.2%	Native
loopFunction0 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js	184ms 709μs	4.7%	184ms 709μs	4.7%	Native
sleep(GC) entry/src/main/ets/pages/Index.ets	9ms 173μs	0.2%	9ms 173μs	0.2%	ArkTS
info entry/src/main/ets/pages/Index.ets	2ms 566μs	0.1%	2ms 566μs	0.1%	NAPI
sleep entry/src/main/ets/pages/Index.ets	489ms 735μs	12.6%	225ms 523μs	5.8%	ArkTS

The 'Heaviest Stack' table lists the following data:

Weight	%	Category	Symbol Name
3s 888ms 889μs	100.0%	System	(root)
3s 586ms 258μs	92.2%	ArkTS	work1 entry/src/main/ets/pages/Index.ets
2s 891ms 320μs	74.3%	ArkTS	work2 entry/src/main/ets/pages/Index.ets
195ms 373μs	5.0%	NAPI	add entry/src/main/ets/pages/Index.ets
190ms 737μs	4.9%	Native	Add(napi_env_*, napi_callback_info_*) /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js
190ms 737μs	4.9%	Native	func10 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js
190ms 737μs	4.9%	Native	func20 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js
190ms 737μs	4.9%	Native	func30 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js
184ms 709μs	4.7%	Native	loopFunction0 /proc/5936/root/data/storage/el1/bundle/libs/arm/libentry.js

The bottom status bar shows 'Sync failed. Follow the given solution to avoid unexpected errors.: Cause: The @ohos/hvigor-ohos-plugin version (2.3.0) is not... (30 minutes ago)' and '1:1 LF UTF-8 2 spaces'.

HiDumper命令行工具：查看UI结构

- ① 开启ArkUI的debug模式：hdc shell param set persist.ace.debug.enabled 1
- ② 重新启动应用
- ③ 获取当前页面对应应用的window ID：hdc shell hidumper -s WindowManagerService -a '-a'
- ④ 通过WinId获取对应页面的控件树文件：hdc shell hidumper -s WindowManagerService -a '-w 28 -element -c'
- ⑤ 查看 arkui.dump 文件

```
# hidumper -s WindowManagerService -a '-a'
```

-----[ability]-----						
-----WindowManagerService-----						
-----ScreenGroup 1-----						
WindowName	DisplayId	Pid	WinId	Type	Mode	Flag
SystemUi_NavigationB	0	1045	7	2112	102	1
SystemUi_PrivacyIndi	0	1045	8	2111	102	1
SystemUi_StatusBar	0	1045	6	2108	102	1
demo0	0	22642	28	1	1	1
EntryView	0	1086	4	2001	1	0
ScreenLockWindow	0	1045	1	2110	1	0
SystemUi_VolumePanel	0	1045	2	2111	1	1
imeWindow	0	1125	3	2105	102	1
RecentView	0	1086	5	2115	1	0
SystemUi_DropdownPan	0	1045	9	2109	1	1
SystemUi_BannerNotic	0	1045	10	2111	1	1

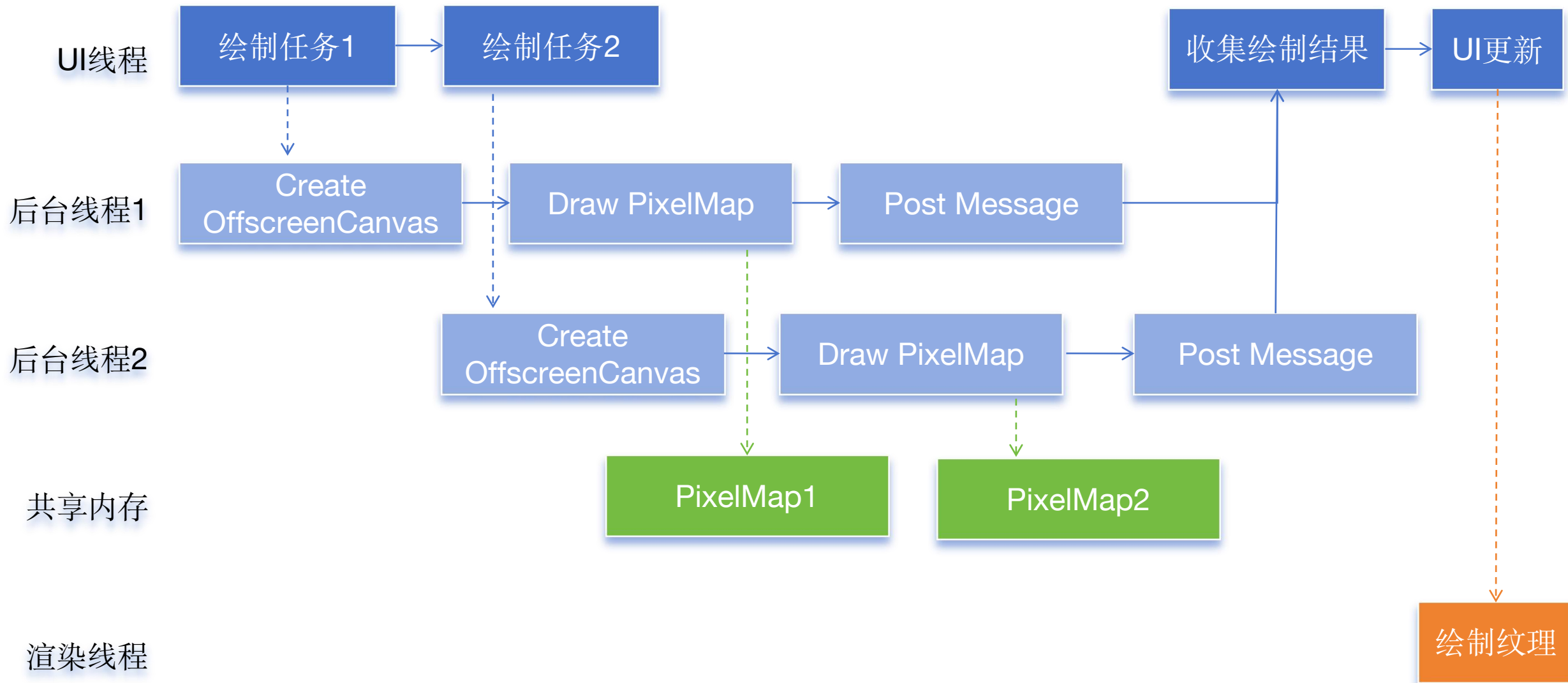
Focus window: 28
total window num: 11

```
# hidumper -s WindowManagerService -a '-w 28 -element -c'
```

-----[ability]-----	
-----WindowManagerService-----	
WindowName: demo0	
DisplayId: 0	
WinId: 28	
Pid: 22642	
Type: 1	
Mode: 1	
Flag: 1	
Orientation: 0	
IsStartingWindow: false	
FirstFrameCallbackCalled: 1	
IsVisible: true	
Focusable: true	
DecoStatus: true	
IsPrivacyMode: false	
OnlySkipSnapshot: 0	
WindowRect: [0, 72, 720, 1136]	
TouchHotAreas: [0, 72, 720, 1136]	
dumpFilePath: /data/storage/e12/base/haps/entry/files/arkui.dump	

探索中的性能提升思路

多线程离屏绘制：并行绘制，UI线程显示纹理



HiDumper命令行工具：查看UI结构

字段说明

childSize: 组件内子组件数量

ID: 当前组件的ID值, 此ID非应用设置, 而是自增累加

Depth: 当前组件在UI Tree中深度

IsDisappearing: 显示状态

FrameRect: 组件尺寸

Background: 组件背景色

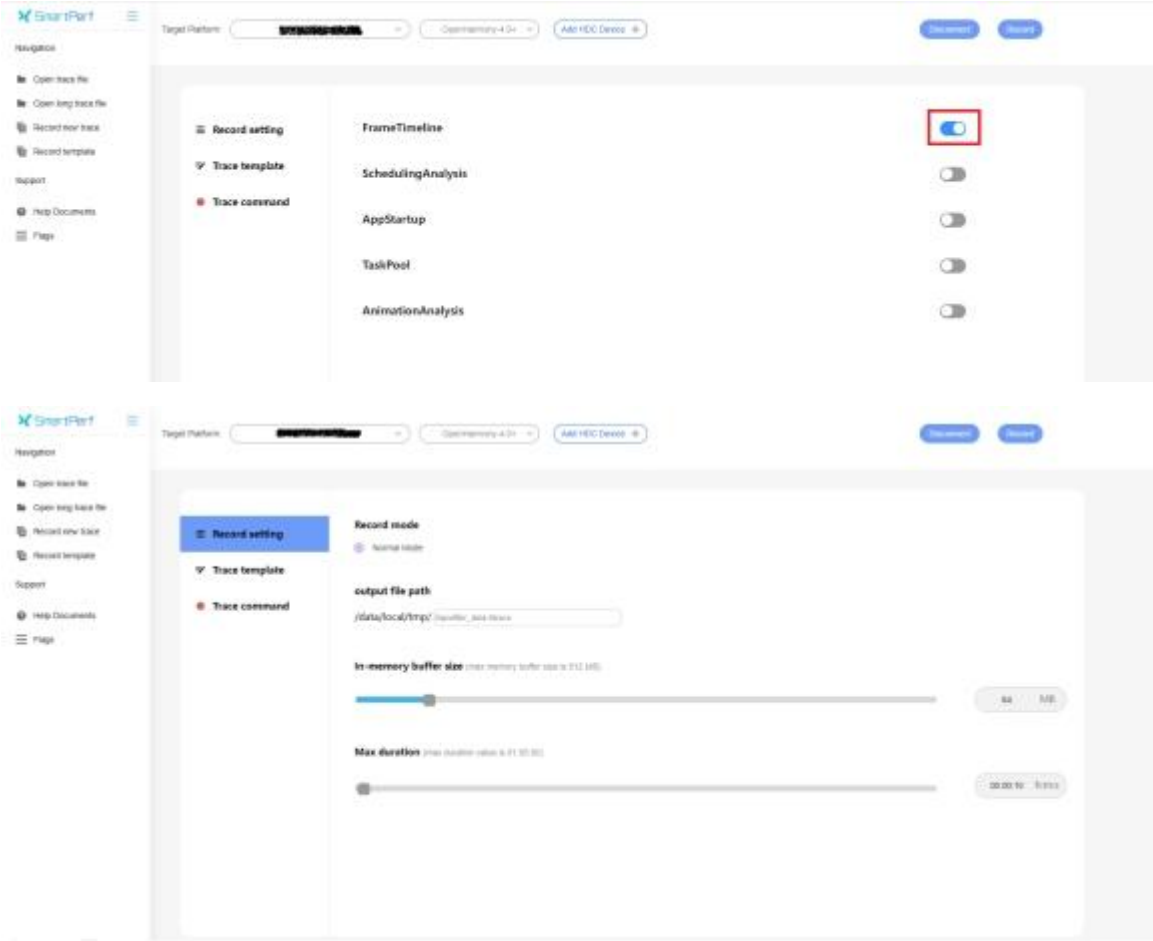
其他: 各组件渲染属性都可查看

// arkui.dump文件内容片断

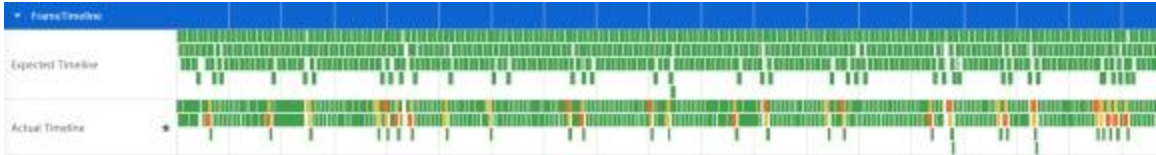
```
|-> GridItem childSize:1
| ID: 22
| Depth: 9
| IsDisappearing: 0
| FrameRect: RectT (360.00, 0.00) - [180.00 x 29.00]
| BackgroundColor: #00000000
...
|-> Stack childSize:1
| ID: 23
| Depth: 10
| IsDisappearing: 0
| FrameRect: RectT (0.00, 0.00) - [180.00 x 29.00]
| BackgroundColor: #FFFFFF00
...
|-> Stack childSize:1
| ID: 24
| Depth: 11
| IsDisappearing: 0
| FrameRect: RectT (0.00, 0.00) - [180.00 x 29.00]
| BackgroundColor: #FF0000FF
...
|-> Stack childSize:1
| ID: 25
| Depth: 12
| IsDisappearing: 0
| FrameRect: RectT (0.00, 0.00) - [180.00 x 29.00]
| BackgroundColor: #00000000
...
|-> Text childSize:0
| ID: 26
| Depth: 13
| IsDisappearing: 0
| FrameRect: RectT (83.00, 0.00) - [14.00 x 29.00]
| BackgroundColor: #00000000
...
```

SmartPerf-Host工具分析应用性能

配置SmartPerf参数



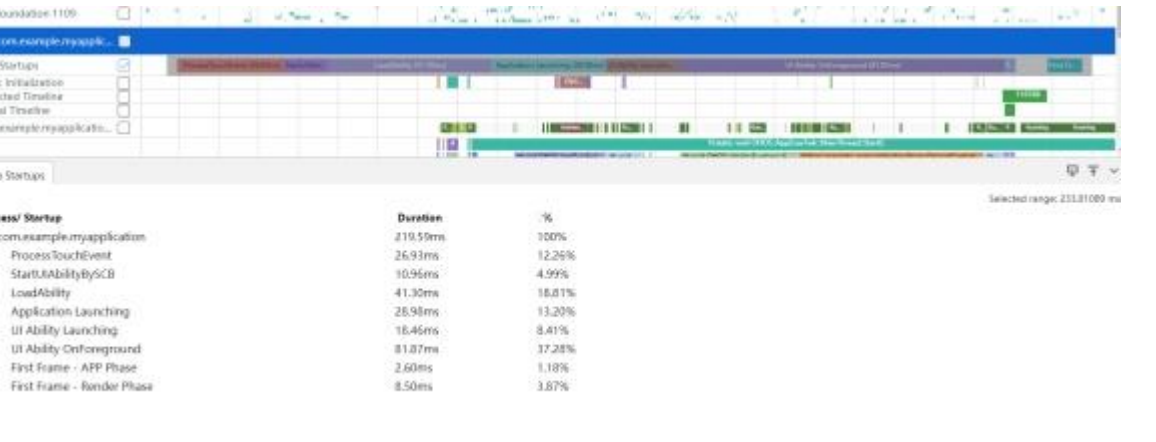
分析UI数据



帧率数据



线程Trace



耗时分析

ArkUI-X 跨平台项目期待您的加入共建

<https://gitee.com/arkui-x>



ArkUI-X

ArkUI-X扩展ArkUI开发框架到多个OS平台, 让开发者基于一套主代码, 就可以构建支持多平台的精美、高性能应用。The ArkUI-X project extends the ArkUI framework to multiple OS platforms. This enables developers to use one main set of code to develop applications for multiple OS platforms.

关注 683

概览

仓库 14

任务 71

Pull Requests 14

动态

成员 19

社区已于2023-12-15发布ArkUI-X-v1.0.0-Release版本, 详情请参考<https://gitee.com/arkui-x/docs/blob/master/zh-cn/release-notes/ArkUI-X-v1.0.0-release.md>

精选

app_framework

ArkUI-X Application Framework | ArkUI-X应用适配层

26 27 10

arkui_for_android

ArkUI-X adaptation to Android | ArkUI-X支持Android平台的适配层

27 28 17

arkui_for_ios

ArkUI-X adaptation to iOS | ArkUI-X支持iOS平台的适配层

24 21 14

cli

ArkUI-X command line tool for cross-platform builds | ArkUI-X跨平台应用构建命令行工具

21 6 2

samples

Cross-platform use cases of ArkUI-X | ArkUI-X跨平台应用示例

26 35 9

docs

ArkUI-X documentation | ArkUI-X开发者文档

33 70 12