HUAWEI

< HDC.Together >

HUAWEI DEVELOPER CONFERENCE 2021

HarmonyOS应用开发案例分享：华为图库

1. 华为首批基于HarmonyOS 新一代UI编程框架完成开发的自研系统级应用

2. 体现HarmonyOS特征“一次开发多端部署”、“分布式超级终端” 的典型应用

3. 依赖HarmonyOS能力子系统数目多，且性能要求高


UX
设计
架构
设计
模块
设计
开发
实践

构建跨终端一致的UX自适应架构

构建跨终端高价值的体验差异化竞争力

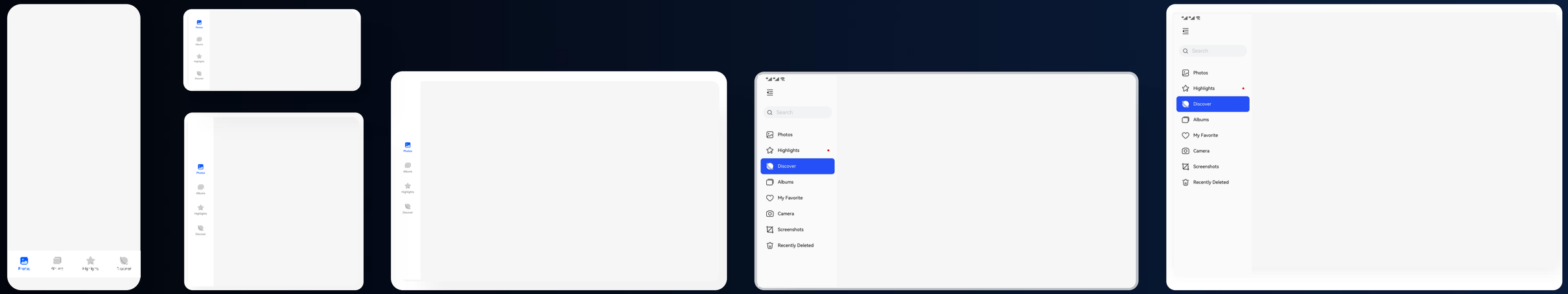
Product Value

一套UX适配规则适配HarmonyOS多设备，支撑一次开发多设备部署，代码最大化复用，提高效率。

User Value

保障用户在多终端间基础体验的一致性，同时发挥不同终端优势，构建高价值差异化体验。

图库导航布局：导航架构基于断点响应式变化



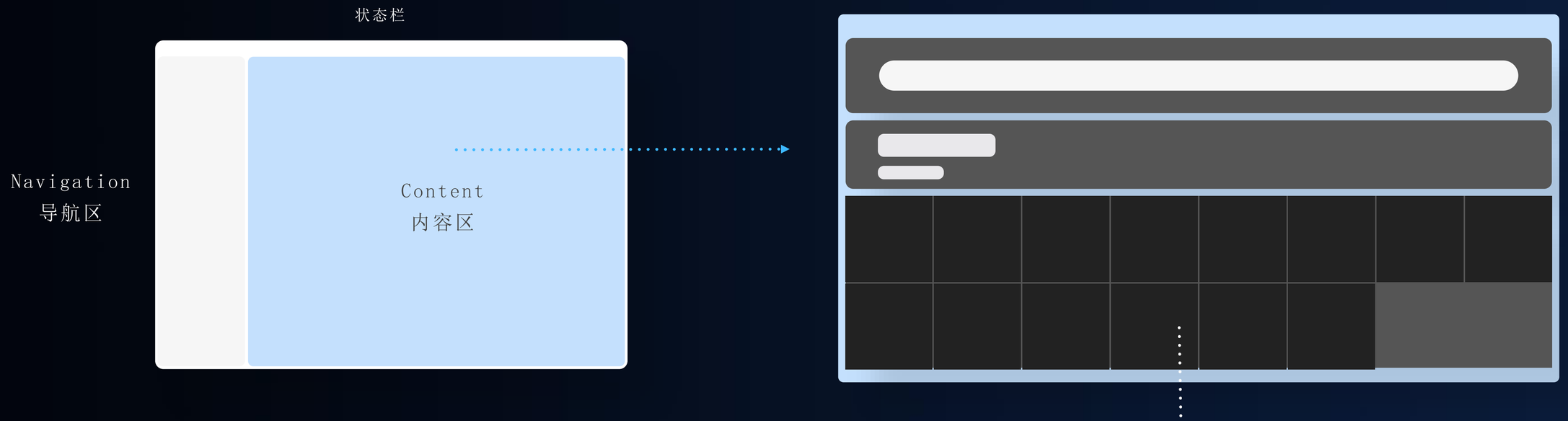
Bottom navigation tabs

Side tabs

Side bar

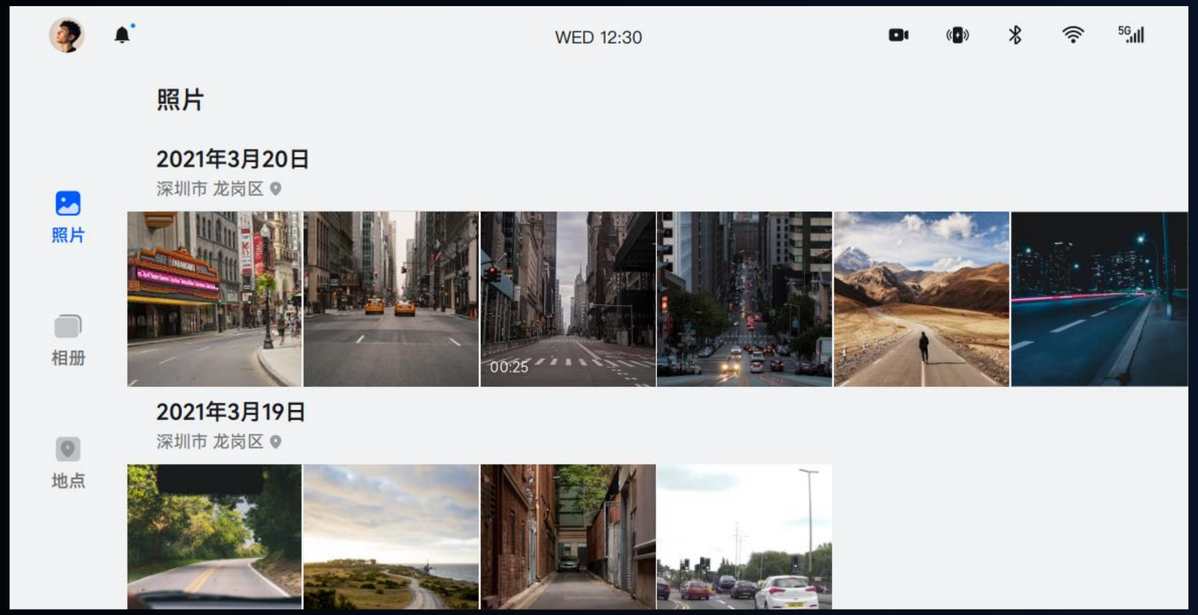
400

图库内容布局：同样基于断点响应式变化

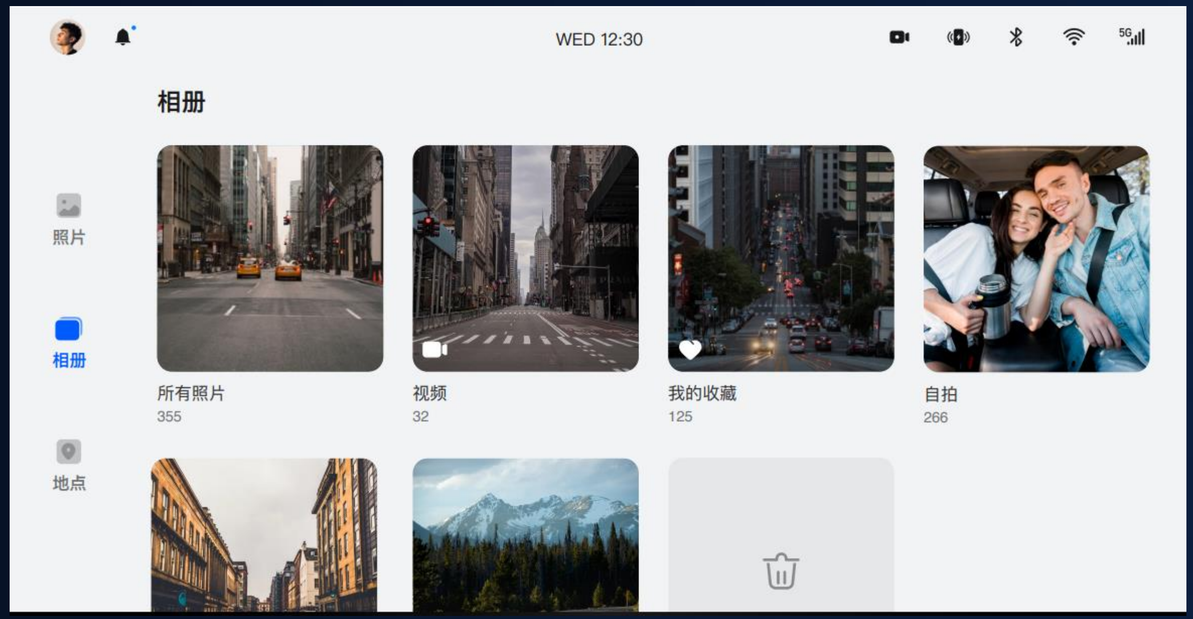


内容区图片宫格展示根据窗口宽度进行自适应增减

图库在某A设备上的UX设计实例

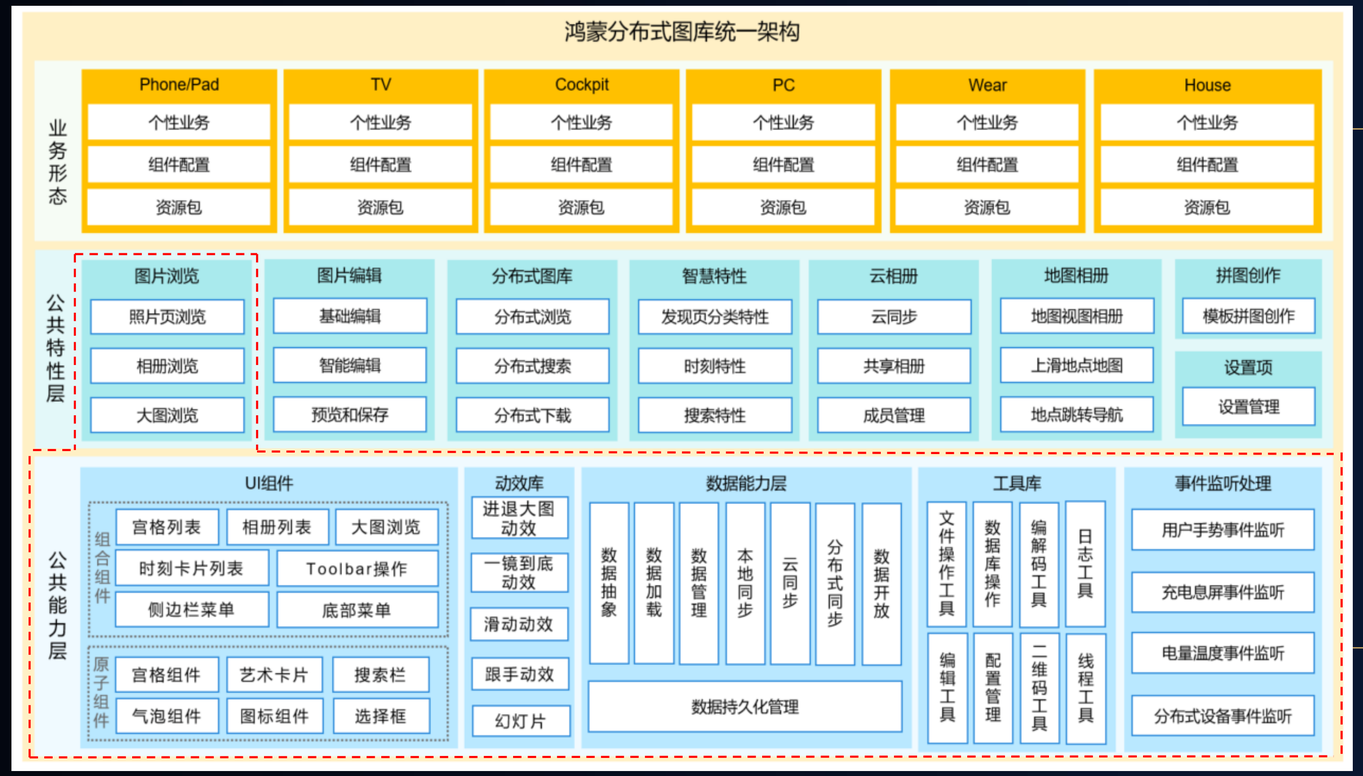


照片页



相册页

业务模块化、能力组件化分层架构支撑一次开发多端复用
配置部署和数据开放



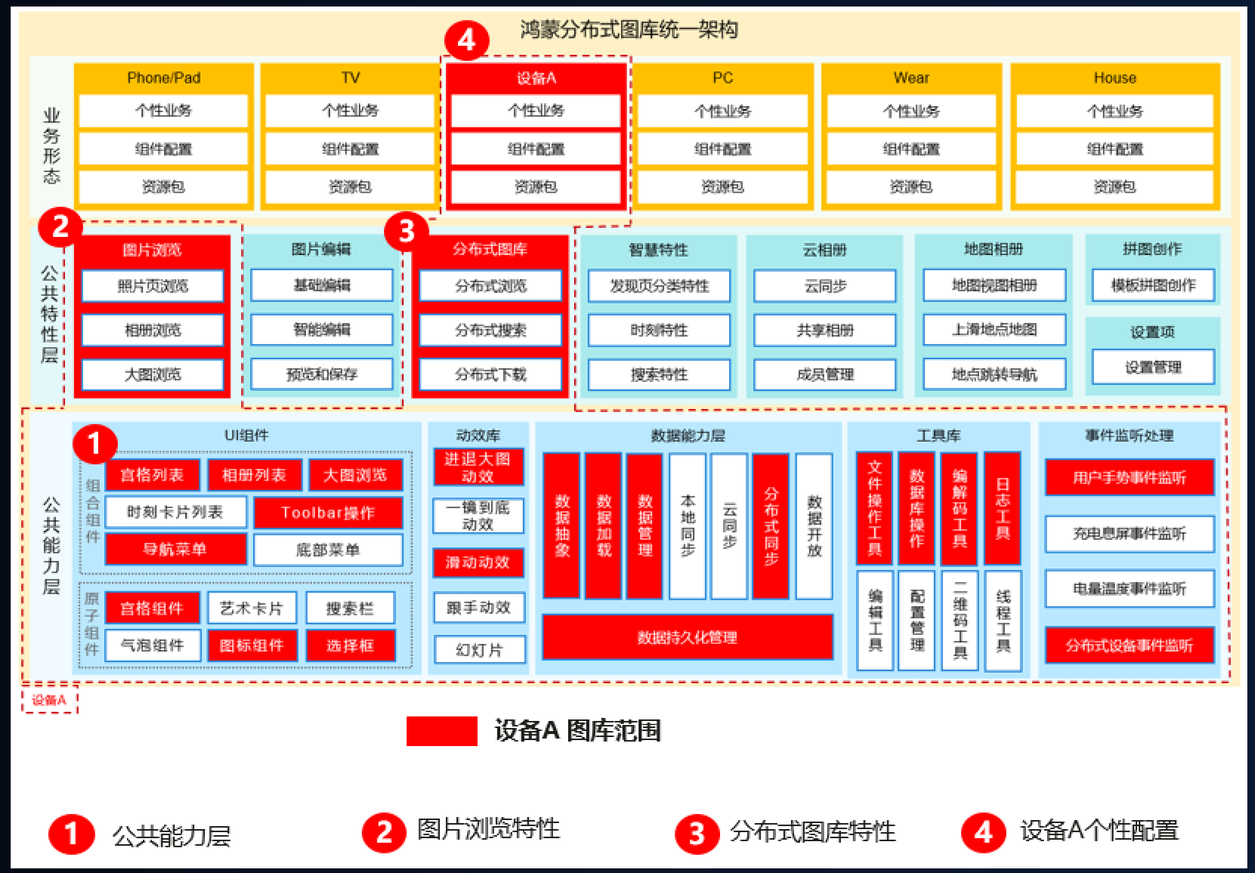
图库最小系统

基于HarmonyOS底座构建应用架构的天生优势

- 声明式编程语言，天生组件化架构
- 原子化程序抽象，天生模块化架构
- 能力下沉子系统，天生分布式架构

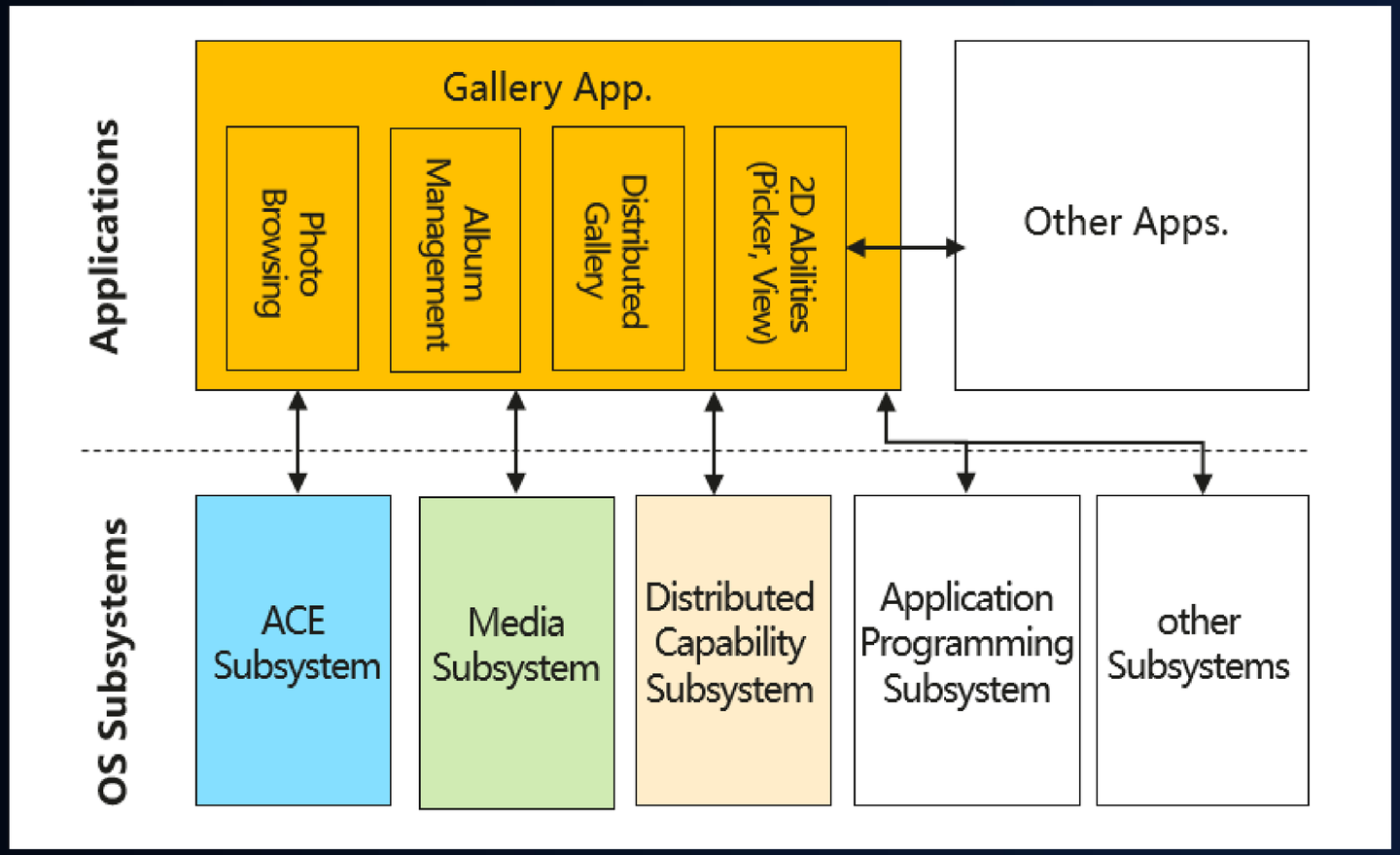
按设备业务需求按需组装发布

最大化重用公共能力层和公共特性层



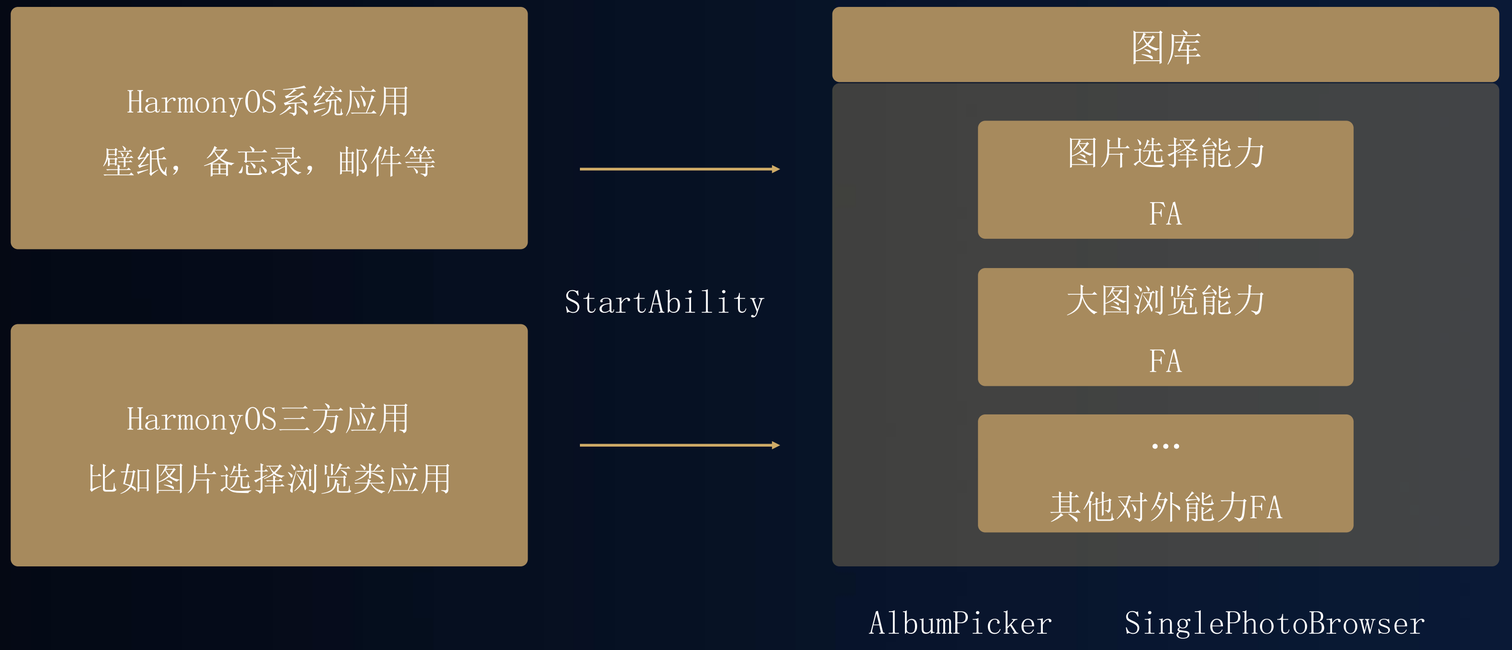
分布式图库对子系统的依赖：

包含ACE子系统、媒体子系统、分布式子系统、应用程序框架等子系统

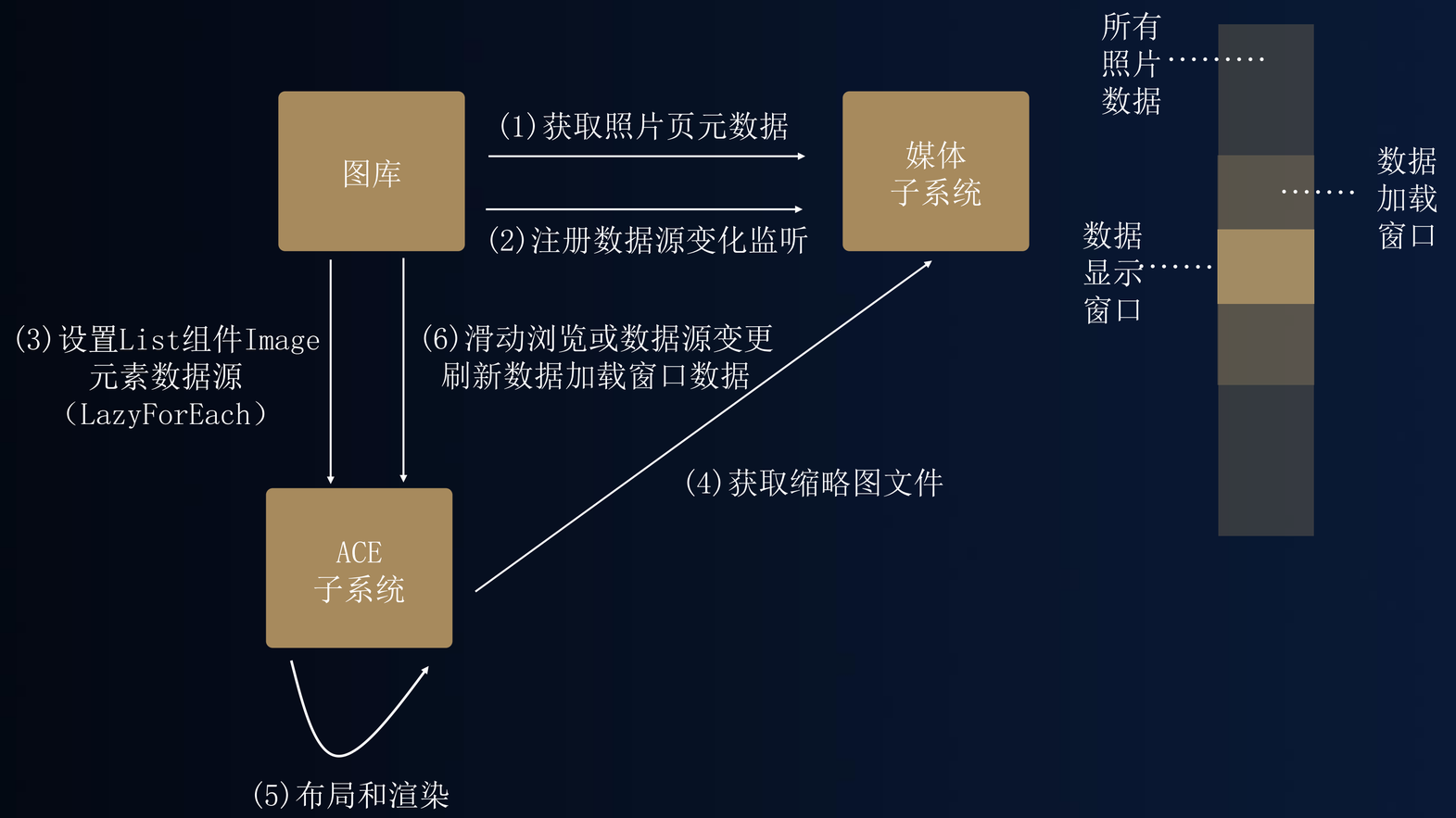


角标 15pt

图库作为HarmonyOS应用子系统的基础核心应用之一
对外提供照片选择、大图浏览等能力



照片页宫格列表 数据加载和显示流程设计



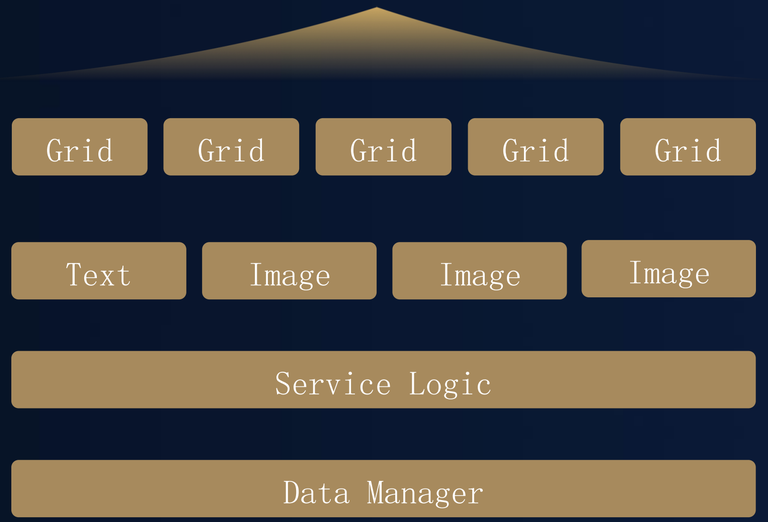
照片页宫格列表 复合组件设计

1. 在应用内多页面之间复用以及跨端复用

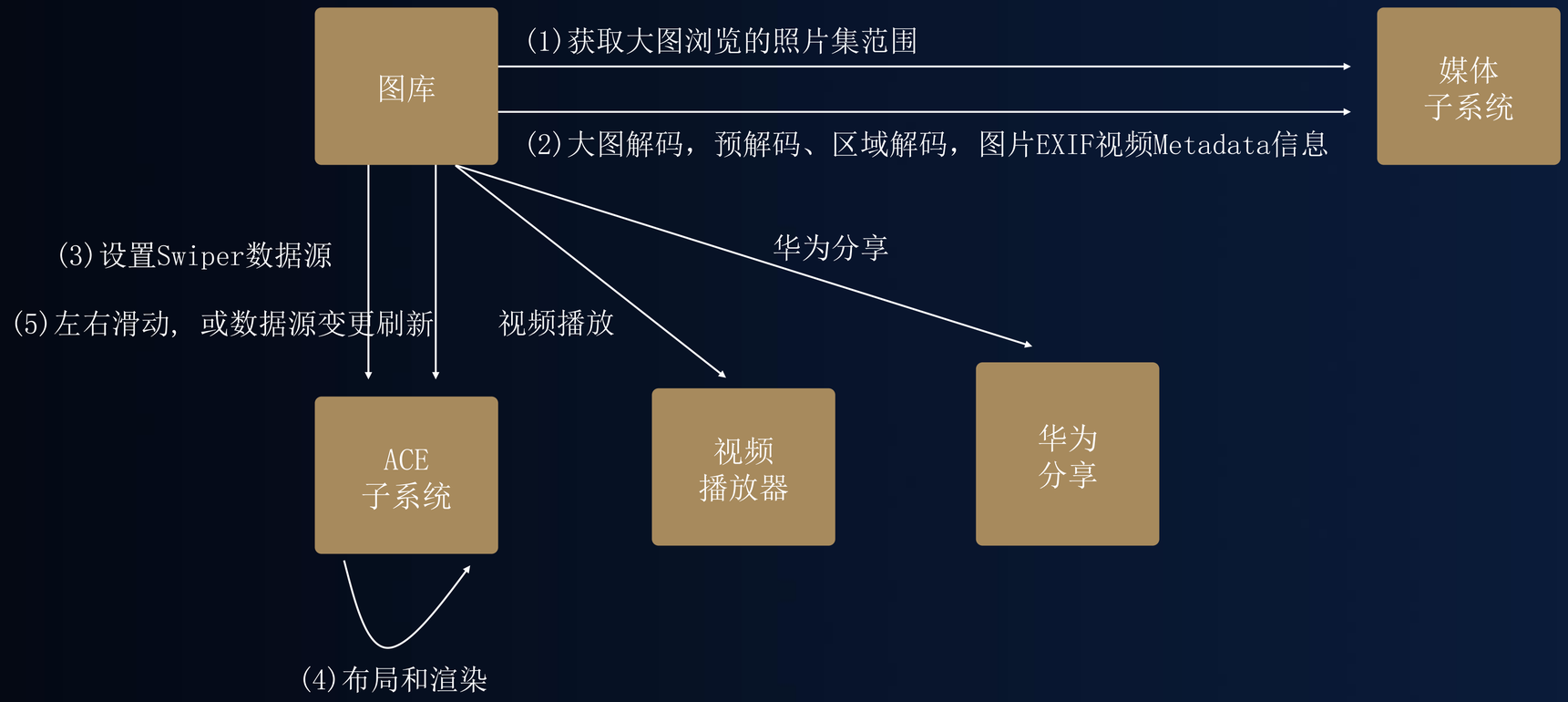
2. 作为照片缩略图宫格列表浏览的载体，在不同设备屏幕上具有共同的本质特征，其不同之处为宫格大小，间距，圆角等，通过实现灵活可配置的组件，可在多端复用

照片页宫格浏览

Grids List 复合组件



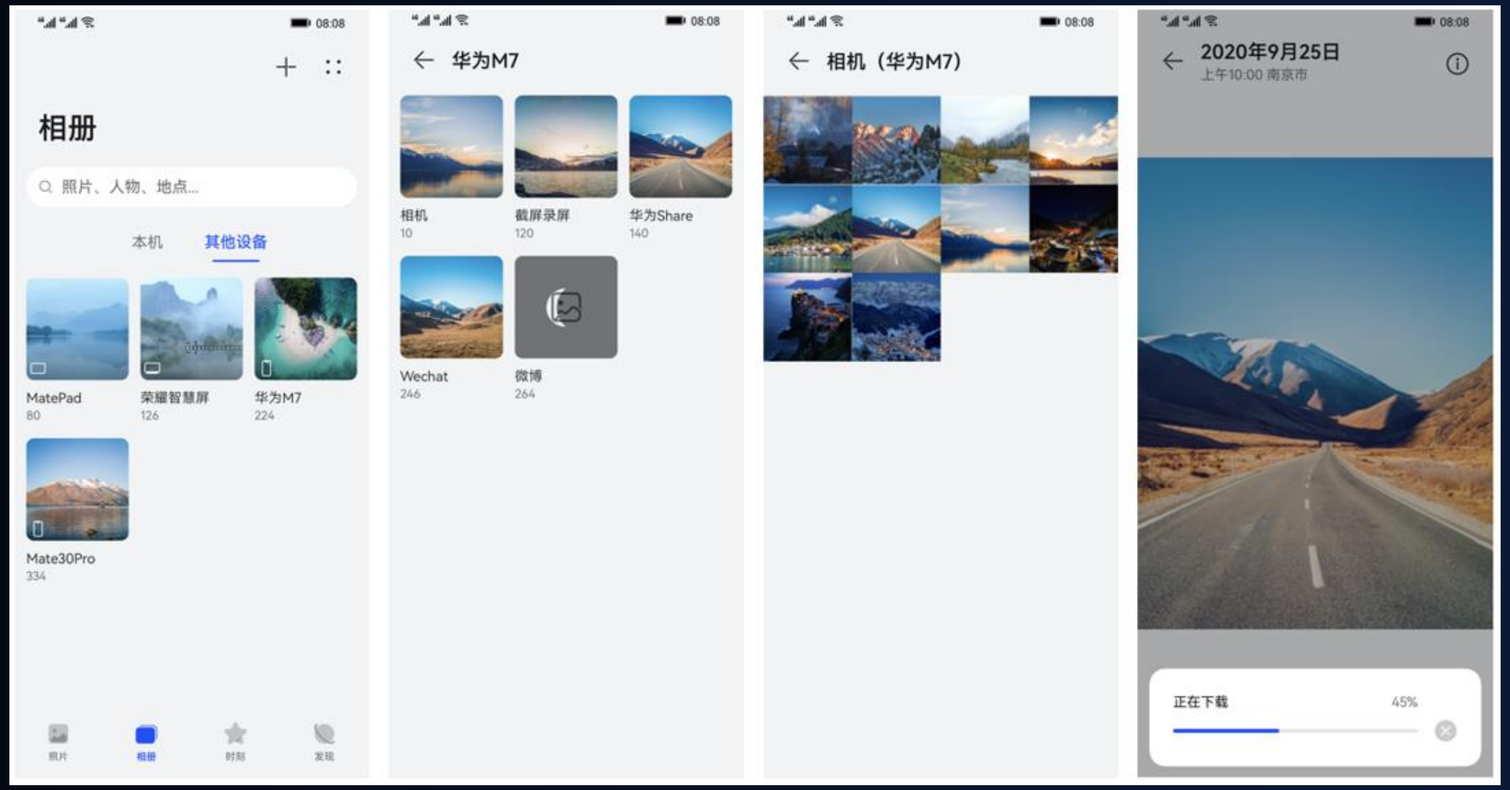
大图浏览 数据加载和显示流程设计



大图浏览 复合组件设计

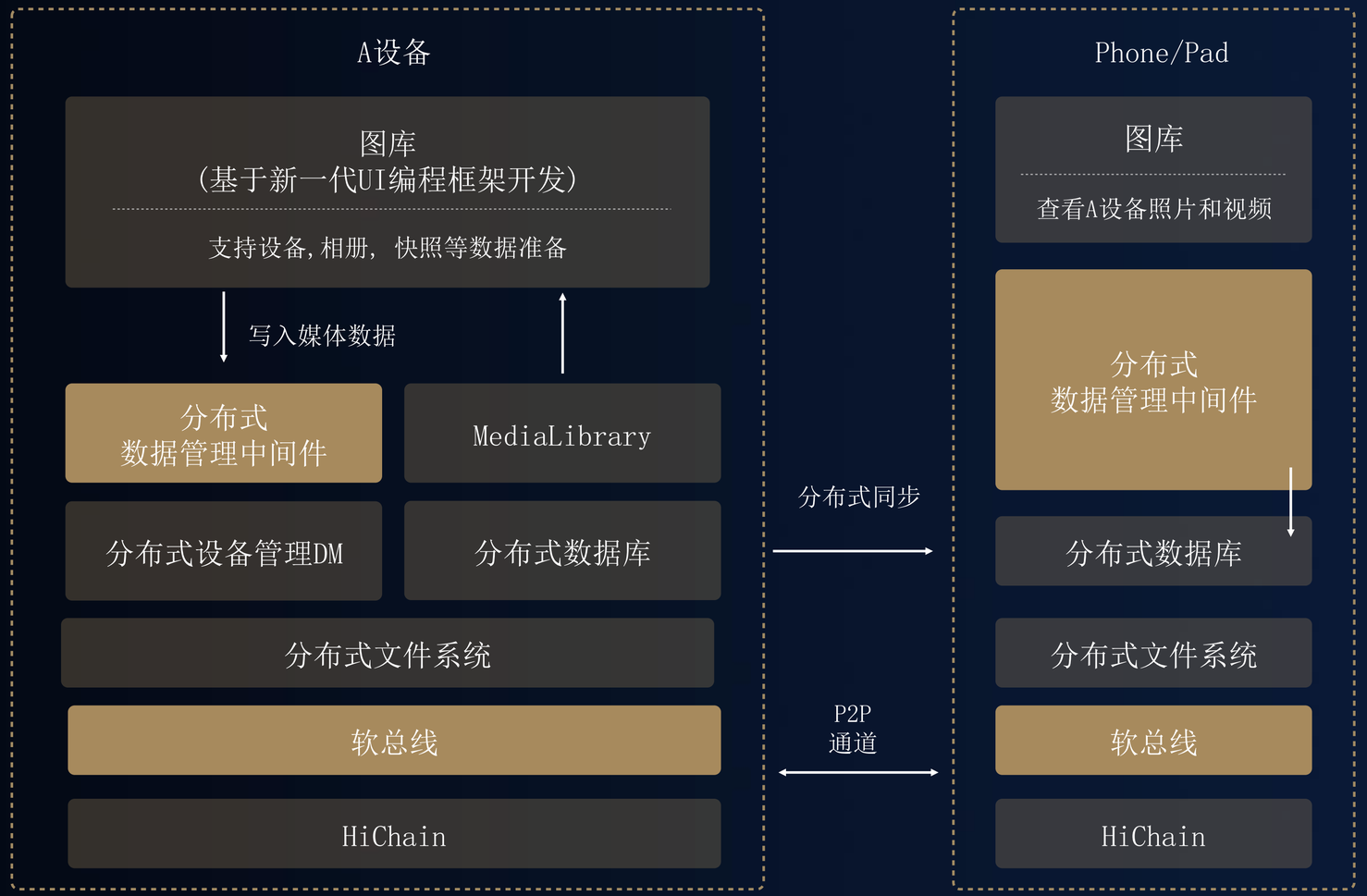
1. 在应用内多页面之间复用以及跨端复用:
2. 作为单张图片或多张图片大图浏览的载体，在不同设备屏幕上都有全屏浏览，左右滑动浏览上下一张照片，支持缩放查看等共同特征，其不同之处有是否有导航预览宫格缩略图栏，缩放方式，沉浸式非沉浸式显示方式等，通过实现灵活可配置的组件，可在多端复用。

分布式媒体资产的浏览和管理中心
支持WIFI AP和P2P接入



分布式子系统依赖：

分布式数据中间件、分布式文件子系统、分布式设备管理、软总线、HiChain等



照片宫格列表界面组件

```
@Component
export struct GridPhoto {
  dataSource: DataSource

  private aboutToAppear(): void {
    this.dataSource = DataSourceFactory.getDataSource()
  }

  build() {
    Flex({
      direction: FlexDirection.Column,
      alignItems: ItemAlign.Start
    }) {
      Stack() {
        Grid() {
          LazyForEach(this.dataSource, (itemBean) => {
            GridItem() {
              ImageComponent({
                item: itemBean
              })
            }
          }, item => JSON.stringify(item))
        }.columnsTemplate(TemplateModel.COLUMN_FOUR_PARAM)
        .columnsGap(UiConstant.PHOTO_SLOT_GAP)
        .rowsGap(UiConstant.PHOTO_SLOT_GAP)
      }
    }
  }

  async prepareData() {...}
}
```

声明式UI,
UI组件化

照片宫格列表数据加载

```
getData(index: number): any {
    // 获取layoutIndex对应的DataIndex
    this.updateSlidingWindow(this.dataIndexes[index], false);
    let result: any = this.getWrappedData(index);
    this.logger.debug(`getData, index: ${index}, data: ${JSON.stringify(result)}`);
    return result;
}
```

```
public updateSlidingWindow(dataIndex: number, isForceUpdate: boolean, requestTime?: number): void {
    /**
      省略若干代码...
    */
    if (isForceUpdate) {
        // 强力刷新时，数据总量可能减少小于窗口尾部,此时需要移动窗口
        if (this.activeEnd > this.mediaCount) {
            newActiveStart = Math.max(0, this.activeStart - (this.activeEnd - this.mediaCount));
            newActiveEnd = newActiveStart + this.windowSize;
            // 默认认为最后的数据被删除，复用重叠窗口数据， 保证删除最后几张数据的场景不会全量刷新
            if (newActiveEnd > this.activeStart) {
                this.dataReuse(newActiveStart, this.activeStart, newActiveEnd);
            }
            this.activeStart = newActiveStart;
            this.activeEnd = newActiveEnd;
        }
        // 窗口内可能存在脏数据，需要重新查询整个窗口内的数据
        requestStart = this.activeStart;
        requestCount = this.windowSize;
    } else {
        // 以dataIndex作为窗口中心进行调整，先计算出新的activeStart， 窗口将以步长大小进行移动
        newActiveStart = this.getWindowActiveStart(dataIndex, windowCenter);
        newActiveEnd = newActiveStart + this.windowSize;
        requestStart = newActiveStart;
        requestCount = this.windowSize;
        // 窗口向前移动，新窗口的尾部在原窗口内，复用重叠窗口数据
        if (newActiveEnd < this.activeEnd && newActiveEnd > this.activeStart) {
            requestCount = this.activeStart - newActiveStart;
            this.dataReuse(newActiveStart, this.activeStart, newActiveEnd);
        }
        // 窗口向后移动，新窗口的头部在原窗口内，复用重叠窗口数据
        if (newActiveStart > this.activeStart && newActiveStart < this.activeEnd) {
            requestStart = this.activeEnd;
            requestCount = newActiveEnd - this.activeEnd;
            this.dataReuse(newActiveStart, newActiveStart, this.activeEnd);
        }
        this.activeStart = newActiveStart;
        this.activeEnd = newActiveEnd;
    }
```

数据滑窗预加载
懒加载/按需加载

角标 15pt

大图界面组件

```
@Component
export struct GridPhoto {
  dataSource: DataSource

  private aboutToAppear(): void {
    this.dataSource = DataSourceFactory.getDataSource()
  }

  build() {
    Row() {
      Stack() {
        Swiper() {
          LazyForEach(this.dataSource, (itemBean) => {
            ImageComponent({
              item: itemBean
            })
          }, item => JSON.stringify(item))
        }
      }
    }
  }

  async prepareData() {...}
}
```

声明式UI,
UI组件化

相册数据加载

```
getData(index: number): any {
    if (index < 0 || index >= this.mediaSetList.length) {
        this.logger.error("index out of the total size, index: " + index + " total size: "
            + this.mediaSetList.length);
        return undefined;
    }
    return this.mediaSetList[index];
}

updateAlbumSetData(requestTime: number, mediaSetList: MediaSet[]): void {
    TraceControllerUtils.startTraceWithTaskId("updateAlbumSetData", requestTime);
    this.logger.info("updateMediaItems size: " + mediaSetList.length);
    if (requestTime < this.lastChangeTime) {
        this.logger.info("request data expired, request again!");
        this.loadData();
        return;
    } else {
        this.lastUpdateTime = requestTime;
        this.hasNewChange = false;
    }
    this.mediaSetList = mediaSetList;
    this.logger.info("updateMediaSet call onDataReloaded");
    this.listeners.forEach(listener => {
        listener.onDataReloaded()
    })
    TraceControllerUtils.finishTraceWithTaskId("updateAlbumSetData", requestTime);
    this.eventBus.emit(Constants.ON_LOADING_FINISHED, [this.totalCount()]);
    TraceControllerUtils.finishTrace("AlbumSetPageInitData");
}
```

数据按需预加载

对外选择界面组件

```
build() {
    Flex({
        direction: FlexDirection.Column,
        justifyContent: FlexAlign.Start,
        alignItems: ItemAlign.Start
    }) {
        ActionBar({isShowBar: true, actionBarProp: this.createActionBar(), onMenuClicked: this.onMenuClicked})
        Stack() {
            if (this.isEmpty) {
                NoPhotoComponent({title: $r('app.string.title_no_albums')})
            }
            Grid() {
                LazyForEach(this.albums, (item) => {
                    GridItem() {
                        ThirdAlbumGridItem({
                            item: item,
                            isMultiPick: this.isMultiPick
                        })
                    }
                }, item => item.name)
            }
```

声明式UI,
UI组件化

页面配置

```
"js": [
  {
    "pages": [
      "pages/index",
      "feature/album/view/AlbumPage",
      "feature/singlephoto/view/PhotoBrowser",
      "feature/select/view/ThirdSelectAlbumSetPage",
      "feature/select/view/ThirdSelectPhotoBrowser",
      "feature/select/view/ThirdSelectAlbumPage"
    ],
    "name": "default",
    "window": {
      "designWidth": 720,
      "autoDesignWidth": false
    },
```

特性按page解耦

扫码参加1024程序员节

〈解锁HarmonyOS核心技能，赢取限量好礼〉

开发者训练营

Codelabs 挑战赛

HarmonyOS技术征文

HarmonyOS开发者创新大赛

扫码了解1024更多信息

报名参加HarmonyOS开发者创新大赛

谢谢

欢迎访问HarmonyOS开发者官网

欢迎关注HarmonyOS开发者微信公众号