@Builder装饰器：自定义构建函数

自定义组件内自定义构建函数：

```
//定义
@Builder MyBuilderFunction(){ ... }
//使用
MyGlobalBuilderFunction()
```

- 允许在自定义组件内定义一个或多个@Builder方法，该方法被认为是该组件的私有、特殊类型的成员函数。
- 自定义构建函数可以在所属组件的build方法和其他自定义构建函数中调用，但不允许在组件外调用。
- 在自定义函数体中，this指代当前所属组件，组件的状态变量可以在自定义构建函数内访问。建议通过this访问自定义组件的状态变量而不是参数传递

全局自定义构建函数

- 全局的自定义构建函数可以被整个应用获取，不允许使用this和bind方法。
- 如果不涉及组件状态变化，建议使用全局的自定义构建方法。

```
//定义
@Builder function MyGlobalBuilderFunction(){ ... }
//使用
MyGlobalBuilderFunction()
```

传递参数

自定义构建函数的参数传递有按值传递和按引用传递两种

- @Builder通过按引用传递的方式传入参数，才会触发动态渲染UI，并且参数只能是一个。
- @Builder如果传入的参数是**两个或两个以上，不会触发动态渲染UI。**
- @Builder传入的参数中同时包含按值传递和按引用传递两种方式，不会触发动态渲染UI。
- @Builder的参数必须按照对象字面量的形式，把所需要的属性一一传入，才会触发动态渲染UI。

使用全局自定义构建函数

创建全局的@Builder方法，在Column里面使用overBuilder()方式调用，通过以对象字面量的形式传递参数，无论是简单类型还是复杂类型，值的改变都会引起UI界面的刷新。

@LocalBuilder和@Builder区别说明

函数componentBuilder被@Builder修饰时，显示效果是 “Child”，函数componentBuilder被@LocalBuildr修饰时，显示效果是“Parent”。

说明：

@Builder componentBuilder()通过this.componentBuilder的形式传给子组件@BuilderParam customBuilderParam，this指向在Child的label，即“Child”。

@LocalBuilder componentBuilder()通过this.componentBuilder的形式传给子组件@BuilderParam customBuilderParam，this指向Parent的label，即“Parent”。

@BuilderParam

参数初始化组件

@BuilderParam装饰的方法可以是有参数和无参数的两种形式，需与指向的@Builder方法类型匹配。@BuilderParam装饰的方法类型需要和@Builder方法类型一致。

@BuilderParam装饰的方法只能被自定义构建函数（@Builder装饰的方法）初始化。如果在API 11中和@Require结合使用，则必须父组件构造传参。

尾随闭包初始化组件

- 此场景下自定义组件内有且仅有一个使用@BuilderParam装饰的属性。
- 此场景下自定义组件不支持使用通用属性。

在自定义组件中使用@BuilderParam装饰的属性时也可通过尾随闭包进行初始化。在初始化自定义组件时，组件后紧跟一个大括号“{}”形成尾随闭包场景。

wrapBuilder：封装全局@Builder

wrapBuilder是一个模板函数，返回一个WrappedBuilder对象。

- **wrapBuilder必须传入被@Builder修饰的全局函数。**
- **重复定义wrapBuilder失效**

```
declare function wrapBuilder< Args extends Object[]>(builder: (...args: Args) =>
void): WrappedBuilder;
```

模板参数Args extends Object[]是需要包装的builder函数的参数列表

e.g.

```
let builderVar: WrappedBuilder<[string, number]> = wrapBuilder(MyBuilder)
let builderArr: WrappedBuilder<[string, number]>[] = [wrapBuilder(MyBuilder)] //可以放入数组
```

作用1 使用场景2

自定义组件Index使用ForEach来进行不同@Builder函数的渲染，可以使用builderArr声明的wrapBuilder数组进行不同@Builder函数效果体现。整体代码会较整洁。

@Styles装饰器：定义组件重用样式

- 当前@Styles仅支持
- 通用属性和通用事件。
- @Styles方法不支持参数
- @Styles可以定义在组件内或全局，在全局定义时需在方法名前面添加function关键字，组件内定义时则不需要添加function关键字。

说明

只能在当前文件内使用，不支持export

```
// setAttribute.ets
export class MyButtonModifier implements AttributeModifier<ButtonAttribute> {
  isDark: boolean = false
  applyNormalAttribute(instance: ButtonAttribute): void {
    if (this.isDark) {
      instance.backgroundColor(Color.Black)
    } else {
      instance.backgroundColor(Color.Red)
    }
  }
}
```

在@Styles的基础上，我们提供了@Extend，用于扩展原生组件样式

@Extend

对指定组件的属性进行扩展

eg

```
@Extend(Text) function fancy (fontSize: number) {
  .fontColor(Color.Red)
  .fontSize(fontSize)
}
```

1.开发者可以在调用时传递参数
2.@Extend支持封装指定组件的私有属性、私有事件和自身定义的全局方法
3.@Extend仅支持在全局定义，不支持在组件内部定义

stateStyles：多态样式

stateStyles是属性方法，可以根据UI内部状态来设置样式，类似于css伪类，但语法不同。ArkUI提供以下五种状态：

- focused：获焦态。
- normal：正常态。
- pressed：按压态。
- disabled：不可用态。
- selected

10+：选中态。

@AnimatableExtend装饰器：定义可动画属性

@AnimatableExtend装饰器用于自定义可动画的属性方法，在这个属性方法中修改组件不可动画的属性。在动画执行过程时，通过逐帧回调函数修改不可动画属性值，让不可动画属性也能实现动画效果。也可通过逐帧回调函数每帧修改可动画属性的值，实现逐帧布局的效果。

- 可动画属性：如果一个属性方法在animation属性前调用，改变这个属性的值可以生效animation属性的动画效果，这个属性称为可动画属性。比如height、width、backgroundColor、translate属性，Text组件的fontSize属性等。
- 不可动画属性：如果一个属性方法在animation属性前调用，改变这个属性的值不能生效animation属性的动画效果，这个属性称为不可动画属性。比如Polyline组件的points属性等。

- @AnimatableExtend仅支持定义在全局，不支持在组件内部定义。
- @AnimatableExtend定义的函数参数类型必须为number类型或者实现 AnimatableArithmetic<T>接口的自定义类型。
- @AnimatableExtend定义的函数体内只能调用@AnimatableExtend括号内组件的属性方

@Require装饰器：校验构造传参

@Require是校验@Prop、@State、@Provide、@BuilderParam和普通变量(无状态装饰器修饰的变量)是否需要构造传参的一个装饰器。

@Require=必传参数