Flutter生命周期

小冰山口

StatefulWidget生命周期

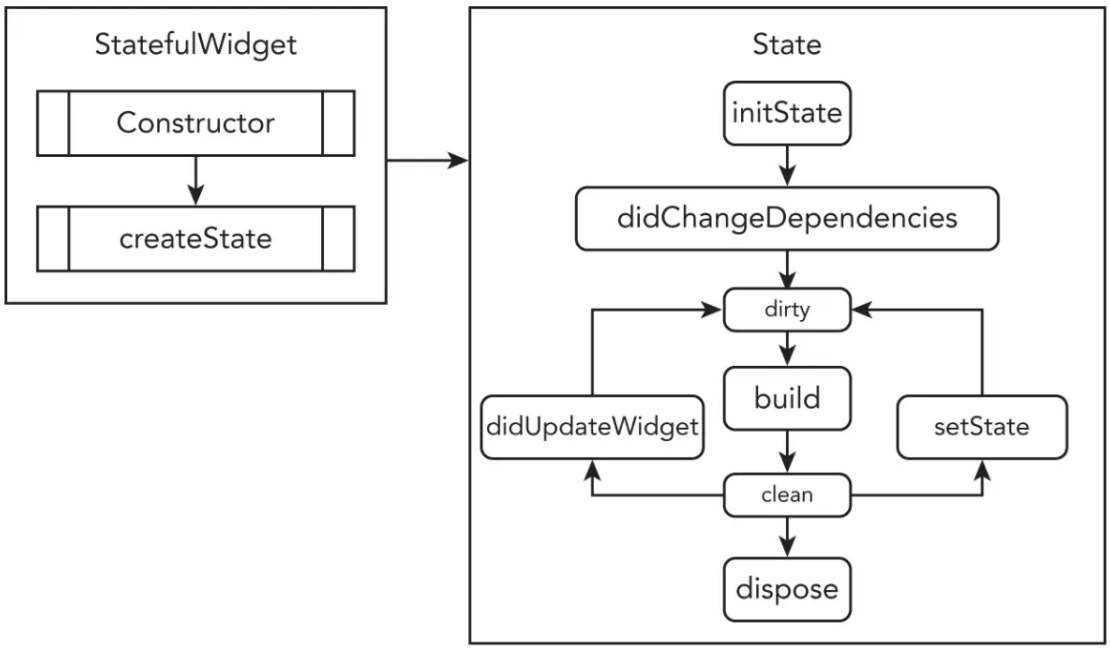

image.png

- 执行顺序, 从上往下

  createState : 创建State, 只执行1次

  initState: 初始State, mounted等于true, 只执行1次

  didChangeDependencies: 父或祖先widget中的InheritedWidget改变时会被调用

  build: UI被重新渲染的时候多次执行

  addPostFrameCallback: 渲染结束回调, 只执行1次

  didUpdateWidget: 父类setState后, 子类就会触发

  deactive: 从组件树中移除State时调用

`dispose`: 组件被释放时调用

不要在`build`里面更新状态, 影响性能

- `initState` 初始状态数据:

```
@override
void initState() {
  super.initState();
  print('初始状态数据');
}
```

- `build`渲染视图, 可多次

```
@override
Widget build(BuildContext context) {
  return Column(
    children: [
      ElevatedButton(
        onPressed: () {
          setState(() {
            imageUrl = imageUrl == img1 ? img2 : img1;
          });
        },
        child: const Text('切换图片'),
      ),
      ImageWidget(imgUrl: imageUrl),
      ImageWidget2(imgUrl: imageUrl),
    ],
  );
 }
}
```

- `mounted` 状态

flutter分配完你的组件树位置, 会设置mounted为true

你需要在mounted == true 情况下, 调用setState()来更新UI, 这才是安全的操作

```
      ElevatedButton(
        onPressed: () {
          if (mounted) {
            setState(() {
              imageUrl = imageUrl == img1 ? img2 : img1;
            });
          }
        },
        child: const Text('切换图片'),
      ),
```

- didChangeDependencies

父或祖先widget中的InheritedWidget改变时会被调用

InheritedWidget和React中的context功能类似, 和逐级传递数据相比, 它们能实现组件跨级传递数据. 这种从根开始向下传递数据的方式, 很适合做全局数据的管理, 如样式, 基础数据等

```
@override
void didChangeDependencies() {
  super.didChangeDependencies();
  print('父或祖先widget中的InheritedWidget改变时会被调用');
}
```

- didUpdateWidget: 父类setState后, 子类就会触发

```
@override
void didUpdateWidget(covariant BannerWidget oldWidget) {
```

```
    super.didUpdateWidget(oldWidget);
    print('父类setState后, 子类就会触发 $oldWidget');
  }
```

ps: 好像没调用过

- addPostFrameCallback: 渲染结束回调, 只执行1次, 我们可以放在initState时设置

```
  @override
  void initState() {
    super.initState();

SchedulerBinding.instance.addPostFrameCallback((timeStamp)
{
      print('渲染结束调用, 只执行1次');
      print(timeStamp);
    });
  }
```

- deactive: 从组件树中移除State时调用

```
  @override
  void deactivate() {
    super.deactivate();
    print('从组件树中移除state时调用');
  }
```

- dispose: 组件被释放时调用

```
  @override
  void dispose() {
    super.dispose();
```

```
    print('组件被释放时调用');
  }
```

`StatelessWidget`生命周期

无状态组件, 不需要处理生命周期, 直接显示即可

在源码中可见`createElement()`创建组件到组件树, 不需要重写去维护

`App`生命周期

- 第一步: 创建`StatefulWidget`组件, 混入`WidgetsBindingObserver`

```
class _BannerWidgetState extends State<BannerWidget> with
WidgetsBindingObserver
```

- 第二步: 添加观察者`addObserver`

```
  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addObserver(this);
  }
```

- 生命周期变化回调: `didChangeAppLifecycleState`

```
@override
  void didChangeAppLifecycleState(AppLifecycleState state)
{
    super.didChangeAppLifecycleState(state);
    print('生命周期变化时回调');
```

```
    switch (state) {
      case AppLifecycleState.resumed:
        print('应用可见并可响应用户操作，app进入前台');
        break;
      case AppLifecycleState.inactive:
        print('用户可见，但不可响应用户操作，比如来了个电话，前后台
切换的过渡状态');
        break;
      case AppLifecycleState.paused:
        print('已经暂停了，用户不可见，不可操作，app进入后台');
        break;
      case AppLifecycleState.detached:
        print('detach');
        break;
      case AppLifecycleState.hidden:
        print('hidden');
        break;
      default:
        break;
    }
  }
```

- 当前系统改变了一些访问性活动的回调

```
  @override
  void didChangeAccessibilityFeatures() {
    super.didChangeAccessibilityFeatures();
    print('当前系统改变了一些访问性活动回调');
  }
```

- 低内存回调

```
  @override
  void didHaveMemoryPressure() {
```

```
  super.didHaveMemoryPressure();
  print('低内存回调');
}
```

- 用户本地设置变化时调用, 如系统语言改变

```
@override
void didChangeLocales(List<Locale>? locales) {
  super.didChangeLocales(locales);
  print('用户本地设置变化时回调, 如系统语言改变');
}
```

- 应用尺寸改变时回调, 例如旋转

```
@override
void didChangeMetrics() {
  super.didChangeMetrics();
  print('应用尺寸改变时回调, 例如旋转');
}
```

- 系统切换主题时回调

```
@override
void didChangePlatformBrightness() {
  super.didChangePlatformBrightness();
  print('系统切换主题时回调');
}
```

- 文字系数变化

```
@override
void didChangeTextScaleFactor() {
  super.didChangeTextScaleFactor();
  print('文字系数变化');
```

```
}
```

- 销毁观察者

```
@override
void dispose() {
  super.dispose();
  // 销毁观察者
  WidgetsBinding.instance.removeObserver(this);
}
```