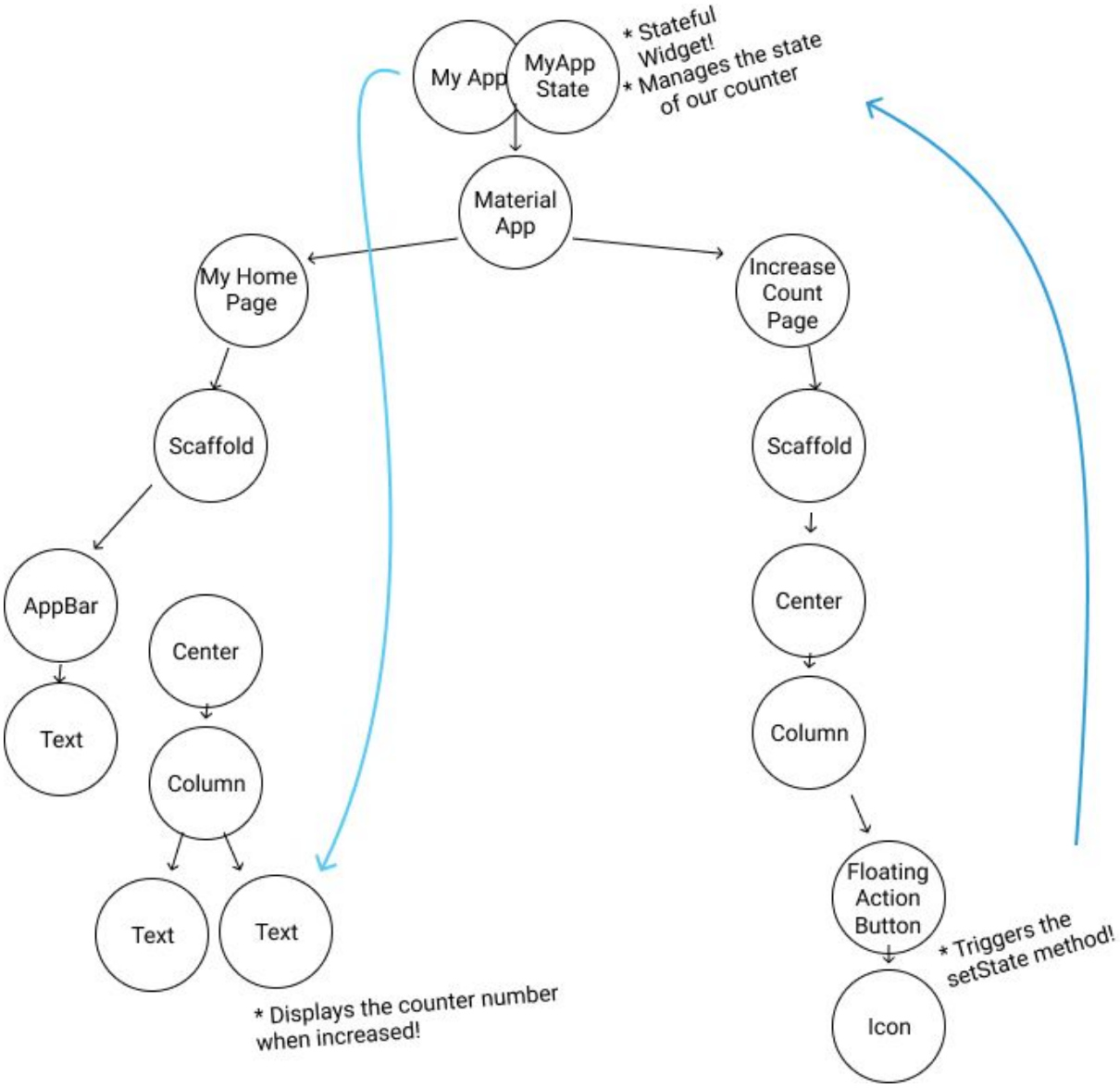
Provider 使用介绍

1. Provider 是什么，能帮助我们解决哪些问题



整个Flutter都是widget 组成，单独页面数据共享很好解决，通过构造函数和回调就能解决；但是如果多页面数据同步就会异常的麻烦； 比如我们的需要获取App主题颜色，字体大小，同时还需要监听他们的改变；

单独页面的状态管理我们知道有StatefulWidget ； 然后通过 setState进行数据刷新；

Provier 包装的是**InheritedWidget**，是我们的代码更容易使用和复用；如果我们使用**InheritedWidget** 每次都创建一个他的子类共享和管理数据；Provider 能够解决跨页面的数据问题，同时可以控制页面的刷新的粒度；

Provier 几个不错的好处（官方）

1. 简单 创建 / 销毁
2. 延迟加载
3. 大大减少每次创建新类模版的时间
4. 非常友好配套调试工具（暂时未研究）
5. 封装了比较简单通用好理解方法去消费**InheritedWidget** 中的数据；
    1. `Provider.of`
    2. `Consumer`
    3. `Selecor`
    4. `context.watch`
    5. `context.read`

name	description
Provider	最基础的Provider，可以提供Value类型数据共享
ListenableProvider	特定的Listenable 监听对象. ListenableProvider 将要监听对象的改变，并且询问小组件依赖他进行重建，当他的调用者在被调用时候
ChangeNotifierProvider	与ListenableProvider区别在于，当需要的时候，会自动调用dispose.
ValueListenableProvider	与ListenableProvider区别在于，仅仅支持value方式获取状态。
StreamProvider	与ListenableProvider区别在于，仅仅支持value方式获取状态。
FutureProvider	与ListenableProvider区别在于，仅仅支持value方式获取状态。

使用介绍

`ChangeNotifierProvider`

1. 使用注意⚠️

```
MyChangeNotifier variable;
ChangeNotifierProvider.value(
  value: variable,
  child: ...
)


ChangeNotifierProvider.value(
  create: MyModel(),
  child: ...
)
```

MultiProvider

```
MultiProvider(
  providers: [
    Provider<Something>(create: (_) => Something()),
    Provider<SomethingElse>(create: (_) => SomethingElse()),
    Provider<AnotherThing>(create: (_) => AnotherThing()),
  ],
  child: someWidget,
)
```

消费数据

非常简单的方式读取 `Provider` 中的数据, 通过BuildContext 中的扩展方法： `Consumer` `watch`， `read`， `selector`

1. `Consumer` `Consumer2<A, B>` `Consumer3<A, B>`

```
Widget build(BuildContext context) {
  return ChangeNotifierProvider(
    create: (_) => Foo(),
  child: Text(Provider.of<Foo>(context).value),
  );
}

Widget build(BuildContext context) {
      return ChangeNotifierProvider(
        create: (_) => Foo(),
        child: Consumer<Foo>(
          builder: (_, foo, __) => Text(foo.value),
        },
      );
    }
```

2. `context.watch<T>()`，监听小部件上T数据的改变

```
extension WatchContext on BuildContext {
  T watch<T>() {
    /// 隐藏了部分代码
   return Provider.of<T>(this);
  }
}
```

3. `context.read<T>()` 不会监听数据的改变

```
/// Exposes the [read] method.
extension ReadContext on BuildContext {
  T read<T>() {
     /// 隐藏了部分代码
    return Provider.of<T>(this, listen: false);
  }
}
```

4. `context.select<T, R>(R cb(T value))` 容许组件监听一小部分数据T的改变

   或者使用静态方法 `Provider.of<T>(context)`，使用的方式和 `watch` / `read` 相同

```
class Home extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text(
      // Don't forget to pass the type of the object you want to obtain to `watch`!
      context.watch<String>(),
    );
  }
}
```

什么情况下应该使用哪种Provider?

1. 不需要监听状态的改变，子组件不需要获取父组件的状态，使用 `Provider` 就足够

2. 需要共享数据，并且监听数据的变化，可以使用 `ListenableProvider` 或者 `ChangeNotifierProvider`，如果需要自主管理数据的 `dispose`，建议使用 `ListenableProvider`

3. 如果有两种数据类型，或者有多种数据依赖关系，使用ProxyProvider系列0

```
/// 创建单一的[Provider]
  /// 使用这个方法时，需要在main.dart中添加 Provider.debugCheckInvalidValueType = null;
  static Provider createProvider<T>(T t) {
    return Provider<T>(
      create: (BuildContext c) => t,
```

```
    );
  }

  /// 创建单一的[ListenableProvider]
  static ListenableProvider createListenableProvider<T extends Listenable>(
      T t) {
    return ListenableProvider<T>(
      create: (BuildContext c) => t,
    );
  }
```

2. ## Provider的原理是什么

InheritedWidget

`InheritedWidget`是Flutter中非常重要的一个功能型组件，它提供了一种数据在widget树中从上到下传递、共享的方式，比如我们在应用的根widget中通过`InheritedWidget`共享了一个数据，那么我们便可以在任意子widget中来获取该共享的数据！这个特性在一些需要在widget树中共享数据的场景中非常方便！如Flutter SDK中正是通过InheritedWidget来共享应用主题（`Theme`）和Locale (当前语言环境)信息的。

```
class ShareDataWidget extends InheritedWidget {
  ShareDataWidget({
    @required this.data,
    Widget child
  }) :super(child: child);

  final int data; //需要在子树中共享的数据，保存点击次数

  //定义一个便捷方法，方便子树中的widget获取共享数据
  static ShareDataWidget of(BuildContext context) {
    return context.dependOnInheritedWidgetOfExactType<ShareDataWidget>();
    //return context.getElementForInheritedWidgetOfExactType<ShareDataWidget>().widget;
  }

  //该回调决定当data发生变化时，是否通知子树中依赖data的Widget
  @override
  bool updateShouldNotify(ShareDataWidget old) {
    //如果返回true，则子树中依赖(build函数中有调用)本widget
    //的子widget的`state.didChangeDependencies`会被调用
    return old.data != data;
  }
}
```

1. ## Provider 在项目中的使用

1. ## Provider 在MVVM中的实践

参考