

Flutter 的状态管理

当你开发 Flutter 时，你肯定会遇到需要在不同页面共享状态的情况，这时候就需要对状态进行管理。

从广义上来讲，Flutter app 的状态是 app 运行时内存中存的所有内容，包括：app 的资源、Flutter 框架里保存的有关 UI、动画状态、纹理、字体等的所有变量。但是这里的状态并不全都需要我们来管理，比如纹理，是 Flutter 框架来帮你管理的，所以我们更关心的状态，是那些在创建 UI 时需要的数据，这部分数据分为了两部分：

1. 本地状态
2. 全局状态

1. 本地状态

当一个状态的作用范围在只在一个 Widget 里，其他 Widget 不需要关心这个状态时，这种状态就是本地状态。

例如 BottomNavigationBar 里的 `_index` 字段，`_index` 用来保存当前选中的项：

```
class MyHomepage extends StatefulWidget {
  @override
  _MyHomepageState createState() =>
  _MyHomepageState();
}

class _MyHomepageState extends State<MyHomepage>
{
  int _index = 0;

  @override
  Widget build(BuildContext context) {
    return BottomNavigationBar(
      currentIndex: _index,
      onTap: (newIndex) {
        setState(() {
          _index = newIndex;
        });
      },
      // ... items ...
    );
  }
}
```

这里的 `_index` 就是本地状态，App 的其他部分完全不会关心当前在哪个页面，所以也不需要 `_index` 的值，只有 `_MyHomepageState` 关心 `_index` 的值，所以 `_index` 只存在于 `_MyHomepageState` 里，而且 `_index` 也不需要存储，因为当你在关闭 `MyHomepage` 后再打开，也不介意 `_index` 重置为0。

所以这里可以看出本地状态的特点，就是：

1. 私有的

2. 暂时的

类似的，还有其他几个本地状态的例子：

- PageView 里表示当前是哪个页面的数据
- 复杂动画的播放进度

2. 全局状态

当状态在 App 的全部或部分使用时，这种状态就是全局状态。

例如用户的登录信息，整个 App 都需要用到，用户的登录信息就是全局状态；又例如电商 App 的购物车数据，在商品页面需要用到，在购物车页面也要用到，在订单页面也需要用到，因为 App 里的好几个部分都要用到购物车数据，所以购物车数据也是全局状态。

全局状态的特点就是：

1. 共享的

类似的，还有其他几个全局状态的例子：

- 用户的数据信息
- 新闻 App 的文章的已读、未读状态数据

如何确定你的状态是本地状态还是全局状态？

要区分本地状态和全局状态，要看状态的作用范围。可以根据下图来划分：

当要确定一个数据是属于本地状态还是全局状态时，先看 这个数据 需要被哪些部分使用，如果只有一个 Widget 使用这个数据，那么这个数据就是本地状态；如果有多个 Widget 使用，那么这个数据就是

全局状态。

同时状态也不是一成不变的，可能随着 App 越来越复杂，原来是本地状态的会变为全局状态，就需要不断重构。比如 `BottomNavigationBar` 的 `_index`，如果需要从外部去更改 `_index`，那么 `_index` 就从本地状态变为全局状态了。

状态管理框架

对本地状态和全局状态管理的框架就叫状态管理框架。

状态管理是一个复杂的主题，而且一直是 Flutter 的热门话题，目前有多种实现状态管理的框架，但一个好的状态管理框架应该具有如下的条件：

- UI 逻辑和业务逻辑应该是分离的
- 在框架的帮助下可以写出高质量的代码
- 框架应该提升 App 的性能
- 框架要容易理解，便于扩展

基于这些条件，我们将探索如下的状态管理框的优劣，从而找到适合的状态管理框：

- `StatefulWidget` 和 `setState()`
- `InheritedWidget`
- Scoped model
- BLoC
- Redux

StatefulWidget 和 setState() 的分析

StatefulWidget 就是有状态的 Widget，其内部管理状态用的就是 setState()。上一篇写的豆瓣电影 APP 用的就是 StatefulWidget 和 setState()，但是有两个很大的缺陷：

1. UI 逻辑和业务逻辑没有分离
2. 只能管理本地状态

如果使用 StatefulWidget 和 setState()，代码会随着 App 的增长，变得越来越难以维护，所以强烈推荐不要使用。

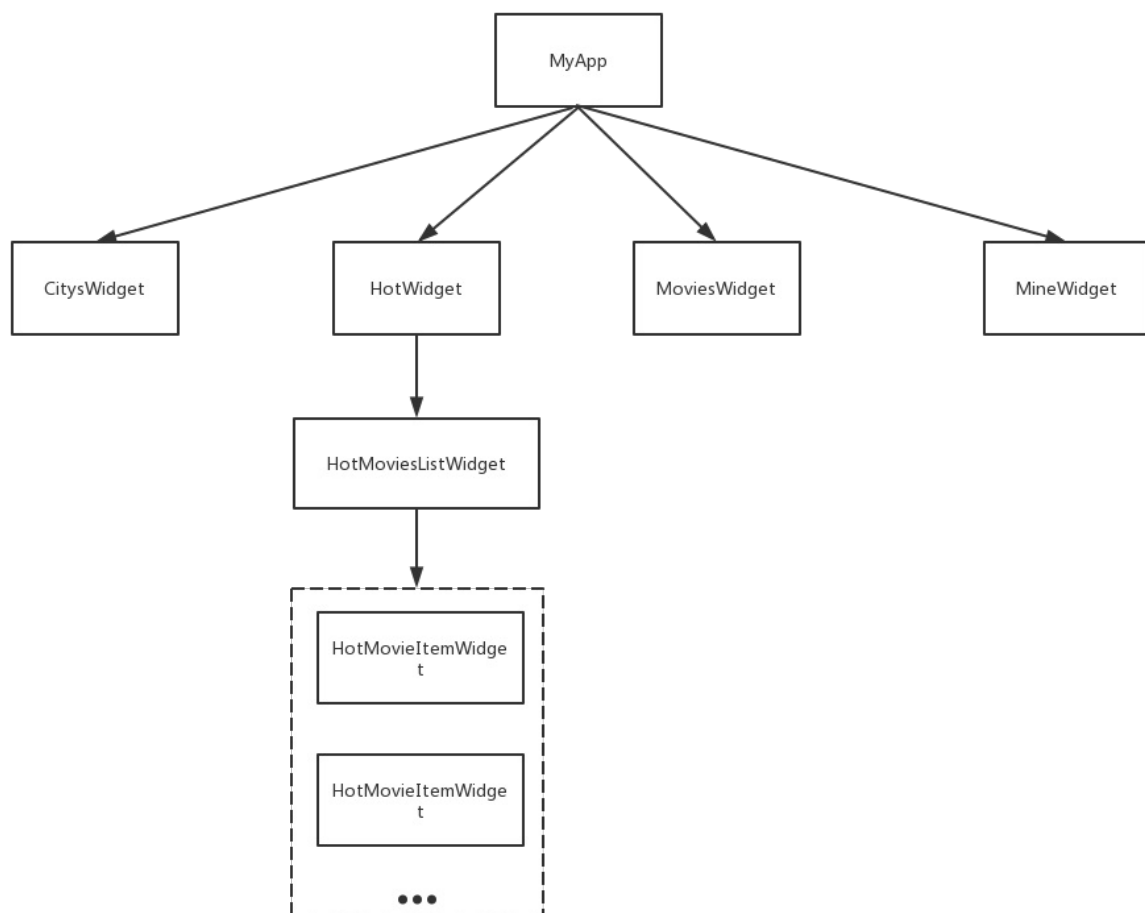
其他状态管理框的分析

为了对其他状态管理框进行分析，就需要使用对应的状态管理框架对豆瓣电影 APP 进行重构

对豆瓣电影APP的状态管理进行重构

在重构之前，我们先列出豆瓣电影 APP 的Widget 树，然后对 Widget 的状态进行分析，看哪些是本地状态，哪些是全局状态。

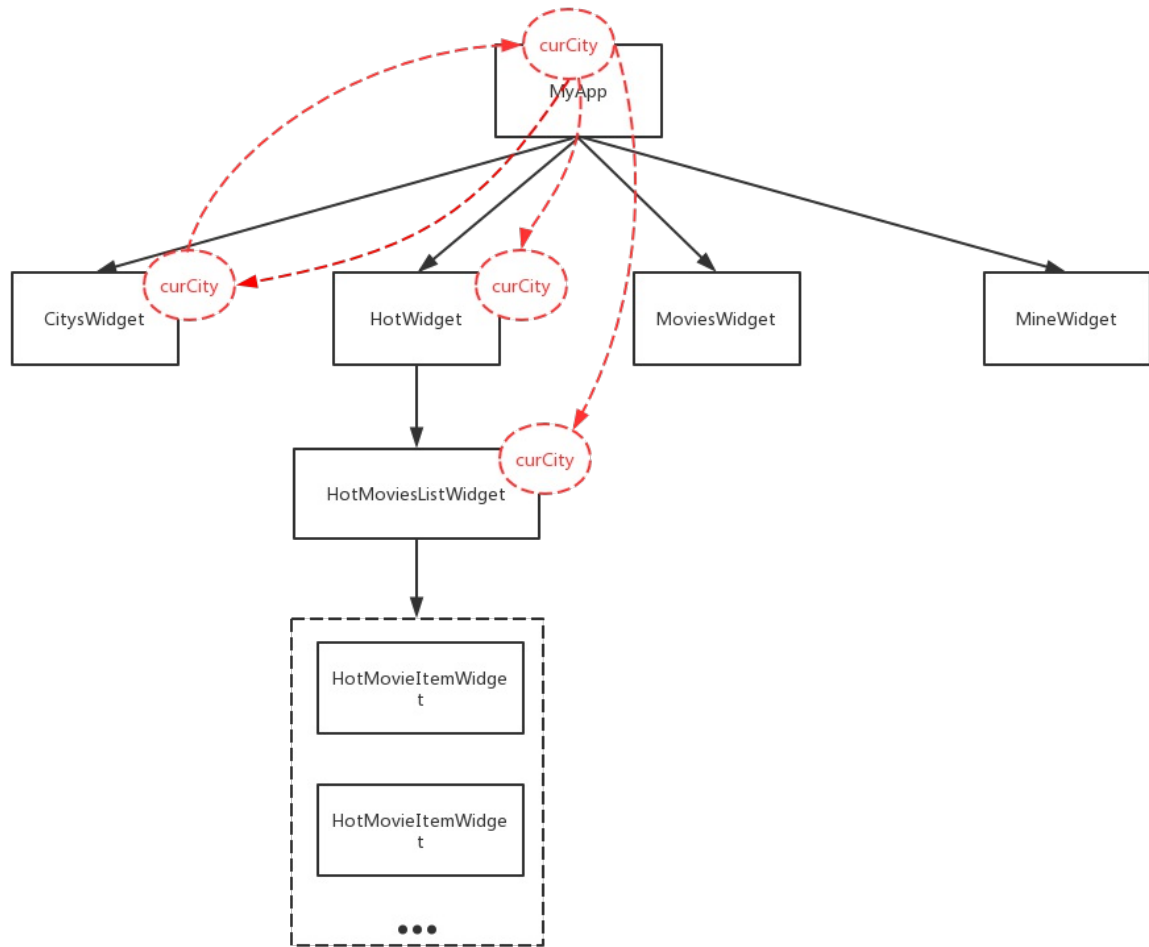
豆瓣电影 APP 的 Widget 树结构为：



可以发现 CitysWidget、HotWidget、HotMovieListWidget 都使用了 curCity 这个数据：

所以，curCity 就是全局状态。

除了区分本地状态和全局状态，还需要弄懂全局状态的流向，因为全局状态是共享的，多个 Widget 都在使用，所以全局状态会受到多个 Widget 的影响。如下是豆瓣电影 APP 的全局状态 curCity 受影响的图：



看箭头，HotWidget 和 MoviesListWidget 只是使用到了 curCity 的值，不会改变 curCity 的值，所以箭头是单向的，这样的关系比较简单，但是在 CitysWidget 里，不仅会用到 curCity 的值，也会改变 curCity 的值，而且当 curCity 的值发生变化时，HotWidget 和 MoviesListWidget 也需要刷新。