Flutter 2
State Management

Oum Saokosal

Master of Engineering in Information Systems, South Korea

012 252 752 (Telegram)

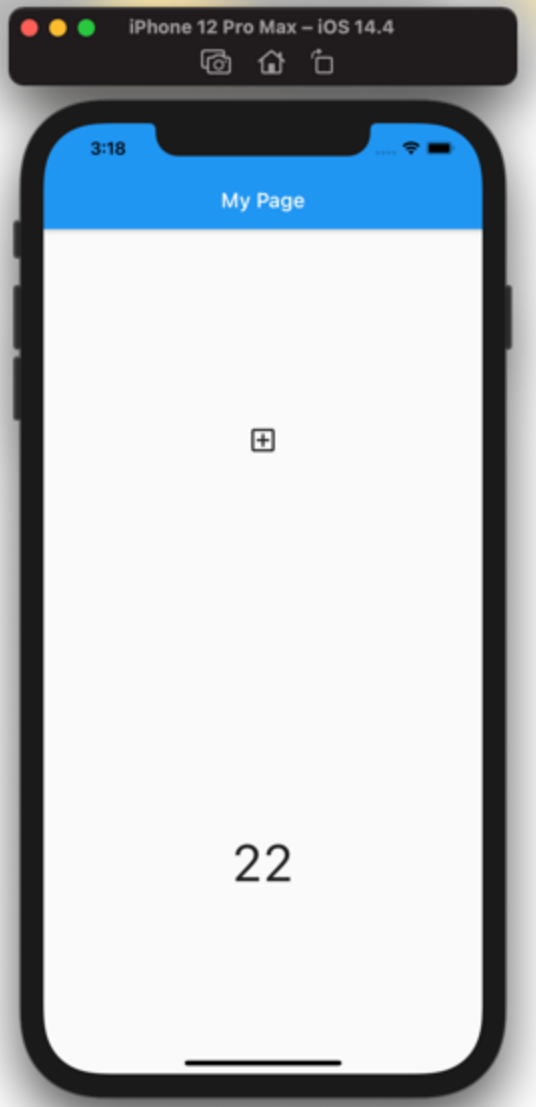
State Management

លំនាំបញ្ហាជាមួយ setState( ) នៅក្នុង Flutter

- នៅក្នុង Flutter វាប្រើ Reactive Programming ដែលក្នុងនោះ ពេលយើងចង់ update ផ្នែកនៃ UI ណាមួយរបស់យើង។ រាល់ពេលយើងចង់ដូរអ្វីមួយ ដូចជាចង់ប្តូរអក្សរ ប្តូរពណ៌ ឬប្តូរ Widget គឺយើងត្រូវហៅ setState( ) ដើម្បីឲ្យវា build នូវ internal state ឡើងវិញ។ ហើយយើងត្រូវប្រើ setState( ) នៅក្នុង class ដែល inherit ចេញពី StatefulWidget។ សូមមើលឧទាហរណ៍នៃការប្រើ setState( )៖

```dart
class MyPage extends StatefulWidget {
  @override
  _MyPageState createState() => _MyPageState();
}

class _MyPageState extends State<MyPage> {
  int _counter = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("My Page"),
      ),
      body: Container(
        child: Column(
          children: [
            _buildButton,
            _buildText,
          ],
        ),
      ),
    );
  }

```

```dart
41
42  Widget get _buildButton{
43    return Expanded(
44      child: Center(
45        child: IconButton(
46          icon: Icon(Icons.add_box_outlined, size: 30,),
47          onPressed: () {
48            setState(() => _counter++);
49          },
50        ),
51      ),
52    );
53  }
54
55  Widget get _buildText {
56    return Expanded(
57      child: Center(
58        child: Text("$_counter", style: TextStyle(fontSize: 50)),
59      ),
60    );
61  }
62  }
63
```



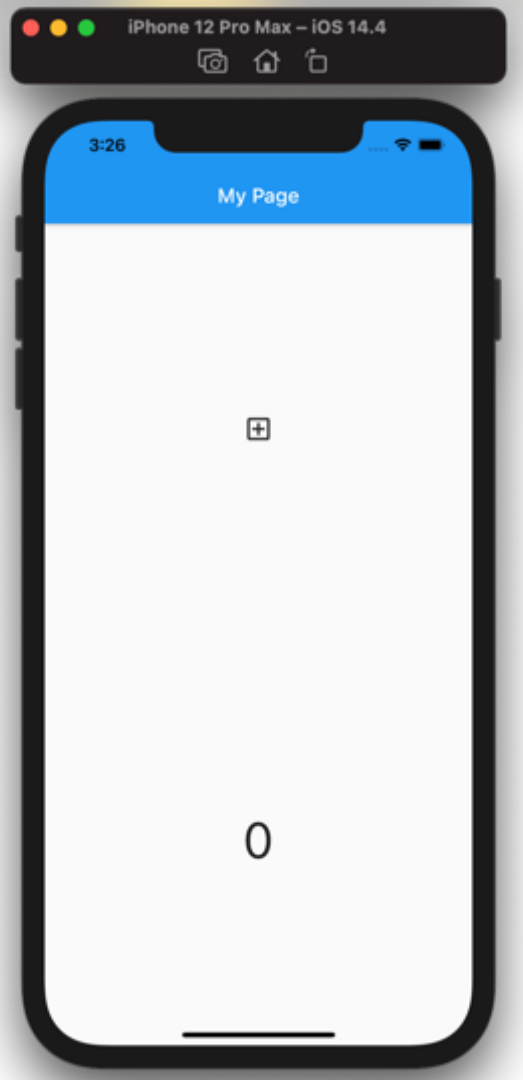
បញ្ហាជាមួយ setState( )

- បើទោះបីជា setState( ) អាចប្តូរ UI បានមែន តែវាអាចប្តូរលុះត្រាតែ UI ទាំងនឹងស្ថិតនៅក្នុង class ជាមួយគ្នា។ **ប៉ុន្តែវាមិនអាចហៅឆ្លង class ផ្សេងគ្នាបានទេ!**

```dart
class MyPage extends StatefulWidget {
  @override
  _MyPageState createState() => _MyPageState();
}

class _MyPageState extends State<MyPage> {
  int _counter = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("My Page"),
      ),
      body: Container(
        child: Column(
          children: [
            MyButton(_counter),
            MyText(_counter),
          ],
        ),
      ),
    );
  }
}
```

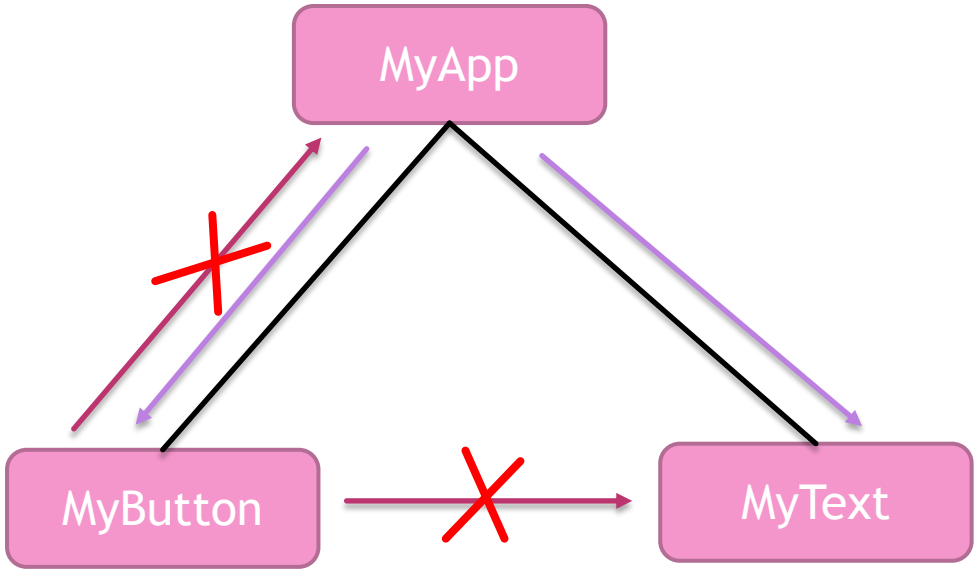
```dart
class MyButton extends StatefulWidget {
  int counter;
  MyButton(this.counter);

  @override
  _MyButtonState createState() => _MyButtonState();
}

class _MyButtonState extends State<MyButton> {
  @override
  Widget build(BuildContext context) {
    return Expanded(
      child: Center(
        child: IconButton(
          icon: Icon(Icons.add_box_outlined, size: 30,),
          onPressed: () {
            setState(() => widget.counter++);
          },
        ),
      ),
    );
  }
}
```

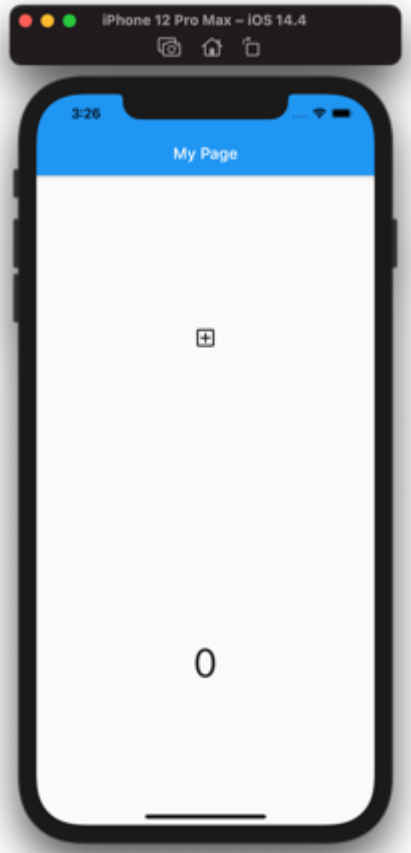
```dart
class MyText extends StatefulWidget {
  final int counter;

  MyText(this.counter);

  @override
  _MyTextState createState() => _MyTextState();
}

class _MyTextState extends State<MyText> {
  @override
  Widget build(BuildContext context) {
    return Expanded(
      child: Center(
        child: Text(
          "${widget.counter}",
          style: TextStyle(fontSize: 50),
        ),
      ),
    );
  }
}
```



Flutter Widgets គឺជា Tree

- App នៅក្នុង Flutter គឺកសាងនៃបណ្តុំនៃ Widget ជាច្រែនគលើគ្នា ហើយមានលក្ខណៈ Tree។ យើងអាចហៅ setState( ) នៅក្នុង class មេ ដើម្បីបញ្ជាអោយប្តូរ UI ក្នុង class កូនបាន។ តែប៉ុន្តែមិនអាចហៅ setState ក្នុងកូនទី១ ហើយទៅដូរ UI ក្នុងកូនទី១បានទេ៖





State Management

- State Management បង្កើតជាចំនុចកណ្តាលមួយនៃគ្រប់គ្រង State។ State Management គឺប្រើជំនួសឲ្យការហៅ setState ពី class ទៅ class មួយទៀត។
- ខាងក្រោមនេះគឺជា package ល្បីៗនៃ State Management៖
  - Provider (https://pub.dev/packages/provider)
  - Redux (https://pub.dev/packages/flutter_redux)
  - Fish-Redux (https://pub.dev/packages/fish_redux)
  - BloC (https://pub.dev/packages/flutter_bloc)
  - GetIt (https://pub.dev/packages/get_it)
  - MobX (https://pub.dev/packages/mobx)
  - Flutter Command (https://pub.dev/packages/flutter_command)
  - Binder (https://pub.dev/packages/binder)
  - GetX (https://pub.dev/packages/get)
  - RiverPod (https://pub.dev/packages/riverpod)

Provider

Provider

Provider គឺជា package មួយដំបូងគេដែលក្រុម Google ណែនាំឲ្យប្រើប្រាស់ក្នុងគ្រប់គ្រង State។ ខាងក្រោមនេះគឺជារបៀបនៃការប្រើប្រាស់៖

1. ដាក់ package Provider ក្នុង pubspec.yaml

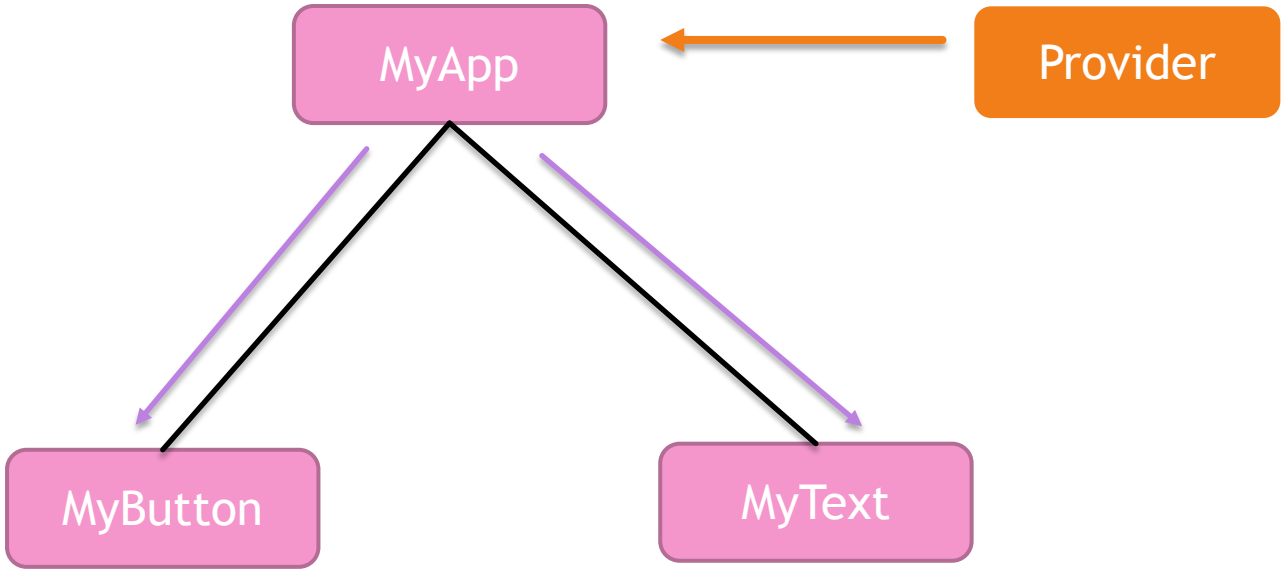
សូមមើល version របស់វាក្នុង https://pub.dev/packages/provider

2. បង្កើត class មួយដែលត្រូវ extends កាន់ទៅ ChangeNotifier។ ចំនុចសំខាន់គឺយើងត្រូវហៅ notifyListeners( ) ដើម្បី Provider ទទួលដឹងពីបំលាស់ប្តូរទិន្នន័យ៖

```dart
import 'package:flutter/foundation.dart';

class CounterLogic extends ChangeNotifier{
  int _counter = 0;
  int get counter => _counter;
  void increase(){
    _counter++;
    notifyListeners();
  }
}
```

Lift-up State (លើក State ឡើងលើ)

3. តាមរយៈរូបខាងក្រោម យើងឃើញហើយថា MyButton និង MyText គឺស្ថិតនៅពីក្រោម MyApp។ ដូច្នេះដើម្បីឲ្យ MyButton អាចឆ្លងឆ្លើយជាមួយ MyText បាន យើងត្រូវលើក State (Lift-up State) មកដាក់ត្រង់ MyApp។ ដូច្នេះយើងត្រូវដាក់ Provider នៅក្នុង class MyApp ត្រង់ចំនុចបែកខ្នែងរវាង MyButton និង MyText៖



```dart
class MyScreen extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("My Page"),
      ),
      body: ChangeNotifierProvider(
        create: (context) => CounterLogic(),
        child: Container(
          child: Column(
            children: [
              MyButton(),
              MyText(),
            ],
          ),
        ),
      ),
    );
  }
}
```

ប្រើ Provider នៅក្នុងWidget កូន ដើម្បីហៅ State

4. នៅក្នុង Widget កូន យើងអាចហៅ State តាមរយៈ Provider៖

```dart
import 'package:provider/provider.dart';

class MyButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {

    final CounterLogic logic = Provider.of<CounteLogic>(context);

    return Expanded(
      child: Center(
        child: IconButton(
          icon: Icon(
            Icons.add_box_outlined, size: 30,
          ),
          onPressed: () => logic.increase(),
        ,
        ),
      ),
    );
  }
}
```

```dart
class MyText extends StatelessWidget {
  @override
  Widget build(BuildContext context) {

    final CounterLogic cm = Provider.of<CounterLogic>(context);

    return Expanded(
      child: Center(
        child: Text("${logic.counter}", style: TextStyle(fontSize: 50)),
      ),
    );
  }
}
```

យើងអាចសង្កេតឃើញ class ទាំង២ MyButton និង MyText គឺបាន extends ចេញពី StatelessWidget។ វាមានន័យ រាល់ការ update ទិន្នន័យ គឺវាមិនបាន build ឡើងវិញទេ។

គួរទាញ variable ឬ method ចេញពី Logic

- នៅក្នុង slide មុននេះ បើទោះជាយើងអាចទាញទិន្នន័យបានមែន ក៏ប៉ុន្តែយើងបានទាញយក CounterLogic ទាំងមូលតែម្តង។ ជំនួសឲ្យការទាញយកទាំងមូលនេះ យើងអាចទាញយកតែ method ឬ variable មួយដែលយើងចង់ប្រើបានហើយ។ យើងប្តូរពីកូដខាងក្រោមនេះ៖

```
final CounterLogic logic = Provider.of<CounterLogic>(context);
logic.increase();
```

ទៅជា៖

```
Provider.of<CounterLogic>(context, listen: false).increase();
```

```
class MyButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
      return Expanded(
       child: Center(
         child: IconButton(
           icon: Icon(Icons.add_box_outlined, size: 30,),
           onPressed: () => Provider.of<CounterLogic>(context, listen: false).increase(),
         ),
        ),
      );
  }
}
```

- ដូចគ្នាទៅនឹងការទាញយក variable គឺយើងគួរទាញវាផ្ទាល់។ ប្លុកពីក្លដខាងក្រោម៖

```
final CounterLogic logic = Provider.of<CounterLogic>(context);
logic.counter;
```

ទៅជា៖

```
int counter = Provider.of<CounterLogic>(context).counter;
```

```
class MyText extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    int counter = Provider.of<CounterLogic>(context).counter;
    return Expanded(
      child: Center(
        child: Text("$counter", style: TextStyle(fontSize: 30)),
      ),
    );
  }
}
```

Consumer Class

- Provider ក៏បានផ្តល់នូវ class មួយឈ្មោះថា Consumer សំរាប់អោយ build UI នៅខាងក្នុងវា៖

```
class MyOtherButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Expanded(
      child: Center(
        child: Consumer<CounterLogic>(
          builder: (context, cm, _){
            return IconButton(
              icon: Icon(
                Icons.add_circle,
                size: cm.counter.toDouble(),
              ),
              onPressed: () => cm.increase(),
            );
          },
        ),
      ),
    );
  }
}
```

Method របស់ context

បន្ទាប់ពីយើងបាន import Provider ហើយ គឺយើងអាចប្រើ method របស់ context បានដែរ ដែល method ទាំងនេះស្រដៀងគ្នានឹង Provider៖

context.read(): វាគ្រាន់តែផ្តល់នូវ Logic តែវាទទួលការផ្លាស់ប្តូរទេ (ប្រើបានតែនៅក្រៅ build)

```
context.read<CounterLogic>().increase();
```

```
Provider.of<CounterLogic>(context, listen: false).increase();
```

context.watch(): ពេលយើងប្រើវា យើងនឹងធ្វើឲ្យ Logic ផ្លាស់ប្តូរ និងទទួលការផ្លាស់ប្តូរមកវិញ (ប្រើបានតែនៅក្នុង build)

```
int counter = context.watch<CounterLogic>().counter;
```

```
int counter = Provider.of<CounterLogic>(context).counter;
```

- **`context.select()`**: ពេលយើងប្រើវាដើម្បីទទួលការផ្លាស់ប្តូរតែផ្នែកតូចមួយណាមួយនៃ Logic តែប៉ុណ្ណោះ (ប្រើបានតែនៅក្នុង build)

```
int counter = context.select<CounterLogic, int>((cm) => cm.counter);
```

```
class MyText extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    int counter = context.select<CounterLogic, int>((cl) => cl.counter);
    return Expanded(
      child: Center(
        child: Text("$counter", style: TextStyle(fontSize: 30)),
      ),
    );
  }
}
```

MultiProvider

- នៅពេលខ្លះ យើងមាន Logic ច្រើនជាងមួយ។ ដូច្នេះយើងអាចប្រើ MultiProvider ដើម្បីដាក់ Provider ទាំងនោះបាន៖

```
import 'counter_Logic.dart';
import 'my_screen.dart';
import 'theme_Logic.dart';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:provider/single_child_widget.dart';

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: _providers,
      child: MaterialApp(home: MyScreen()),
    );
  }

  final List<SingleChildWidget> _providers = [
    ChangeNotifierProvider(create: (context) => CounterLogic()),
    ChangeNotifierProvider(create: (context) => ThemeLogic()),
  ];
}
```

```dart
import 'package:flutter/foundation.dart';

class ThemeLogic extends ChangeNotifier{
  bool _isDark = false;

  bool get isDark => _isDark;

  void setDark(bool dark) {
    _isDark = dark;
    notifyListeners();
  }
}
```

ProxyProvider

- ករណីខ្លះយើងមាន Provider ២ ដែល Provider2 វាអាស្រ័យទៅ Provider1។ ដូច្នេះយើងអាចប្រើ ProxyProvider បាន ដែលយើងប្រើវាដើម្បីបញ្ជូលតំលៃពី Provider មួយទៅ Provider មួយទៀត។

```
final List<SingleChildWidget> _providers = [
  ChangeNotifierProvider(create: (context) => CounterLogic()),
  ProxyProvider<CounterLogic, CounterStringLogic>(
      update: (context, counterLogic, counterStringLogic) {
    return CounterStringLogic(counterLogic);
  }),
```

```dart
import 'counter_Logic.dart';

class CounterStringLogic {
  CounterLogic counterLogic;
  CounterStringLogic(this.counterLogic);

  String get counterString {
    int c = counterLogic.counter;
    return "$c item${ c != 0 ? 's' : ''}";
  }
}
```

```dart
//Example of Calling…
class MyStringText extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    String counter = context.read<CounterStringLogic>().counterString;
    return Expanded(
      child: Center(
        child: Text("$counter",
            style: TextStyle(fontSize: 20, color: Colors.pink)),
      ),
    );
  }
}
```