07

다양한 Provider

상태관리

프로바이더 기본 개념

프로바이더란?

- 프로바이더provider는 2019년 구글 IO에서 앱의 상태 관리 프레임워크로 소개
- 프로바이더에서 앱의 상태를 관리하는 데 제공하는 기법
  - •Provider: 하위 위젯이 이용할 상태 제공
  - •Consume: 하위 위젯에서 상태 데이터 이용

- 하위 위젯이 이용할 상태를 제공하는 여러 가지 Provider
  - •Provider: 기본 프로바이더
  - •ChangeNotifierProvider: 상태 변경 감지 제공
  - •MultiProvider: 여러 가지 상태 등록
  - •ProxyProvider: 다른 상태를 참조해서 새로운 상태 생산
  - •StreamProvider: 스트림 결과를 상태로 제공
  - •FutureProvide: 퓨처 결과를 상태로 제공

- 하위에서 프로바이더의 상태를 이용하는 방법도 여러 가지를 제공
  - •Provider.of(): 타입으로 프로바이더의 상태 데이터 획득
  - •Consumer: 상태를 이용할 위젯 명시
  - •Selector: 상태를 이용할 위젯과 그 위젯에서 이용할 상태 명시

프로바이더 기본 개념

프로바이더 이용하기

• 프로바이더 패키지 추가하기

```
dependencies:
  provider: ^6.0.2
```

- 상태 데이터 선언하기
  - 상태를 이용하는 위젯의 상위에 Provider를 등록하고 프로바이더에 상태 데이터를 추가해서 하위 위젯이 이용
  - Provider<int>.value()로 하위 위젯이 이용할 상태를 등록

• 상태 데이터 선언하기

```
Provider<int>.value(
    value: 10,
    child: SubWidget(),
)
```

프로바이더 기본 개념

프로바이더 이용하기

- Provider() 생성자를 이용하면 create 속성에 함수를 지정하여 이 함수에서 반환하는 값을 상태로 이용

• Provider() 생성자 이용하기

```
Provider<int>(
    create: (context) {
      int sum = 0;
      for (int i = 1; i <= 10; i++) {
        sum += i;
      }
      return sum;
    },
    child: SubWidget(),
)
```

프로바이더 기본 개념

프로바이더 이용하기

- 상태 데이터 이용
  - 프로바이더의 상태를 이용하는 하위 위젯은 Provider.of() 함수로 상태 데이터를 얻을 수 있습니다.

• 프로바이더의 상태 데이터 얻기

```
Widget build(BuildContext context) {
  final data = Provider.of<int>(context);
  ... (생략) ...
}
```

다양한 프로바이더 알아보기

변경된 상태를 하위 위젯에 적용하기 — ChangeNotifierProvider

- 변경된 상태 데이터를 하위 위젯에 적용하려면 ChangeNotifierProvider를 이용
- hangeNotifierProvider에 등록하는 상태 데이터는 ChangeNotifier를 구현

• 상태 데이터 모델 클래스

```
class Counter with ChangeNotifier {
  int _count = 0;
  int get count => _count;

  void increment() {
    _count++;
    notifyListeners();
  }
}
```

다양한 프로바이더 알아보기

변경된 상태를 하위 위젯에 적용하기 — ChangeNotifierProvider

- 상태 데이터가 변경된 후 변경 사항을 적용하려면 notifyListeners() 함수를 호출

• 다시 빌드할 위젯 등록

```
ChangeNotifierProvider<Counter>.value(
    value: Counter(),
    child: SubWidget(),
)
```

다양한 프로바이더 알아보기

멀티 프로바이더 등록하기 — MultiProvider

- 프로바이더를 한꺼번에 등록해서 이용할 때 하나의 프로바이더 위젯에 다른 프로바이더를 등록하여 계층 구조로 만들 수 있습니다.

• 여러 프로바이더 등록하기

```
Provider<int>.value(
    value: 10,
    child: Provider<String>.value(
        value: "hello",
        child: ChangeNotifierProvider<Counter>.value(
          value: Counter(),
          child: SubWidget(),
        )
    ),
)
```

다양한 프로바이더 알아보기

멀티 프로바이더 등록하기 — MultiProvider

- MultiProvider는 providers 속성을 제공하며 이 속성에 여러 프로바이더를 배열로 등록.

• MultiProvider를 이용해 여러 프로바이더 등록하기

```
MultiProvider(
  providers: [
    Provider<int>.value(value: 10),
    Provider<String>.value(value: "hello"),
    ChangeNotifierProvider<Counter>.value(value: Counter()),
  ],
  child: SubWidget()
)
```

다양한 프로바이더 알아보기

멀티 프로바이더 등록하기 — MultiProvider

- 프로바이더를 여러 개 등록할 때 타입 중복 문제가 발생할 수 있습니다.

• 타입이 중복된 프로바이더

```
MultiProvider(
  providers: [
    Provider<int>.value(value: 10),
    Provider<String>.value(value: "hello"),
    ChangeNotifierProvider<Counter>.value(value: Counter()),
    Provider<int>.value(value: 20),
    Provider<String>.value(value: "world"),
  ],
  child: SubWidget()
)
```

다양한 프로바이더 알아보기

멀티 프로바이더 등록하기 — MultiProvider

- 하위 위젯이 프로바이더를 이용할 때는 제네릭 타입으로 이용하므로 같은 제네릭 타입으로 등록하면 마지막에 등록한 프로바이더를 이용

• 하위 위젯에서 프로바이더 이용

```
Widget build(BuildContext context) {
  var counter = Provider.of<Counter>(context);
  var int_data = Provider.of<int>(context);
  var string_data = Provider.of<String>(context);
  return Row(
    children: <Widget>[
      Text('Provider : '),
      Text('int : $int_data, string : $string_data, count : ${counter.count}'),
      ElevatedButton(
        child: Text('increment'),
        onPressed:() {
          counter.increment();
        },
      )
    ],
  );
}
```

▶실행 결과

다양한 프로바이더 알아보기

상태 조합하기 — ProxyProvider

- 어떤 상탯값을 참조해서 다른 상탯값이 결정되게 할 수도 있습니다.
- 참조 상태를 다른 상태에 전달해야 하는데 이를 쉽게 구현하도록 ProxyProvider를 제공
- ProxyProvider<A, B>로 선언한다면 A는 전달받을 상태 타입이며, B는 A를 참조해서 만들 상태 타입

• 상태 조합하기

```
MultiProvider(
    providers: [
      ChangeNotifierProvider<Counter>.value(value: Counter()),
      ProxyProvider<Counter, Sum>(
          update: (context, model, sum) {
            return Sum(model);
          }
      )
    ],
    child: SubWidget()
)
```

다양한 프로바이더 알아보기

상태 조합하기 — ProxyProvider

- ProxyProvider는 전달받는 상태 개수에 따라 ProxyProvider2~ProxyProvider6까지 제공

• 여러 가지 상태 조합하기

```
ProxyProvider2<Counter, Sum, String>(
    update: (context, model1, model2, data) {
      return "${model1.count}, ${model2.sum}";
    }
)
```

다양한 프로바이더 알아보기

상태 조합하기 — ProxyProvider

- ProxyProvider의 생명주기
  - 프로바이더에 등록한 상태 객체는 싱글톤으로 운영
  - ProxyProvider에 등록한 상태 객체는 데이터가 변경될 때마다 객체가 다시 생성될 수 있습니다.

• 프로바이더 2개 등록

```
MultiProvider(
    providers: [
      ChangeNotifierProvider<Counter>.value(value: Counter()),
      ProxyProvider<Counter, Sum>(
          update: (context, model, sum) {
            return Sum(model);
          }
      ),
    ],
    child: SubWidget()
)
```

다양한 프로바이더 알아보기

상태 조합하기 — ProxyProvider

- update에 등록한 함수가 매번 호출되더라도 이전 상태 객체를 그대로 이용하면서 상탯값만 바꾸는 것이 효율적일 때가 있습니다.
- update에 등록한 함수의 세 번째 매개변수로 이전에 이용했던 상태 객체를 전달

• 상태 객체 생성 판단하기

```
ProxyProvider<Counter, Sum>(
    update: (context, model, sum) {
      if (sum != null) {     // 상탯값만 갱신
        sum.sum = model.count;
        return sum;
      } else {               // 새로운 객체 생성
        return Sum(model);
      }
    }
),
```

다양한 프로바이더 알아보기

퓨처 데이터 상태 등록하기 — FutureProvider

- FutureProvider는 Future로 발생하는 데이터를 상태로 등록하는 프로바이더

• 퓨처 데이터를 상태로 등록하기

```
FutureProvider<String>(
    create: (context) => Future.delayed(Duration(seconds: 4), () => "world"),
    initialData: "hello"
),
```

• FutureProvider의 상태를 이용하는 하위 위젯

```
var futureState = Provider.of<String>(context);
return Column(
  children: <Widget>[
    Text('future : ${futureState}'),
  ],
);
```

다양한 프로바이더 알아보기

스트림 데이터 상태 등록하기 — StreamProvider

- StreamProvider는 Stream으로 발생하는 데이터를 상태로 등록할 때 사용

• 1초마다 1~5까지 5번 숫자를 만드는 스트림 함수

```
Stream<int> streamFun() async* {
  for (int i = 1; i <=5; i++) {
    await Future.delayed(Duration(seconds: 1));
    yield i;
  }
}
```

• 스트림 데이터 상태로 등록하기

```
StreamProvider<int>(
    create: (context) => streamFun(),
    initialData: 0
)
```

• StreamProvider의 상태를 이용하는 하위 위젯

```
var streamState = Provider.of<int>(context);
return Column(
  children: <Widget>[
    Text('stream : ${streamState}'),
  ],
);
```

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare