03

StatelessWidget 과 StatefulWidget

UI Architecture

정적인 화면 만들기

- 위젯은 다음 3가지 클래스 가운데 하나를 상속받아 작성
  - •StatelessWidget: 상태를 관리하지 않는 정적인 위젯
  - •StatefulWidget: 상태를 관리하는 동적인 위젯
  - •InheritedWidget: 여러 위젯에서 공통으로 이용할 상태 관리 위젯

- 정적인 화면을 만들 때는 StatelessWidget을 상속받는 클래스를 선언

• 정적인 화면 만들기

```
class MyApp extends StatelessWidget {

  @override
  Widget build(BuildContext context) {

    return MaterialApp(
       ... (생략) ...
    );
  }
}
```

동적인 화면 만들기

- StatefulWidget은 상태를 유지하는 위젯
- 상태란 화면에서 갱신해야 하는 데이터를 의미
- StatefulWidget을 상속받은 클래스와 State를 상속받은 클래스가 필요
  - •StatefulWidget: 위젯 클래스
  - •State: StatefulWidget의 상탯값을 유지하고 화면을 구성하는 클래스

• 동적인 화면 만들기

```
class MyWidget extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return _MyWidgetState();
  }
}
```

• 상태 클래스 구현하기

```
class _MyWidgetState extends State<MyWidget> {

  @override
  Widget build(BuildContext context) {
    ... (생략) ...
  }
}
```

동적인 화면 만들기

상탯값 변경하기

- State에 선언한 변숫값을 단순히 변경하는 것만으로 화면을 다시 빌드하지는 않습니다.

• 상탯값 변경 예(화면에 반영되지 않음)

```
class _MyWidgetState extends State<MyWidget>{
  bool enabled = false;
  String stateText = "disable";

  void changeCheck() {
    if (enabled) {
      stateText = "disable";
      enabled = false;
    } else {
      stateText = "enable";
      enabled = true;
    }
  }
  ... (생략) ...
}
```

동적인 화면 만들기

상탯값 변경하기

- setState() 함수는 State 클래스에서 사용할 수 있으며 화면을 다시 빌드
- setState() 함수를 호출하면 화면을 구성하는 build() 함수가 다시 호출되고 그 결과로 반환된 위젯으로 화면을 갱신

• 상탯값 변경하기

```
class _MyWidgetState extends State<MyWidget>{
  bool enabled = false;
  String stateText = "disable";

  void changeCheck() {
    setState(() {
      if (enabled) {
        stateText = "disable";
        enabled = false;
      } else {
        stateText = "enable";
        enabled = true;
      }
    });
  }
  ... (생략) ...
}
```

동적인 화면 만들기

상태 클래스의 역할 알아보기

• 위젯 클래스 구현 예

```
class MyParentStatefulWidget extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return _MyParentStatefulWidgetState();
  }
}
class _MyParentStatefulWidgetState extends State<MyParentStatefulWidget> {
  ... (생략) ...
  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        MySubStatelessWidget(),
        MySubStatefulWidget()
      ],
    );
  }
}
```

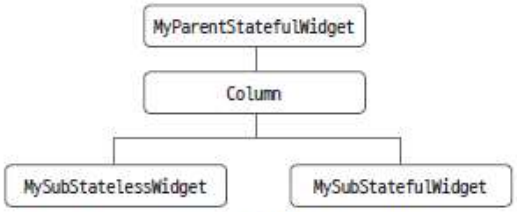

그림 8-7 위젯 트리

동적인 화면 만들기

상태 클래스의 역할 알아보기

- 플러터에서 위젯은 불변이므로 StatelessWidget이든 StatefulWidget이든 화면을 다시 빌드하면 이전 객체를 다시 이용하는 것이 아니라 새로운 객체가 생성
- StatefulWidget은 데이터를 유지하면서 다양한 업무를 처리하고 그 결과로 화면을 갱신
- StatefulWidget은 위젯 트리 구조에 포함해 매번 생성되게 만들고, 실제 데이터와 업무 로직은 State 객체를 따로 두어 화면이 다시 빌드될 때마다 매번 생성되지 않게 합니다.

동적인 화면 만들기

상태 클래스의 역할 알아보기

• 상태 클래스

```
class MySubStatefulWidget extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return _MySubStatefulWidgetState();
  }
}
```

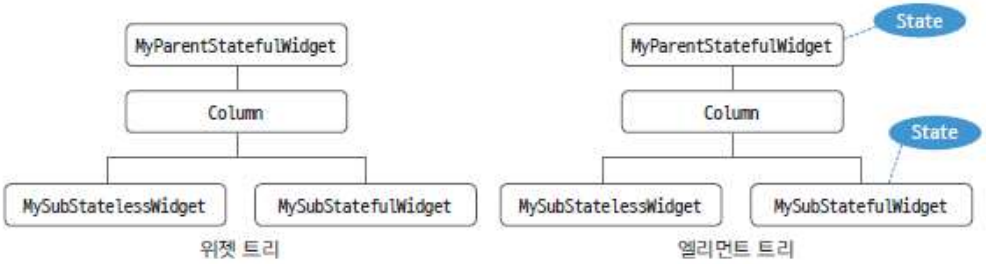

그림 8-8 엘리먼트 트리

상태의 생명주기

- Clean은 State에 의해 화면이 출력되고 있는 정상 상태이며, Dirty는 State 화면을 다시 빌드해야 하는 상태

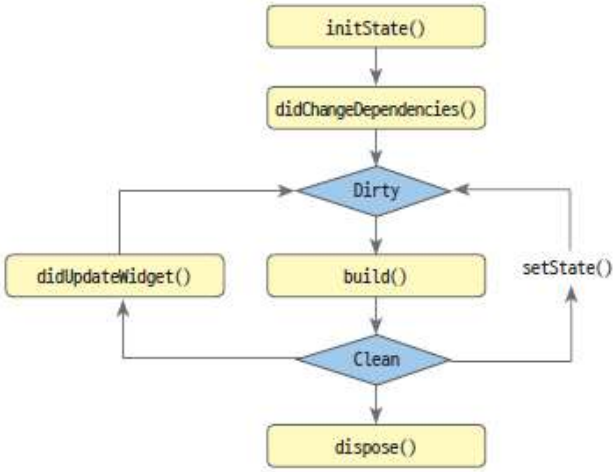

그림 8-10 State 객체의 생명 주기

상태의 생명주기

initState() 함수 호출 시점

- initState() 함수는 State 객체가 생성되자마자 가장 먼저 최초에 한 번 호출

didChangeDependencies() 함수 호출 시점

- didChangeDependencies() 함수는 initState() 함수가 호출된 후에 이어서 호출
- 상위 위젯의 상태 데이터가 변경될 때 하위 위젯의 didChangeDependencies()가 자동으로 호출되어 이 함수에서 상위 위젯의 변경된 상태 데이터를 이용

didUpdateWidget() 함수 호출 시점

- State에서는 자신과 연결된 StatefulWidget이 다시 생성되는 순간을 감지

상태의 생명주기

build() 함수 호출 시점

- •최초 호출
- •setState() 함수에 의해 호출
- •didUpdateWidget() 함수에 의해 호출

dispose() 함수 호출 시점

- dispose() 함수는 상태 객체를 소멸할 때 자동으로 호출

감사합니다

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

윌리엄 셰익스피어
William Shakespeare