Flutter 위젯 기초

상태관리와 위젯 리팩토링

학습목표

- Flutter의 Stateless 위젯과 Stateful 위젯의 개념과 차이점 이해
- 위젯 리팩토링을 통한 성능 최적화 방법 습득
- 상태관리를 활용한 사용자 인터페이스 구현 방법 학습

Flutter 위젯의 이해

- Flutter의 모든 UI 요소는 위젯으로 구성
- 위젯은 Stateless와 Stateful 두 가지 유형으로 분류
- 위젯 트리 구조를 통한 계층적 UI 구성 가능
- MaterialApp과 Scaffold를 통한 기본 앱 구조 제공

Stateless 위젯 구현

- 상태 변경이 필요 없는 정적 UI 구현에 사용
- 한번 생성되면 변경되지 않는 불변 위젯
- 성능상 이점이 있어 가능한 많이 활용 권장
- build 메서드를 통해 UI 구성

stateless_widget.dart

기본적인 Stateless 위젯 구현 예제입니다.

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

void main() {
  runApp(const MyApp());
}

// 앱의 메인 위젯
class MyApp extends StatelessWidget {
  // MyApp 위젯의 생성자. Key는 위젯의 식별에 사용될 수 있습니다.
  const MyApp({Key? key}) : super(key: key);

  // 위젯의 UI를 빌드하는 메서드. BuildContext는 위젯 트리의 현재 위치를 나타냅니다.
  @override
  Widget build(BuildContext context) {
    // 앱의 제목을 정의합니다.
    const title = 'Stateless Widget demo';

    // MaterialApp은 Flutter 앱의 기본 구조를 제공합니다.
    return MaterialApp(
      title: title, // 앱 제목
      home: Scaffold(
        appBar: AppBar(
          title: const Text(title), // 앱 바에 제목을 표시합니다.
        ),
        body: const MyTextWidget(), // 앱의 body에 MyTextWidget을 표시합니다.
      ),
    );
  }
}
```

stateless_widget_body.dart

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

// 텍스트를 표시하는 StatelessWidget
class MyTextWidget extends StatelessWidget {
  // MyTextWidget의 생성자
  const MyTextWidget({Key? key}) : super(key: key);

  // 위젯의 UI를 빌드하는 메서드
  @override
  Widget build(BuildContext context) {
    // Center 위젯은 자식 위젯을 가운데 정렬합니다.
    return const Center(
      child: Text('Hello'), // 'Hello' 텍스트를 표시합니다.
    );
  }
}
```

Stateful 위젯 구현

- 동적으로 변경되는 상태를 관리하는 위젯
- setState를 통한 상태 업데이트 가능
- 위젯의 생명주기 관리 용이
- 상태 변경에 따른 UI 자동 업데이트

stateful_widget.dart

상태 관리가 필요한 Stateful 위젯 예제입니다.

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

// 상태를 가질 수 있는 StatefulWidget
class MyTextWidget extends StatefulWidget {
  // MyTextWidget의 생성자
  const MyTextWidget({Key? key}) : super(key: key);

  // createState() 메서드는 이 위젯에 대한 State 객체를 생성합니다.
  @override
  _MyTextWidgetState createState() => _MyTextWidgetState();
}

// MyTextWidget의 상태를 관리하는 State 클래스
class _MyTextWidgetState extends State<MyTextWidget> {
  // count 변수는 클릭 횟수를 저장합니다.
  int count = 0;

  // 위젯의 UI를 빌드하는 메서드
  @override
  Widget build(BuildContext context) {
    // GestureDetector는 탭과 같은 제스처를 감지합니다.
    return GestureDetector(
      // onTap은 위젯이 탭될 때 호출되는 콜백 함수입니다.
      onTap: () {
        // setState()를 호출하여 UI를 업데이트합니다.
        setState(() {
          count++; // 클릭 횟수를 증가시킵니다.
        });
      },
      // Center 위젯은 자식 위젯을 가운데 정렬합니다.
      child: Center(
        // Text 위젯은 텍스트를 표시합니다.
        child: Text('Click Me: $count')
      ),
    );
  }
}
```

위젯 리팩토링

- 코드의 가독성과 재사용성 향상
- 빌드 컨텍스트 최적화를 통한 성능 개선
- 데이터 로직과 UI 로직의 분리
- 유지보수가 용이한 구조 설계

refactored_widget.dart

리팩토링된 위젯 구조 예제입니다.

```dart
// 데이터 모델 클래스
class DataItem {
  final String title;    // 제목
  final String subtitle; // 부제목
  final String url;      // 이미지 URL

  // DataItem 생성자
  const DataItem({
    required this.title,
    required this.subtitle,
    required this.url,
  });
}

// DataItem을 사용하는 View 클래스
class DataView {
  // DataItem 인스턴스 생성 및 초기화
  final DataItem item = const DataItem(
    title: 'Hello',
    subtitle: 'subtitle',
    url: 'https://oreil.ly/O4PEn'
  );
}
```

container_widget.dart

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

// Container 위젯을 사용하는 StatelessWidget
class MyContainerWidget extends StatelessWidget {
  // MyContainerWidget 생성자
  MyContainerWidget({Key? key}) : super(key: key);

  // DataView 인스턴스 생성
  final DataView data = DataView();

  // 위젯의 UI를 빌드하는 메서드
  @override
  Widget build(BuildContext context) {
    // Container 위젯은 크기와 색상을 지정할 수 있는 사각형 영역을 만듭니다.
    return Container(
      width: 200,   // 너비 200
      height: 180,  // 높이 180
      color: Colors.black, // 배경색 검정
      // Column 위젯은 자식 위젯을 세로로 배열합니다.
      child: Column(
        children: [
          // Image.network 위젯은 네트워크에서 이미지를 로드하여 표시합니다.
          Image.network(data.item.url),
          // Text 위젯은 텍스트를 표시합니다.
          Text(
            data.item.title, // 제목 표시
            style: const TextStyle(fontSize: 20, color: Colors.white), // 스타일 지정
          ),
          // Text 위젯은 텍스트를 표시합니다.
          Text(
            data.item.subtitle, // 부제목 표시
            style: const TextStyle(fontSize: 16, color: Colors.grey), // 스타일 지정
          ),
        ],
      ),
    );
  }
}
```

요약

- Stateless 위젯은 상태 변경이 필요 없는 정적 UI에 사용
- Stateful 위젯은 동적으로 변경되는 상태를 관리
- 위젯 리팩토링을 통해 가독성과 성능을 향상
- 데이터 모델과 UI 로직을 분리하여 구조화

END