07

다양한 Bloc 기법

상태관리

다양한 Bloc 이용 기법

Bloc 옵저버

- Bloc 옵저버는 BlocObserver를 상속받아 작성하는 개발자 클래스
- Bloc 클래스처럼 onEvent(), onTransition(), onError() 함수를 가지며 추가로 onChange() 함수도 가집니다.

• Bloc 옵저버 작성하기

```
class MyBlocObserver extends BlocObserver {
  @override
  void onEvent(Bloc bloc, Object? event) {
    super.onEvent(bloc, event);
    print('observer onEvent...${bloc.state}.');
  }

  @override
  void onTransition(Bloc bloc, Transition transition) {
    super.onTransition(bloc, transition);
    print('observer onTransition...${transition}.');
  }

  @override
  void onError(BlocBase bloc, Object error, StackTrace stackTrace) {
    super.onError(bloc, error, stackTrace);
    print('observer onError....');
  }

  @override
  void onChange(BlocBase bloc, Change change) {
    super.onChange(bloc, change);
    print('observer onChange....${change.currentState}, ${change.nextState}');
  }
}
```

다양한 Bloc 이용 기법

Bloc 옵저버

- Bloc 옵저버 클래스를 선언했으면 Bloc에 등록
- 어디선가 한 번만 해주면 되며 보통은 앱의 진입점인 main() 함수에서 다음 코드처럼 등록

• Bloc에 옵저버 등록하기

```
void main() {
  BlocOverrides.runZoned(() {
      runApp(MyApp());
    },
    blocObserver: MyBlocObserver()
  );
}
```

다양한 Bloc 이용 기법

Bloc 옵저버

- Bloc 옵저버의 함수가 호출되는 시점은 Bloc 클래스의 함수가 호출되는 시점과 동일
- Bloc 옵저버를 이용하면 옵저버의 함수가 먼저 호출되고 이후에 Bloc의 함수가 호출

• 상태 데이터 변경하기

```
on<IncrementEvent>((event, emit) {
  print('bloc state change before....');
  emit(state + event.no);
  print('bloc state change after....');
});
```

▶ 실행 결과

```
observer onEvent...0.
bloc onEvent......Instance of 'IncrementEvent'
bloc state change before....
observer onTransition...Transition { currentState: 0, event: Instance of
'IncrementEvent', nextState: 2 }.
bloc onTransition.... Transition { currentState: 0, event: Instance of
'IncrementEvent', nextState: 2 }
observer onChange....0, 2
bloc state change after....
```

다양한 Bloc 이용 기법

멀티 Bloc 프로바이더

- Bloc 클래스는 여러 개이며 이를 등록하려면 다음처럼 하나의 Bloc 프로바이더 하위에 다시 Bloc 프로바이더를 추가하는 식으로 등록

• Bloc 프로바이더 중첩 등록하기

```
BlocProvider<BlocCounter>(
    create: (context) => BlocCounter(),
    child: BlocProvider<UserBloc>(
      create: (context) => UserBloc(),
      child: MyWidget(),
    )
)
```

다양한 Bloc 이용 기법

멀티 Bloc 프로바이더

- MultiBlocProvideder를 이용하면 좀 더 편리

• 멀티 Bloc 프로바이더 사용하기

```
MultiBlocProvider(
  providers: [
    BlocProvider<BlocCounter>(create: (context) => BlocCounter()),
    BlocProvider<UserBloc>(create: (context) => UserBloc())
  ],
  child: MyWidget(),
)
```

다양한 Bloc 이용 기법

Bloc 빌더

- 위젯에서 Bloc 객체를 얻는 기본 방법은 BlocProvider.of()를 이용

• Bloc 객체 얻기

```
final BlocCounter counterBloc = BlocProvider.of<BlocCounter>(context);
```

- BlocBuilder를 이용해 Bloc의 상태 데이터를 좀 더 쉽게 얻을 수 있습니다.

• BlocBuilder로 Bloc 객체 얻기

```
BlocBuilder<BlocCounter, int>(
  builder: (context, count) {
    return Row(
      children: <Widget>[
        Text(
          'Bloc : $count',
        ),
      ],
    );
  },
)
```

다양한 Bloc 이용 기법

Bloc 빌더

- 상위에 등록하지 않은 Bloc를 BlocBuilder에서 정의하는 위젯에서만 사용하는 경우.
- 이때는 BlocBuilder 안에 bloc 속성으로 이용하려는 Bloc 객체를 지정

• BlocBuilder의 bloc 속성 사용하기

```
final userBloc = UserBloc();
... (생략) ...
BlocBuilder<UserBloc, User?>(
  bloc: userBloc,
  builder: (context, user) {
    return Row(
      children: [
        Text('user ${user?.name}, ${user?.address}'),
        ElevatedButton(
            onPressed: () {
              userBloc.add(CreateUserEvent(User('kkang', 'seoul')));
            },
            child: Text('create')),
        ElevatedButton(
            onPressed: () {
              userBloc.add(UpdateUserEvent(User('kim', 'busan')));
            },
            child: Text('update'))
      ],
    );
  },
)
```

다양한 Bloc 이용 기법

Bloc 빌더

- Bloc의 상탯값이 변경되더라도 builder에 명시한 위젯을 다시 빌드하지 않을 때는 buildWhen 속성을 이용
- Bloc의 상탯값이 변경될 때 buildWhen에 선언한 함수가 자동으로 호출
- true를 반환하면 builder의 함수가 자동으로 호출되어 위젯을 다시 빌드
- false를 반환하면 builder의 함수가 호출되지 않습니다.

• buildWhen 속성 사용하기

```
BlocBuilder<BlocCounter, int>(
  buildWhen: (previous, current) {
    return true;
  },
  builder: (context, count) {
    ... (생략) ...
  },
),
```

다양한 Bloc 이용 기법

Bloc 리스너

- BlocListener는 BlocBuilder와 마찬가지로 상탯값을 얻으려고 사용하지만 builder 속성이 없습니다.
- BlocListener를 선언하면서 이용하려는 Bloc와 상태의 타입을 제네릭으로 선언
- 해당 Bloc의 상탯값이 변경될 때마다 listener 속성에 지정한 함수가 자동으로 호출

• Bloc 리스너 이용하기

```
BlocListener<BlocCounter, int>(
    listener: (context, state) {
      ... (생략) ...
    },
)
```

다양한 Bloc 이용 기법

Bloc 리스너

- listenWhen으로 상탯값이 변경될 때 listener에 지정한 함수를 호출해야 하는지를 제어

• listenWhen 속성으로 리스너 제어하기

```
BlocListener<BlocCounter, int>(
    listenWhen: (previous, current) {
      return true;
    },
    listener: (context, state) {
      ... (생략) ...
    },
)
```

다양한 Bloc 이용 기법

Bloc 리스너

- BlocListener를 한꺼번에 여러 개 등록할 때는 MultiBlocListener를 이용

• Bloc 리스너 여러 개 등록하기

```
MultiBlocListener(
  listeners: [
    BlocListener<BlocCounter, int>(
      listenWhen: (previous, current) {
        return true;
      },
      listener: (context, state) {
        ... (생략) ...
      },
    ),
    BlocListener<UserBloc, User?>(
        listener: (context, user) {
          ... (생략) ...
        }
    )
  ],
)
```

다양한 Bloc 이용 기법

Bloc 컨슈머

- Bloc Builder와 BlocListener를 함께 사용

• BlocBuilder와 BlocListener를 함께 사용하기

```
BlocListener<BlocCounter, int>(
  listener: (context, state) {
    ... (생략) ...
  },
  child: BlocBuilder<BlocCounter, int>(
    builder: (context, count) {
      ... (생략) ...
    },
  ),
)
```

- BlocConsumer를 이용하면 BlocListener와 BlocBuilder를 함께 사용하는 부분을 쉽게 작성

• Bloc 컨슈머 사용하기

```
BlocConsumer<BlocCounter, int>(
    listener: (context, state) {
      ... (생략) ...
    },
    builder: (context, count) {
      ... (생략) ...
    }
)
```

다양한 Bloc 이용 기법

Bloc 컨슈머

- BlocConsumer에 listenWhen과 buildWhen을 추가해 상탯값이 변경될 때 listener, builder 속성에 지정한 함수를 호출해야 하는지를 제어

• 리스너와 빌더 함수 호출 제어하기

```
BlocConsumer<BlocCounter, int>(
    listenWhen: (previous, current) {
      return true;
    },
    listener: (context, state) {
      ... (생략) ...
    },
    buildWhen: (previous, current) {
      return true;
    },
    builder: (context, count) {
      ... (생략) ...
    }
)
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare