06

Future

네트워킹과 비동기

퓨처와 퓨처 빌더

퓨처 — Future

- 시간이 오래 걸리는 작업이라고 가정

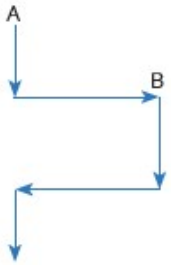

그림 16-1 동기 실행

• 동기 프로그래밍

```
void sum() {
  var sum = 0;
  Stopwatch stopwatch = Stopwatch();
  stopwatch.start();
  for (int i = 0; i < 500000000; i++) {
    sum += i;
  }
  stopwatch.stop();
  print("${stopwatch.elapsed}, result: $sum");
}

void onPress() {
  print('onPress top...');
  sum();
  print('onPress bottom...');
}
```

▶ 실행 결과

```
onPress top...
0:00:01.700258, result: 124999999750000000
onPress bottom...
```

퓨처와 퓨처 빌더

퓨처 — Future

- 시간이 오래 걸리는 작업이 처리되는 동안 다른 작업도 함께 처리하는 것을 비동기 프로그래밍이라고 하며, 이때 Future 클래스를 사용합니다
- Future는 다트 언어에서 제공하는 클래스이며 미래에 발생할 데이터를 의미

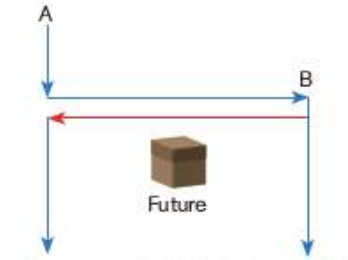

그림 16-2 Future를 이용한 비동기 프로그래밍

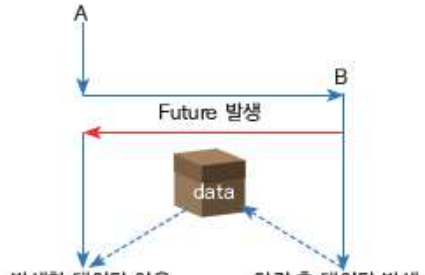

그림 16-3 Future와 데이터

• 비동기 처리

```
Future<int> sum() {
  return Future<int>(() {   // 미래의 데이터를 담을 상자 반환
    var sum = 0;
    Stopwatch stopwatch = Stopwatch();
    stopwatch.start();
    for (int i = 0; i < 500000000; i++) {
      sum += i;
    }
    stopwatch.stop();
    print("${stopwatch.elapsed}, result: $sum");
    return sum;   // 실제 데이터를 상자에 담기
  });
}

void onPress() {
  print('onPress top...');
  sum();
  print('onPress bottom...');
}
```

▶ 실행 결과

```
onPress top...
onPress bottom...
0:00:01.612296, result: 124999999750000000
```

퓨처와 퓨처 빌더

퓨처 빌더 — FutureBuilder

- 결과가 나올 때까지 대기했다가 화면에 출력해 주는 위젯이 필요한데, FutureBuilder가 그 역할

• FutureBuilder의 생성자

```
const FutureBuilder<T>(
  { Key? key,
  Future<T>? future,
  T? initialData,
  required AsyncWidgetBuilder<T> builder }
)
```

- FutureBuilder가 출력하는 화면은 생성자 매개변수로 지정되는 AsyncWidgetBuilder에 명시

• AsyncWidgetBuilder 정의

```
AsyncWidgetBuilder<T> = Widget Function(
  BuildContext context,
  AsyncSnapshot<T> snapshot
)
```

퓨처와 퓨처 빌더

퓨처 빌더 — FutureBuilder

- AsyncWidgetBuilder의 두 번째 매개변수 타입이 AsyncSnapshot이며 이곳에 Future 데이터를 전달

• 퓨처 데이터 출력하기

```
body: Center(
  child: FutureBuilder(
    future: calFun(),
    builder: (context, snapshot) {
      if (snapshot.hasData) {
        return Text('${snapshot.data}');
      }
      return CircularProgressIndicator();
    },
  ),
),
```

await와 async

• Future에 담은 데이터 가져오기

```
void onPress() {
  print('onPress top...');
  Future<int> future = sum();
  print('onPress future: $future');
  print('onPress bottom...');
}
```

▶ 실행 결과

```
onPress top...
onPress future: Instance of 'Future<int>'
onPress bottom...
0:00:01.641200, result: 124999999750000000
```

타입 정보만 출력

await와 async

then() 함수 사용하기

- Future 객체의 then()을 이용하여 매개변수에 콜백 함수를 등록
- Future에 데이터가 담기는 순간 콜백 함수가 호출

• then() 함수로 Future에 담은 데이터 가져오기

```
void onPress() {
  print('onPress top...');
  Future<int> future = sum();
  future.then((value) => print('onPress then... $value'));
  future.catchError((error) => print('onPress catchError... $error'));
  print('onPress bottom...');
}
```

▶ 실행 결과

```
flutter: onPress top...
flutter: onPress bottom...
flutter: 0:00:01.625221, result: 124999999750000000
flutter: onPress then...124999999750000000
```

데이터 출력

await와 async

• 시간이 오래 걸리는 함수 2개

```
Future<int> funA() {
  return Future.delayed(Duration(seconds: 3), () {
    return 10;
  });
}
Future<int> funB(int arg) {
  return Future.delayed(Duration(seconds: 2),() {
    return arg * arg;
  });
}
```

• then() 함수 중첩

```
Future<int> calFun() {
  return funA().then((aResult) {
    return funB(aResult);
  }).then((bResult) {
    return bResult;
  });
}
```

await와 async

await와 async 사용하기

- await는 실행 영역에 작성하며 async는 선언 영역에 작성
- await는 한 작업의 처리 결과를 받아서 다음 작업을 처리해야 할 때 먼젓번 작업의 처리가 끝날 때까지 대기시키는 용도

• await, async로 처리

```
Future<int> calFun() async {
  int aResult = await funA();
  int bResult = await funB(aResult);
  return bResult;
}
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare