# Chapter9 - 일시 중단 함수

## 9.1. 일시 중단 함수와 코루틴

### 9.1.1. 일시 중단 함수란 무엇인가?

일시 중단 함수는 suspend fun 키워드로 선언되는 함수로 함수 내에 일시 중단 지점을 포함할 수 있는 특별한 기능을 한다. 일시 중단 함수는 주로 코루틴의 비동기 작업과 관련된 복잡한 코드들을 구조화하고 재사용할 수 있는 코드의 집합으로 만드는 데 사용된다.

```
// Code9-1.kt
fun main() = runBlocking<Unit> {
  delay(1000L)
  println("Hello World")
  delay(1000L)
  println("Hello World")
}
```

```
// Code9-2.kt
fun main() = runBlocking<Unit> {
  delayAndPrintHelloWorld()
  delayAndPrintHelloWorld()
}

suspend fun delayAndPrintHelloWorld() {
  delay(1000L)
  println("Hello World")
}
```

일시 중단 함수는 일반 함수와 용도가 같은데 다른 점은 일시 중단 지점을 포함하고 있다는 것이다.

### 9.1.2. 일시 중단 함수는 코루틴이 아니다

일시 중단 함수는 코루틴 내부에서 실행되는 코드의 집합일 뿐, 코루틴이 아니다.

```
// Code9-3.kt
fun main() = runBlocking<Unit> {
  val startTime = System.currentTimeMillis()
  delayAndPrintHelloWorld()
```

```
  delayAndPrintHelloWorld()
  println(getElapsedTime(startTime))
}

suspend fun delayAndPrintHelloWorld() {
  delay(1000L)
  println("Hello World")
}

/*
// 결과:
Hello World
Hello World
지난 시간: 2017ms
*/

fun getElapsedTime(startTime: Long): String = "지난 시간:
${System.currentTimeMillis() - startTime}ms"
```

이 코드에서 생성되는 코루틴은 runBlocking 코루틴 단 한 개 뿐이며, 코루틴 내부에서 2개의 delayAndPrintHelloWorld 함수가 순차적으로 실행된다.

## 9.1.3. 일시 중단 함수를 별도의 코루틴상에서 실행하기

만약 일시 중단 함수를 코루틴처럼 비동기적으로 사용하고 싶다면 일시 중단 함수를 코루틴 빌더로 감싸야 한다.

```
// Code9-7.kt
fun main() = runBlocking<Unit> {
  val startTime = System.currentTimeMillis()
  launch {
    delayAndPrintHelloWorld()
  }
  launch {
    delayAndPrintHelloWorld()
  }
  println(getElapsedTime(startTime))
}

suspend fun delayAndPrintHelloWorld() {
  delay(1000L)
  println("Hello World")
}
```

```
/*
// 결과:
지난 시간: 3ms
Hello World
Hello World
*/

fun getElapsedTime(startTime: Long): String = "지난 시간:
${System.currentTimeMillis() - startTime}ms"
```

이 코드에서 launch 함수가 호출돼 생성된 코루틴들은 delayAndPrintHelloWorld 함수의 호출로 1초간 스레드 사용 권한을 양보한다. 자유로워진 스레드는 다른 코루틴인 runBlocking 코루틴에 의해 사용될 수 있으므로 곧바로 마지막 줄의 `println(getElapsedTime(startTime))` 가 실행된다.

# 9.2. 일시 중단 함수의 사용

## 9.2.1. 일시 중단 함수의 호출 가능 지점

코틀린에서 일시 중단이 가능한 지점은 다음 두 가지이다.

1. 코루틴 내부
2. 일시 중단 함수

### 일시 중단 함수에서 다른 일시 중단 함수 호출하기

일시 중단 함수는 또다른 일시 중단 함수에서 호출될 수 있으며, 데이터베이스와 서버에서 키워드로 검색을 실행해 결과를 가져오는 searchByKeyword 일시 중단 함수를 다음과 같이 만들 수 있다.

```
// Code9-10.kt
suspend fun searchByKeyword(keyword: String): Array<String> {
  val dbResults = searchFromDB(keyword)
  val serverResults = searchFromServer(keyword)
  return arrayOf(*dbResults, *serverResults)
}

suspend fun searchFromDB(keyword: String): Array<String> {
  delay(1000L)
  return arrayOf("[DB]${keyword}1", "[DB]${keyword}2")
}

suspend fun searchFromServer(keyword: String): Array<String> {
```

```
  delay(1000L)
  return arrayOf("[Server]${keyword}1", "[Server]${keyword}2")
}
```

하지만 launch나 async 코루틴 빌더 함수는 CoroutineScope의 확장 함수로 선언돼 있기 때문에 이렇게 코드를 작성하면 오류(Unresolved reference: async)가 발생한다. 그 이유는 일시 중단 함수 내부에서는 일시 중단 함수를 호출한 코루틴의 CoroutineScope 객체에 접근할 수 없기 때문이다. (간단히 말해서 일시 중단 함수 내부에서는 CoroutineScope 객체에 접근 불가능)

### 9.2.2.2. coroutineScope 사용해 일시 중단 함수에서 코루틴 실행하기

coroutineScope 일시 중단 함수를 사용하면 일시 중단 함수 내부에 새로운 CoroutineScope 객체를 생성할 수 있다. coroutineScope 함수는 구조화를 깨지 않는 CoroutineScope 객체를 생성한다.

```
// Code9-12.kt
fun main() = runBlocking<Unit> {
  val startTime = System.currentTimeMillis() // 1. 시작 시간 기록
  val results = searchByKeyword("Keyword") // 2. 검색 실행 및 결과 값 반환 받기
  println("[결과] ${results.toList()}") // 3. 결과값 출력
  println(getElapsedTime(startTime)) // 4. 지난 시간 표시
}

suspend fun searchByKeyword(keyword: String): Array<String> = coroutineScope
{ // this: CoroutineScope
  val dbResultsDeferred = async {
    searchFromDB(keyword)
  }
  val serverResultsDeferred = async {
    searchFromServer(keyword)
  }

  return@coroutineScope arrayOf(*dbResultsDeferred.await(),
*serverResultsDeferred.await())
}

suspend fun searchFromDB(keyword: String): Array<String> {
  delay(1000L)
  return arrayOf("[DB]${keyword}1", "[DB]${keyword}2")
}

suspend fun searchFromServer(keyword: String): Array<String> {
  delay(1000L)
  return arrayOf("[Server]${keyword}1", "[Server]${keyword}2")
}
```

```
fun getElapsedTime(startTime: Long): String = "지난 시간:
${System.currentTimeMillis() - startTime}ms"

/*
// 결과:
[결과] [[DB]Keyword1, [DB]Keyword2, [Server]Keyword1, [Server]Keyword2]
지난 시간: 1039ms
*/
```

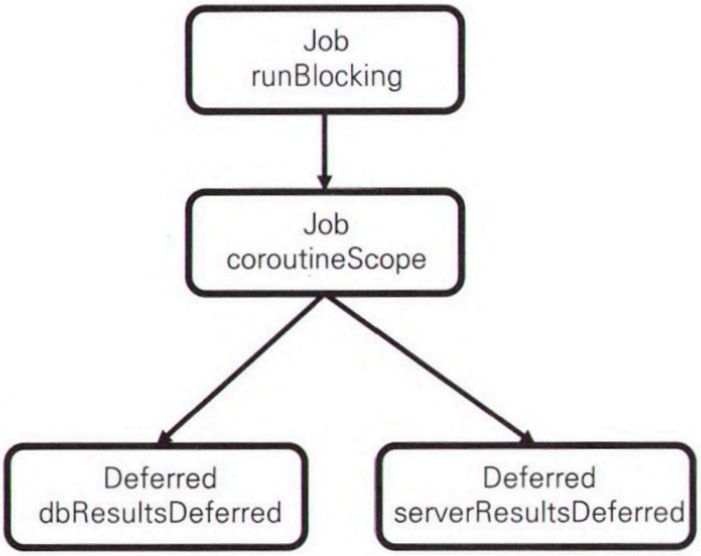


**그림 9-2** coroutineScope 호출 시 코루틴의 구조화

여기서 문제는 자식 코루틴이 오류를 발생시키면 부모 코루틴으로 오류를 전파하여 다른 자식 코루틴도 취소시킨다는 점이다.

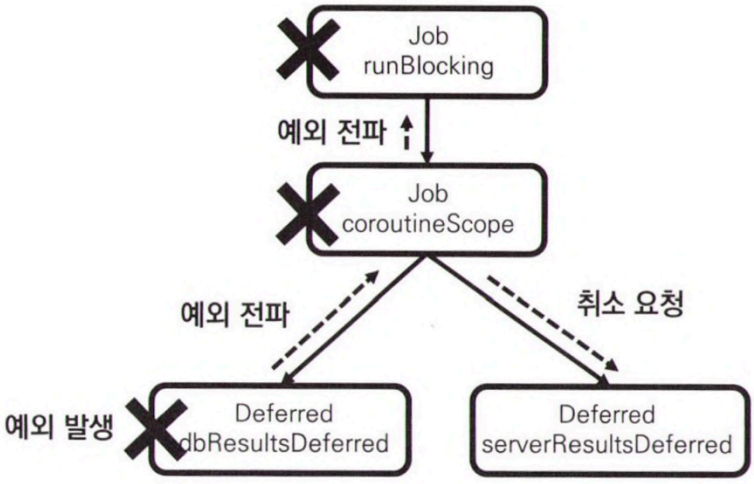


**그림 9-3** 예외 발생 시 생기는 문제

```kotlin
// Code9-13.kt
fun main() = runBlocking<Unit> {
  println("[결과] ${searchByKeyword("Keyword").toList()}")
}
/*
// 결과:
[결과] [[Server]Keyword1, [Server]Keyword2]
*/

suspend fun searchByKeyword(keyword: String): Array<String> =
supervisorScope { // this: CoroutineScope
  val dbResultsDeferred = async {
    throw Exception("dbResultsDeferred에서 예외가 발생했습니다")
    searchFromDB(keyword)
  }
  val serverResultsDeferred = async {
    searchFromServer(keyword)
  }

  val dbResults = try {
    dbResultsDeferred.await()
  } catch (e: Exception) {
    arrayOf() // 예외 발생 시 빈 결과 반환
  }

  val serverResults = try {
    serverResultsDeferred.await()
  } catch (e: Exception) {
    arrayOf() // 에러 발생 시 빈 결과 반환
  }

  return@supervisorScope arrayOf(*dbResults, *serverResults)
}

suspend fun searchFromDB(keyword: String): Array<String> {
  delay(1000L)
  return arrayOf("[DB]${keyword}1", "[DB]${keyword}2")
}

suspend fun searchFromServer(keyword: String): Array<String> {
  delay(1000L)
  return arrayOf("[Server]${keyword}1", "[Server]${keyword}2")
}
```

```
fun getElapsedTime(startTime: Long): String = "지난 시간:
${System.currentTimeMillis() - startTime}ms"
```

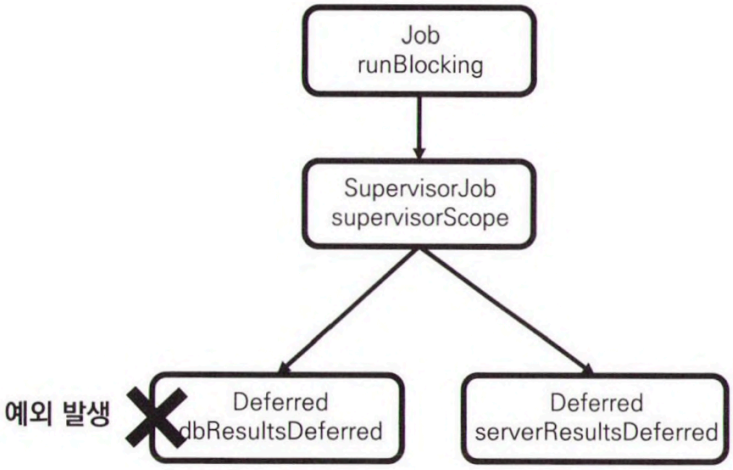


**그림 9-4** 일시 중단 함수에서 supervisorScope 사용

이처럼 supervisorScope 함수를 일시 중단 함수 내부에서 사용하면 구조화를 깨지 않는 새로운 CoroutineScope 객체도 만들 수 있고, 이 CoroutineScope 객체 하위에서 실행되는 코루틴들의 예외 전파도 방지할 수 있다.