iCo 기술세션

Trouble Shooting

강대훈

목차

1. 코인 추천 로직
2. Throttle / Debounce
3. SSE Client
4. 챗봇 고도화 작업

코인 추천 로직?

Alan API 호출 -> 응답 파싱 -> Upbit API 호출

Callback

```
struct RecommendCoinDTO: Codable {
    let name: String
    let symbol: String
    let comment: String
}
```

codetoimg.com

X 10

Callback

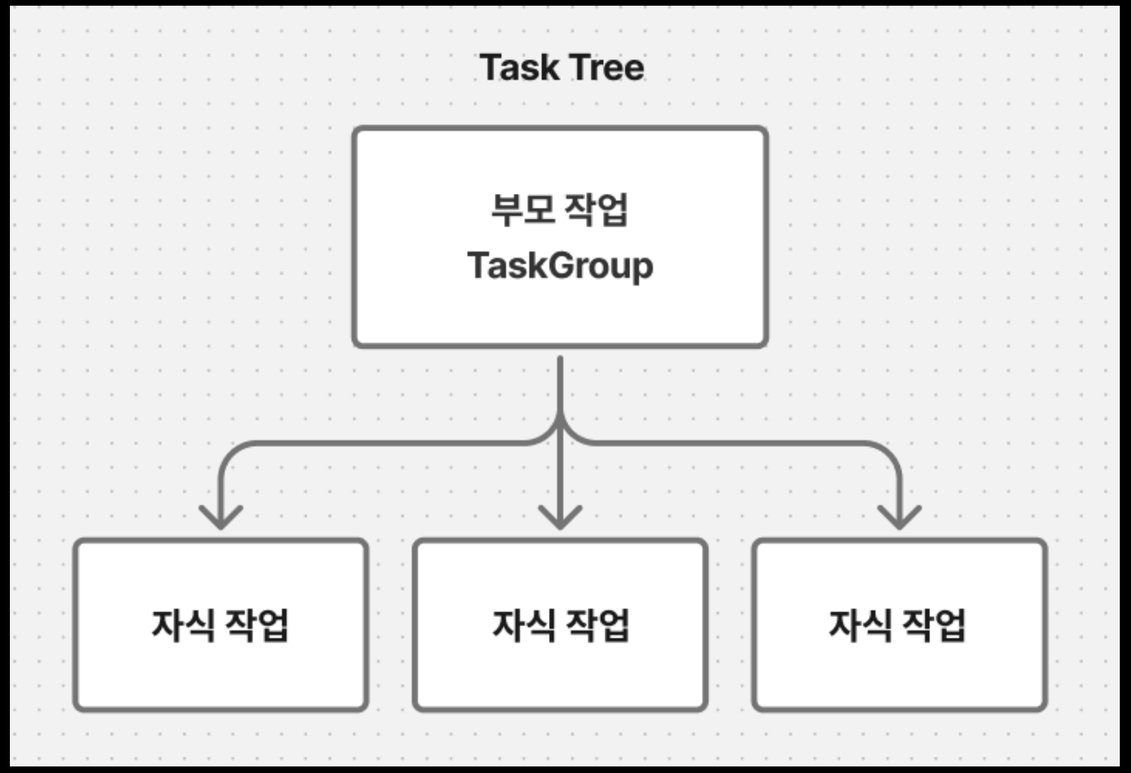
```
private func fetchRecommendCoins(from dtos: [RecommendCoinDTO]) async -> [RecommendCoin] {
    await withTaskGroup(of: RecommendCoin?.self) { group in
        ...
    }
}
```

codetoimg.com

작업의 협력적 취소

TaskGroup은 구조적 동시성 작업 (작업 취소의 전파)

만약 자식 작업중에서 하나라도 에러가 발생한다면?



비구조적 동시성 작업

Callback

```
let task = Task {
    Task.detached {
        // 작업...
    }

    Task.detached {
        // 작업...
    }

    Task.detached {
        // 작업...
    }
}

task.cancel()
```

작업의 협력적 취소 - withThrowingTaskGroup

에러 발생시에 작업 취소 전파 O

Callback

```swift
private func fetchRecommendCoins(from dtos: [RecommendCoinDTO]) async throws -> [RecommendCoin] {
    try await withThrowingTaskGroup(of: RecommendCoin.self) { group in
        for dto in dtos {
            group.addTask {
                guard let data = try await self.upbitService.fetchQuotes(id: "KRW-\(dto.symbol)").first else {
                    throw NetworkError.invalidResponse
                }

                return RecommendCoin(
                    imageURL: nil,
                    comment: dto.comment,
                    coinID: data.coinID.replacingOccurrences(of: "KRW-", with: ""),
                    name: dto.name,
                    tradePrice: data.tradePrice,
                    changeRate: data.changeRate * 100,
                    changeType: RecommendCoin.TickerChangeType(rawValue: data.change)
                )
            }
        }

        var results: [RecommendCoin] = []

        for try await coin in group {
            results.append(coin)

            if results.count == numberOfCoins {
                group.cancelAll()
                break
            }
        }
        return results
    }
}
```

작업의 협력적 취소 - withTaskGroup

에러 발생시에 작업 취소 전파 X

Callback

```
private func fetchRecommendCoins(from dtos: [RecommendCoinDTO]) async -> [RecommendCoin] {
    await withTaskGroup(of: RecommendCoin?.self) { group in
        for dto in dtos {
            group.addTask {
                do {
                    guard let data = try await self.upbitService.fetchQuotes(id: "KRW-\(dto.symbol)").first else {
                        return nil
                    }

                    return RecommendCoin(
                        imageURL: nil,
                        comment: dto.comment,
                        coinID: data.coinID.replacingOccurrences(of: "KRW-", with: ""),
                        name: dto.name,
                        tradePrice: data.tradePrice,
                        changeRate: data.changeRate * 100,
                        changeType: RecommendCoin.TickerChangeType(rawValue: data.change)
                    )
                } catch {
                    return nil
                }
            }
        }

        var results: [RecommendCoin] = []

        for await coin in group {
            if let coin = coin {
                results.append(coin)

                if results.count == numberOfCoins {
                    group.cancelAll()
                    break
                }
            }
        }
        return results
    }
}
```

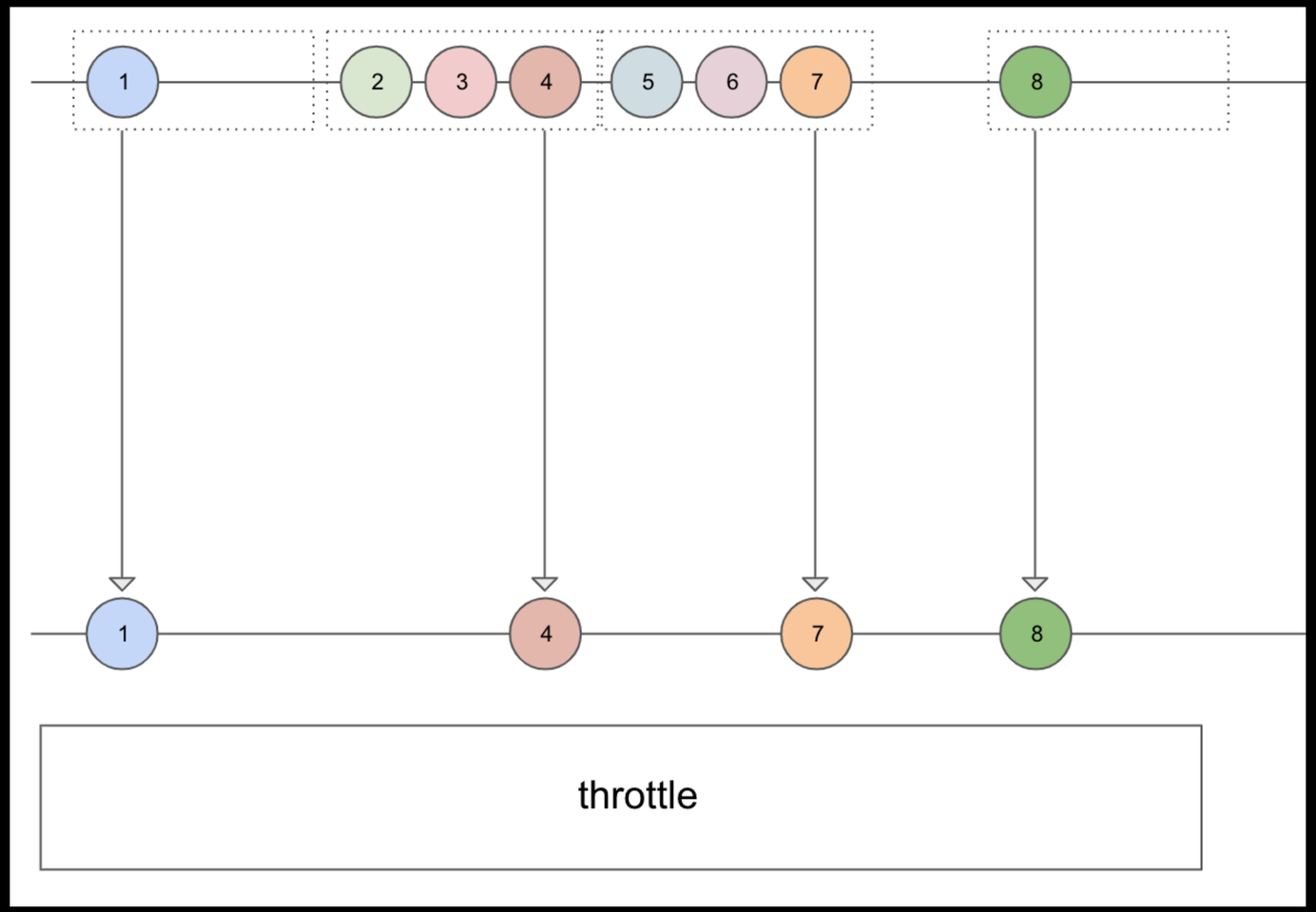
Throttle & Debounce

왜 사용하는가?

-> 이벤트가 과도하게 발생하는 상황을 제어할 수 있다!

Throttle

지정한 시간 간격으로 마지막 이벤트만 호출



https://docfriends.github.io/DevStrory/2019-01-29/swift-delay/

Throttle

적용 예제

Callback

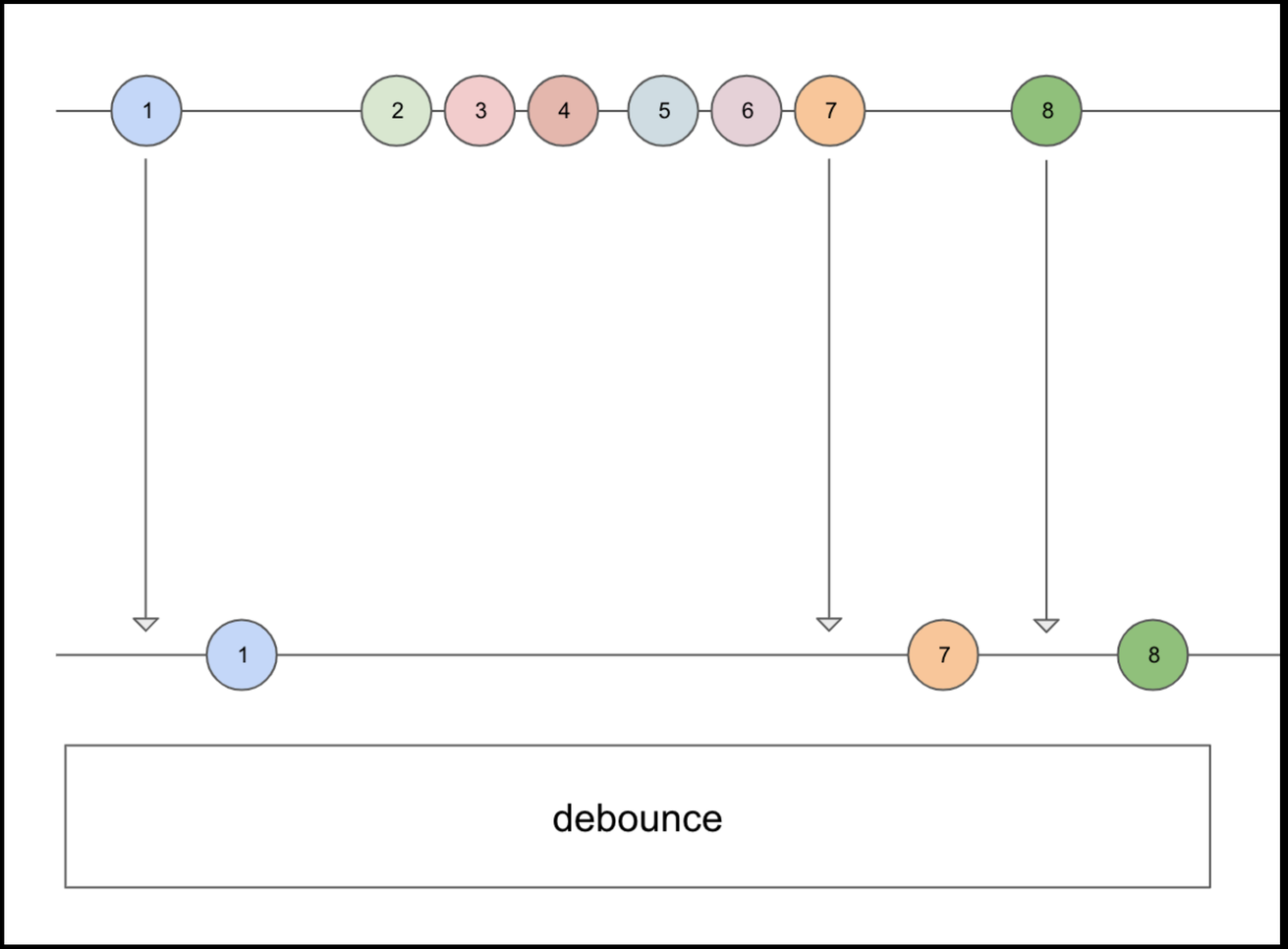
```swift
/// Throttle을 사용하기 위한 객체입니다.
final class Throttle {
    private var isBlocking: Bool = false

    /// Throttle을 적용합니다.
    /// - Parameters:
    ///   - block: 적용할 실행 구문을 작성합니다.
    ///   - seconds: 소수점 형태로 몇 초간 Throttle 할 건지 입력합니다. (기본 값 0.3)
    func run(seconds: Double = 0.3, _ block: @escaping () -> Void) {
        guard !isBlocking else { return }
        isBlocking = true
        block()

        Task {
            try await Task.sleep(for: .seconds(seconds))
            isBlocking = false
        }
    }
}
```


Callback

```swift
throttle.run {
    PlayerManager.shared.moveBackward()
}

throttle.run {
    Task { await PlayerManager.shared.moveForward() }
}
```

Debounce

이벤트가 일어날 때마다 항상 취소 후 시간 딜레이를 적용



https://docfriends.github.io/DevStrory/2019-01-29/swift-delay/

Debounce

적용 예제

Callback

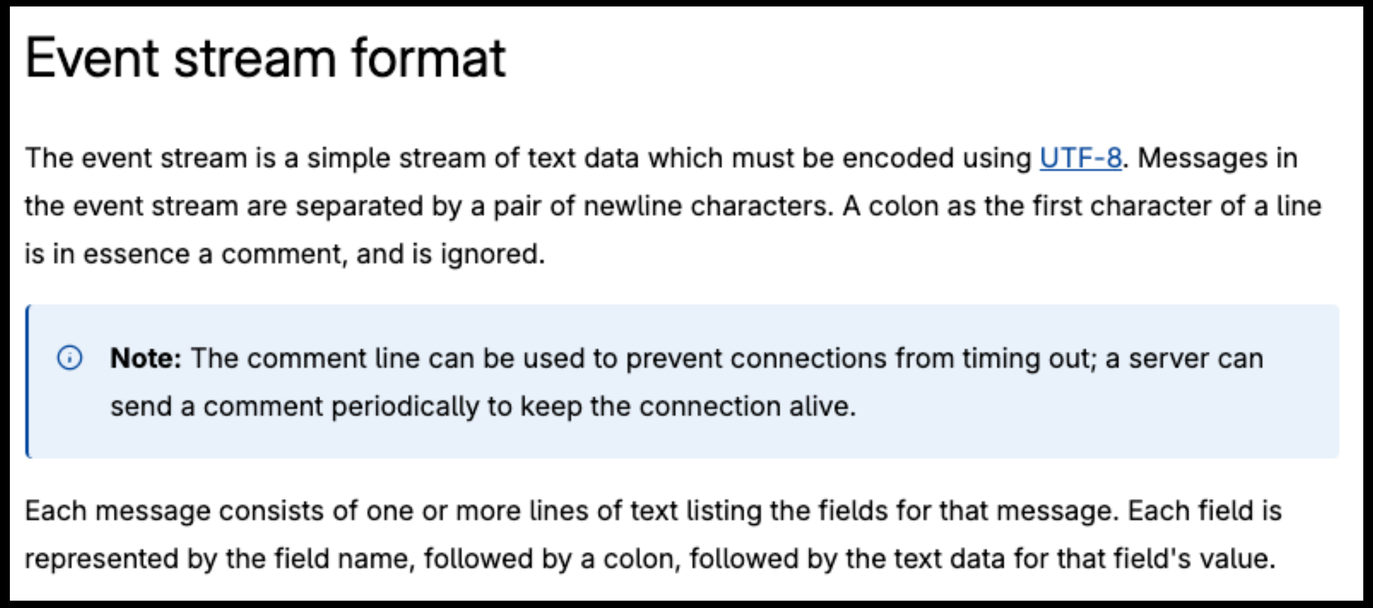
```swift
private var continuation: AsyncStream<String>.Continuation?
private var task: Task<Void, Never>?

init(upbitService: UpBitAPIService = UpBitAPIService()) {
    // ...
    observeStream()
}

/// 사용자 검색어 스트림을 관찰하고 검색을 트리거합니다.
///
/// - 해당 메소드는 0.3초의 디바운스가 적용되어 있습니다.
private func observeStream() {
    let stream = AsyncStream<String> { continuation in
        self.continuation = continuation
    }

    task = Task {
        for await keyword in stream.removeDuplicates().debounce(for: .seconds(0.3)) {
            await performSearch(with: keyword)
        }
    }
}
```


Callback

```swift
/// 검색 키워드를 스트림에 전달합니다.
/// - Parameter keyword: 사용자가 입력한 검색어 문자열입니다.
func sendKeyword(with keyword: String) async {
    continuation?.yield(keyword)
}
```

SSE Protocol

- HTTP 기반 단방향 통신 방식
- 서버에서 클라이언트로의 데이터 실시간 전송
- 하나의 응답 스트림에 여러 개의 데이터가 전달 될 수 있다⭐



Event stream format

The event stream is a simple stream of text data which must be encoded using UTF-8. Messages in the event stream are separated by a pair of newline characters. A colon as the first character of a line is in essence a comment, and is ignored.

> **Note:** The comment line can be used to prevent connections from timing out; a server can send a comment periodically to keep the connection alive.

Each message consists of one or more lines of text listing the fields for that message. Each field is represented by the field name, followed by a colon, followed by the text data for that field's value.

https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events/Using_server-sent_events?utm_source=chatgpt.com#event_stream_format

PreferenceKey

- defaultValue: 기본 값
- reduce: 여러 하위 뷰들의 값을 받고, 그 값들을 하나로 합친다.

```
struct CountKey: PreferenceKey {
    static var defaultValue: Int { 0 }
    static func reduce(value: inout Int, nextValue: () -> Int) {
        value += nextValue()
    }
}
```

```
Circle()
    .frame(width: 24, height: 24)
    .foregroundStyle(.blue)
    .preference(key: CountKey.self, value: value)
```

```
.onPreferenceChange(CountKey.self) { total in
    count = total
}
```

챗봇 고도화

자동 스크롤 구현

바닥에 닿아있는지 판단하는 공식

컨텐츠의 Height <= 스크롤한 Height + 뷰포트 Height + 임계값