SWIFT CONCURRENCY, ASYNC LET VA TASK GROUP O‘RTASIDAGI FARQLAR

Cho‘lliyev Shoxrux Ibadullayevich
Muhammad al-Xorazmiy nomidagi
Toshkent Axborot Texnologiyalari Universiteti

Annotatsiya: Ushbu maqola Swift Concurrency-da tuzilgan parallel ishlar (Structured Concurrency) – async let va Task Group o‘rtasidagi farqlarni tahlil qiladi. Maqola ularning hayotiy tsikli, xatolarni boshqarish va ishlatishdagi nozik jihatlarni misollar bilan ochib beradi. iOS dasturchilariga ushbu ikki mexanizmning qanday ishlashi va qaysi holatlarda ulardan qaysi biri afzal ekanligini tushunishga yordam berish maqsad qilingan. Xatolarni tarqatish (error propagation) va implicit bekor qilish (cancellation) kabi muhim jihatlar alohida ko‘rib chiqiladi.

Swift Concurrency-da tuzilgan parallel ishlar, ya’ni async let va Task Group haqida. Bu ikkisi bir xil maqsadda – bir nechta vazifani parallel ravishda bajarish uchun ishlatilsa-da, ularning hayotiy sikli va xatti-harakatlari biroz farq qiladi. Ushbu maqolada men ushbu farqlarni misollar bilan tushuntiraman va e’tibor berish kerak bo‘lgan nozik holatlarni ko‘rsataman.

Async let nima?

async let yordamida siz bir nechta vazifani parallel ravishda boshlashingiz mumkin. Masalan:

```
func fetchData() async {
    async let birinchi = fetchPart1()
    async let ikkinchi = fetchPart2()
    async let uchinchi = fetchPart3()
    let natija = await (birinchi, ikkinchi, uchinchi)
    print(natija)
}
```

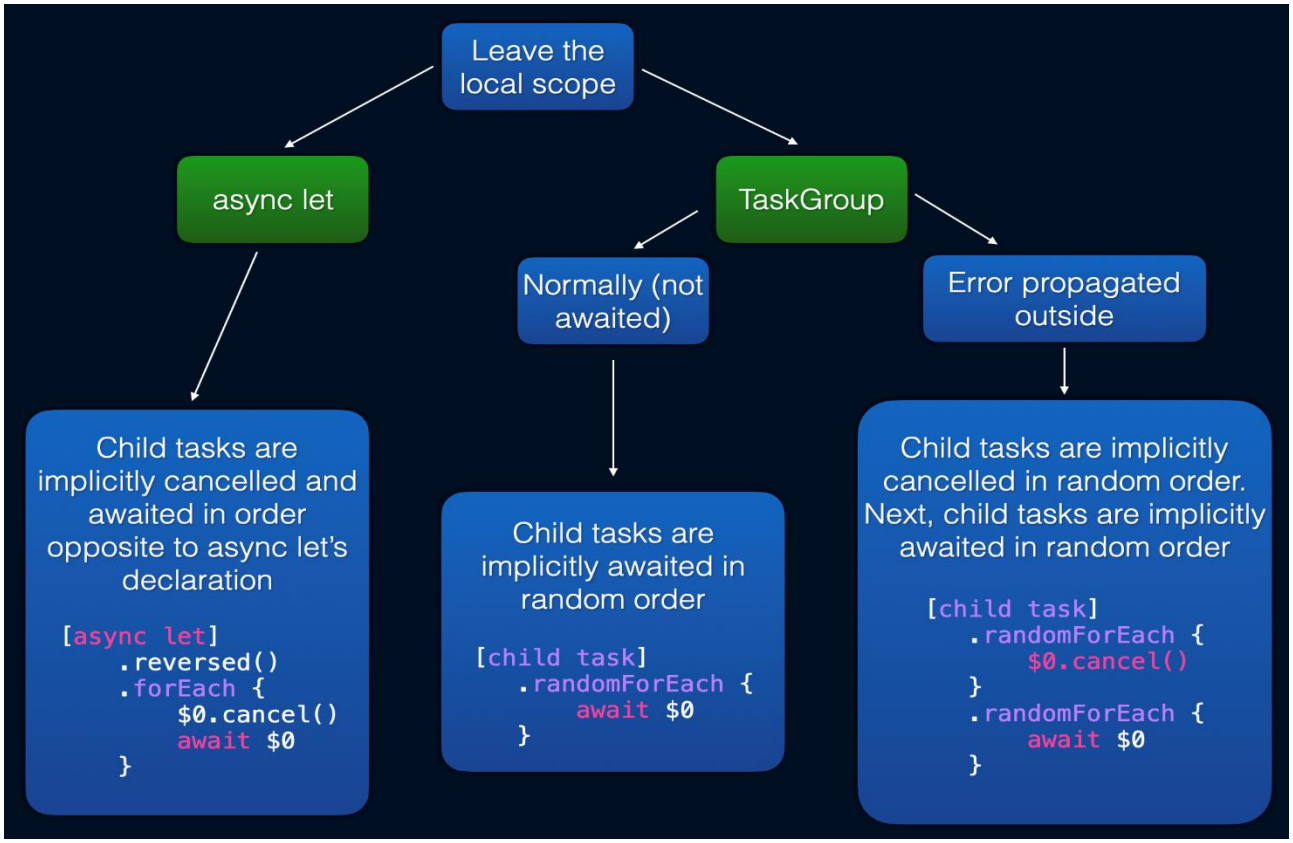
Bu yerda har bir async let yangi bola vazifa (child task) yaratadi. Bu vazifalar parallel ravishda ishga tushadi va standart parallel ijrochi (default concurrent executor) orqali bajariladi. async let nafaqat asinxron funksiyalar, balki sinxron funksiyalar yoki oddiy ifodalar bilan ham ishlaydi:

```
async let son = 123
async let matn = "Salom, dunyo!"
```

async letning hayotiy tsikli u yaratilgan mahalliy doira (local scope) – masalan, funksiya yoki do/catch bloki bilan bog‘liq. Agar bu doiradan chiqilsa, masalan, normal yakunlansa yoki xato yuzaga kelsa, barcha async let vazifalari avtomatik ravishda

bekor qilinadi va kutib turiladi (implicitly cancelled and awaited). Agar siz ularni aniq await qilmasangiz ham, doira oxirida bu jarayon avtomatik amalga oshiriladi.

E’tibor bering: async letda natijalarni await qilish tartibi ketma-ket bo‘ladi – tuple ichidagi elementlar chapdan o‘ngga qarab kutib chiqiladi. Bu xatolarni boshqarishda muhim rol o‘ynaydi, keyinroq bu haqda batafsilroq gaplashamiz.



Task Group nima?

Agar sizga dinamik sonli vazifalarni parallel ravishda boshqarish kerak bo‘lsa, Task Groupdan foydalanasiz:

```
func fetchData(count: Int) async {
    var natijalar = [String]()

    await withTaskGroup(of: String.self) { guruh in
        for indeks in 0..<count {
            guruh.addTask {
                await self.fetchPart(indeks)
            }
        }
        for await natija in guruh {
            natijalar.append(natija)
        }
    }
    print(natijalar)
```

```
}
```

Task Groupning hayotiy tsikli withTaskGroup funksiyasi ichidagi yopiq blok (closure) bilan bog‘liq. Agar blok normal yakunlansa va bola vazifalar await qilinmasa, ular faqat kutib chiqiladi, lekin bekor qilinmaydi. Agar xato tufayli blokdan chiqilsa, vazifalar bekor qilinadi va kutib chiqiladi.

Task Groupda natijalar “birinchi tugagan – birinchi qaytariladi” printsipi bo‘yicha keladi, chunki u AsyncSequence sifatida ishlaydi. Bu async letdan farqli jihati.

Hayotiy tsiklning nozik holatlari

Keling, bu ikki mexanizmning hayotiy tsiklidagi farqlarni misollar bilan ko‘rib chiqamiz.

Agar vazifalar await qilinmasa va doira normal yakunlansa

async let: Doira normal yakunlansa, barcha bola vazifalar bekor qilinadi va kutib chiqiladi. Masalan:

```
func tezkor() async {
    print("tezkor boshlandi")
    try await Task.sleep(nanoseconds: 5_000_000_000)
    print("tezkor tugadi")
}

func sekin() async {
    print("sekin boshlandi")
    try await Task.sleep(nanoseconds: 10_000_000_000)
    print("sekin tugadi")
}
func boshlash() async {
    async let t = tezkor()
    async let s = sekin()
    print("doiradan chiqyapman")
}
```

Natija: “doiradan chiqyapman”dan keyin vazifalar teskari tartibda bekor qilinadi – avval sekin, keyin tezkor.

Task Group: Agar blok normal yakunlansa, vazifalar faqat kutib chiqiladi, lekin bekor qilinmaydi:

```
await withTaskGroup(of: Void.self) { guruh in
    guruh.addTask { await tezkor() }
    guruh.addTask { await sekin() }
    print("blokdan chiqyapman")
}
```

Natija: Vazifalar to‘liq bajarilguncha kutib turiladi.

Agar xato yuzaga kelsa va doira tark etilsa

async let: Agar doirada xato yuzaga kelsa, vazifalar bekor qilinadi va kutib chiqiladi. Masalan, agar tezkor va sekin xato tashlasa, lekin faqat bittasi await qilinsa, ikkinchisi bekor qilinadi.

Task Group: Xato yuzaga kelsa, vazifalar bekor qilinadi va kutib chiqiladi, lekin tartib tasodifiy bo‘ladi.

Xatolarni boshqarishdagi farqlar

async letda xatolarni await qilish tartibi muhim. Masalan, agar birinchi vazifa xato tashlasa, ikkinchisi await qilinmasdan bekor qilinadi. Task Groupda esa birinchi xato tashlagan vazifa darhol qaytariladi, qolganlari esa bekor qilinishi mumkin.

Xulosa

Agar sizda xatolarni tez aniqlash va “fail fast” strategiyasi muhim bo‘lsa, Task Group yaxshiroq tanlovdir, chunki uning xatolarni qaytarish tartibi aniqroq. async let esa statik sonli vazifalar uchun qulay, lekin await tartibiga e’tibor berish kerak.

Swift Concurrency-ni o‘rganishda ushbu nozik jihatlarni hisobga oling. Amaliyot bilan bu mexanizmlarni yaxshiroq tushunasiz!

Foydalanilgan adabiyotlar:

1. Apple Developer Documentation – Swift Concurrency bo‘yicha rasmiy hujjatlar.
2. Swift Evolution – async let taklifi.
3. WWDC sessiyalari – Structured Concurrency haqida ma’lumotlar.