Icicle keretrendszer

IceyLeagons

Tóth Tamás, Kissik Márton

Felkészítő tanárok: Rozgonyi-Borus Ferenc, Motesiczki Ottó

SZTE Vántus István Gyakorló Zeneművészeti Szakgimnázium,

6722 Szeged, Tisza Lajos krt. 79-81

Esztergomi Dobó Katalin Gimnázium, 2500 Esztergom, Bánomi út 9.

1. Mi az az Icicle?

Az Icicle, egy innovatív, Minecraft szerver-oldali plugin készítésére szolgáló, a Spring által ihletett Java/Kotlin keretrendszer. A fejlesztéséhez a szikrát az adta, hogy a Minecraft fejlesztés támogatására nem igazán található alkalmas keretrendszer, és ha vannak is újrafelhasználható eljárások, azok nem egy központi tárházban találhatók, hanem egy-egy specifikus könyvtárban, több különálló projektben érhetők el. A keretrendszer elkészítése egy hullámvasút volt, eredetileg csak néhány hasznos funkciót tartalmazó eljárásgyűjteményből fejlődött ki. A keretrendszert háromszor írtuk újra, a műszakilag megfelelő és legelegánsabb változatot keresve. Keretrendszerünk modulokba van rendezve, amelyek – helytakarékosság céljából – csakis akkor kerülnek letöltésre, betöltésre, valamint frissítésre, ha tényleg szükség van rájuk. Az Icicle-nek a szíve az ún. „***core***" modul, mely tartalmazza a saját „***bean***" és „***dependency injection***" megoldásainkat, a különböző annotáció kezelőket és a keretrendszerhez nélkülözhetetlen alapfunkciókat. A projekt felépítése alapján a „***core***"-t egy szabadon választott környezetbe kell implementálni az interfészek segítségével, így utat nyitva a szerteágazó felhasználásra. Igyekeztünk minden ponton testreszabhatóvá és könnyen használhatóvá tenni szoftverünket, ezáltal biztosítva annak jövőbeli felhasználási lehetőségét is.

2. Probléma megoldásának menete

Kitűzött célunk az egyszerűség, testreszabhatóság, és a kis méret volt. Bármennyire is gondoltuk a feladatot egyszerűnek, nem bizonyult annak – ezt az újraírások száma is jól mutatja. Az első nagy feladat a „***bean***"-ek és a „***dependency injection***" megvalósítása volt. Mivel nem akartunk sok külső szoftvert felhasználni (azok nagy mérete, és lassúsága miatt), így megírtuk a saját megoldásainkat.

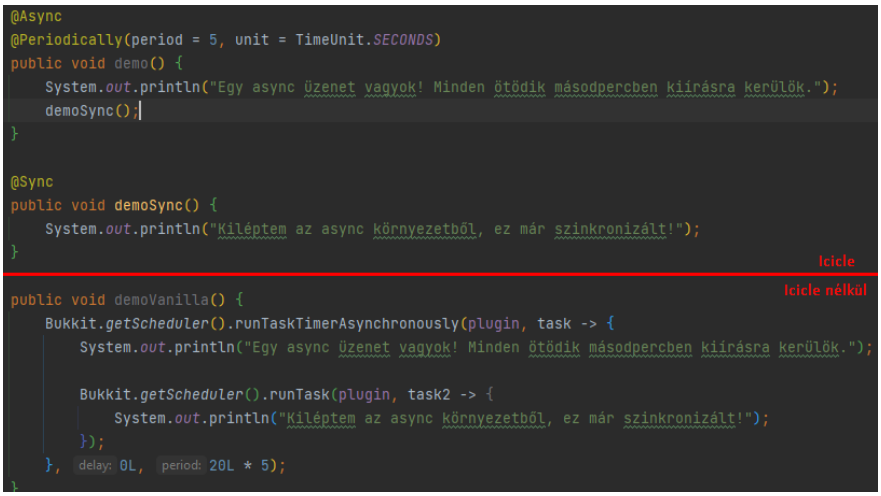
2.1. Babok és függőségek (dependency injection)

Rendszerünkben minden osztály, amelynek az annotációs fája – ehhez is saját megoldást kellett keresnünk, mivel a Java alapból nem kezeli az „*annotációk annotációit*" – tartalmazza az ún. @AutoCreate annotációt, automatikusan létre lesz hozva, és az osztály függőségei is injektálásra kerülnek. Ehhez a

rendszerhez került hozzá a körkörös függőség tesztelése, az osztályok proxizása, és az osztályok futás közbeni létrehozása (az utóbbi kettőhöz egy ***ByteBuddy*** nevű könyvtárat használtunk).

2.2. Osztályok proxyzása, az AOP jelenléte

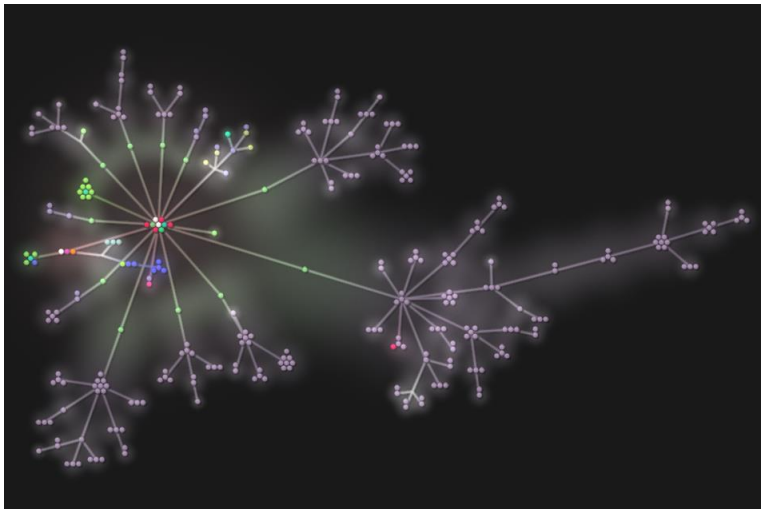
A proxizás az *aspektus-orientált* programozás egyik fő elve. Segítségével képesek vagyunk egy metódus lefutását vagy egy osztály viselkedését alterálni úgy, hogy az eredeti kód elé, közé, illetőleg utána saját logikát injektálunk, ami akár még az eredeti kód lefutását is megakadályozhatja, vagy módosíthatja annak paramétereit, illetőleg a környezetét. Ez a megoldás lehetőséget ad letisztultabb, látszólag „natív" kódok írására, amelyre példát az 1. ábra mutat: a program egy üzenetet ír ki minden 5. másodpercben, egyet szinkronizálva, egyet pedig aszinkron módon. (Míg az Icicle alapút nem, addig a szokványos módszert betöltéskor egyszer manuális meg kell hívni).



1. ábra: Proxy bemutatása

2.3. Felkészülés a jövőre

A jövőben, a saját és mások munkájának megkönnyítése érdekében, a saját annotációk (ebbe beletartozik az @AutoCreate, valamint a saját dependency injection/autowiring annotációk) kezelésére egyszerű módot kínálunk: pár sorban hozzá lehet adni a keretrendszerhez új funkciókat, az annotáció-kezelők segítségével. Továbbá, mivel a Kotlin programnyelv manapság egyre népszerűbb, annak támogatása is bekerült a projektbe. Annak ellenére, hogy a Kotlin és a Java képes integrálódni egymással, a „***bean***"-ek létrehozása során mégis számos plusz funkciót kellett megvalósítanunk, hogy megfelelően kezeljük ezt a feltörekvő nyelvet is.



2. ábra: A projekt kiterjedése 2021. december 31-én (Gource-ábra)

2.4. Alapvető funkciók és kiegészítő modulok

A „***core***” tartalmaz egy konfigurációkezelőt, és egy lokalizációs rendszert is a többnyelvű projektek támogatására. Utóbbi egy kisméretű, egyszerűen bővíthető „script nyelvet” - hasonló az Excelben található függvények formátumához - is használ. Példa erre, egy névelő kereső, ami a ***targy*** paraméter elé a megfelelő névelőt írja ki:
{IF(SW(targy, ‘a','á','o','ó','u','ú','e','é','i','í','ö','ő','ü','ű'), ‘Az’, ‘A’)} {targy} az enyém.

Ezen a ponton kezdtünk el több különböző feladaton dolgozni párhuzamosan. Így készült el, a Minecraft-tól még távolálló Gradle bővítményünk, saját szerializáció könyvtárunk, modul betöltőnk. Majd követték őket a Minecraft modulok, vagyis a parancsrendszer, az NMS (**N**et.**M**inecraft.**S**erver) könyvtár, valamint a protokoll könyvtár. Készül továbbá egy GUI és adatbázis kezelő alrendszer is. Természetesen ezeknek az eszközöknek a támogatásában nem mi vagyunk az elsők, viszont mi egy rendkívül egyszerű formára törekedtünk, minél kevesebb kód használatával, ami a szoftver méretét csökkenti és a teljesítményét jelentősen növeli.

A modulok közül sok időt fektettünk a parancsrendszerre, hiszen a legtöbb esetben, ezen keresztül használunk egy-egy Minecraft plugint. Ez a rendszer követi az eddigi mintákat, vagyis az egyszerű és gyors használatot annotációkkal, illetőleg az erőteljes testreszabhatóság lehetőségével. Jelen esetben az ún. middleware-ekkel a parancs beírása és a végrehajtása között lehet beavatkozni az eseményekbe, a paraméter feldolgozókkal pedig a parancs, illetőleg metódus paramétereit konvertáljuk a megfelelő objektumra egy szöveges bemenetet használva.

```java
@CommandManager("test")
public class TestCommand {

    @PlayerOnly
    @Command(value = "four", returnsTranslationKey = false)
    public String asd2(@CommandSender Player player, String arg, @Optional Player argOpt) {
        return "Executed2 sender: " + player.getName() + " | arg: " + arg + " opt: " + argOpt;
    }
}
```

3. ábra: Példa egy parancs készítésére (9 sor, a szokásos 50+ helyett)

3. Elért eredmények

Keretrendszerünk készítésére és a precizitásra rengeteg időt fordítottunk, amit a jövőbeli projektjeink fejlesztése során kapunk majd vissza, hiszen a hosszú hagyományos kód helyett csak pár sort kell majd írjunk. Ha további funkcióra lesz szükség, akkor csak írunk egy kezelőt a saját annotációkra alapozva. Bármi ami a keretrendszerben rendelkezésre áll, a gyors és hatékony programozást támogatja, ami a Gradle és a jövőben az IntelliJ IDEA bővítményünk segítségével még produktívvá tehető.

Projektünk nyílt forráskódú, jelenleg is fejlesztés alatt áll (még béta). Honlapja megtalálható a https://icicle.iceyleagons.net/ címen. Utoljára egy kis érdekesség: projektünk 2021. december 31-én 18 792 sorból áll.