Java 9 alkalmazás architektúra

EXPERTISE WITH NO LIMITS

Témakörök

- Modularitás
- Service-k
- Moduláris JVM
- Custom runtime image
- Migráció

Modularitás

Java 9 előtt:

- A fejlesztési, release egység: JAR
- Classpath
- Bonyolult verziózás
- A Classpath-on keresés rossz verzió használatához vezethetett
- JAR nem tartalmaz elérési és egyéb meta-adatot

Modularitás

Objektum orientáltság Java 8-ban:

- Class szintű kódújrahasznosítás
- Öröklődés, interface-k
- Elérési módosítók
- Erős típusosság
- Dependenciák

Modularitás

Release a Java 8-ban:

- JAR-ok egységbe zárják a library, application file-okat
- Használatkor a Classpath-hoz adódik hozzá
- Néha több száz JAR a Classpath-on

Modularitás

Elérési módosítók Java 8-ban:

- Public
- Protected
- <default>
- private

Modularitás

Java 9:

- A fejlesztési, release egység: module
- Támogatja a “nagyban” fejlesztést
- Beépítve a Java nyelvbe
- Minden szinten használható (Applikáció, library, JVM)
- Célja: megbízhatóság, karbantarthatóság, biztonság

Modularitás

Java 9 modulok:

- Új elérési módok (változatlan módosítók):
  - Public
  - Public egy célzott modulnak
  - Public modulon belül
  - Protected
  - <default>
  - private

Modularitás

Java 9 module-info.java:

Az exports, requires, exports to, requires transitive kulcsszavak:

```
module model {
    exports hu.houg.book;
    exports hu.houg.author;
    exports hu.houg.data.author to backend, backend_implementation;
    exports hu.houg.data.book to backend, backend_implementation;
}
```

Modularitás

Java 9 modulok:

Az elérési módosítok mellett dependenciakezelés:

```
module backend {
    requires transitive model;
    exports hu.houg.authorservice;
    exports hu.houg.bookservice;
}
```

Service

Modularitás segítségével service-ek:

- User-provider architektúra
- Egyszerűsíti a module-gráfot
- Runtime választhatunk providert

Service

Java 9 API:

```
module backend {
    requires transitive model;
    exports hu.houg.authorservice;
    exports hu.houg.bookservice;
}
```

Service

Java 9 Provider:

```
module backend_implementation {
    exports hu.houg.authordefaultservice;
    exports hu.houg.bookdefaultservice;
    requires backend;
    provides hu.houg.authorservice.AuthorService
            with hu.houg.authordefaultservice.DefaultAuthorService;
    provides hu.houg.bookservice.BookService
            with hu.houg.bookdefaultservice.DefaultBookService;
}
```

Service

Java 9 User:

```
module frontend {
    requires backend;
    //requires backend_implementation;
    uses hu.houg.bookservice.BookService;
    uses hu.houg.authorservice.AuthorService;
}
```

Service

Java 9 ServiceLoader class:

```
public class Main {

    public static void main(String[] args) {

        BookService bookService;

        ServiceLoader<BookService> bookServices =
ServiceLoader.load(BookService.class);

        bookService = bookServices.findFirst().get();

        System.out.println(bookService.getRandomBook().getTitle());

    }

}
```

Moduláris JVM

Java 9 JVM:

- Az eddigi monolit JVM felbontva ~90 modulra
- Minden modul egy-egy jól definiált funkcionalitás (pl Logging, JavaFX)
- Növeli a skálázhatóságot
- Javítja a biztonságot
- Performancia

Moduláris JVM

Java 9 JVM:

- $java –list-modules
- java.activation@9.0.4
- java.base@9.0.4
- java.compiler@9.0.4
- java.corba@9.0.4
- ...

Moduláris JVM

Java.se modul:

```
module java.se {
        requires transitive java.desktop;
        requires transitive java.sql;
        requires transitive java.xml;
        requires transitive java.prefs;
// ...
}
```

Custom runtime image

Custom Java runtime enviroment:

- Csak olyan modulok, amiket az alkalmazás is használ
- Csökkenthető a memórialábnyom
- Növelt biztonság

Custom runtime image

Custom Java runtime enviroment:

- Rövidebb felállási idő
- Jobb performancia
- Elkészíthető a Jlink eszköz segítségével

További változások

- Java Interface-ek módosítása
- JShell
- Stream API fejlesztések
- Multi-Release JAR

Java 10 újdonságok

EXPERTISE WITH NO LIMITS

Témakörök

- Lokális változó típus meghatározás
- Garbage Collection Interface
- Parallel Full GC a G1-hez
- Thread-local Handshakes
- Heap-allocation külső memória eszközön
- Új release calendar

Lokális változó típus meghatározás

- Új kulcsszó: var
- "Dinamikus" típusmeghatározás a lokális változókhoz

1. var list = new ArrayList<String>(); // infers ArrayList<String>
2. var stream = list.stream(); // infers Stream<String>

Garbage Collection Interface

- Garbage Collection Interface kialakítása
- Egyszerűsíti a GC algoritmusok használatát

Parallel Full GC a G1-hez

- G1 GC a default
- A Full GC esetén parallelizáció

Thread-local Handshakes

- Lehetőség callbackek hívására egyes threadeken, teljes JVM safepoint nélkül
- Ez lehetőséget ad a JVM-nek csupán egy-egy thread megállítására
- Ezzel könnyebbé válik a stacktrace-ek begyűjtése, monitorozás, javul a thread locking rendszer, a memóriakezelés

Heap-allocation külső memória eszközön

- Konfigurálható alternatív memória eszköz heap allokálására
- Ezzel lehetőség nyílik az alacsonyabb rendű processek másik eszközön való futtatására több JVM-mel rendelkező rendszerek esetén

Release Calendar és további újdonságok

- Új, fél éves release ciklusok, LTS verziók
- Root Certificates: Célja, hogy közelebb hozza az OpenJDK-t és az Oracle JDK-t, biztonsági javítások az OpenJDK-ban
- Kísérleti JIT compiler
- Egyéb kódbázis tisztántartás

Java 11 újdonságok

EXPERTISE WITH NO LIMITS

Témakörök

- Lokális változó szintaxis a lambda kifejezésekhez
- Egy file-os kódbázis futtatás
- Standard HTTP kliens
- API-k változásai
- Nest-based access controll
- Dynamic Class-file Constants
- Epsilon Garbage collector
- Flight Recorder

Témakörök

- Low-overhead heap profiler
- Transport Layer Security 1.3
- ZCG garbage collector
- Deprecated és Unsupported feature-ök

Lokális változó szintaxis a lambda kifejezésekhez

Eddig:

```
1.  list.stream()
2.      .map(s -> s.toLowerCase())
3.      .collect(Collectors.toList());
```

Mostantól:

```
1.  list.stream()
2.      .map((var s) -> s.toLowerCase())
3.      .collect(Collectors.toList());
```

Lokális változó szintaxis a lambda kifejezésekhez

És amiért jó:

```
1. list.stream()
2.         .map((@Notnull var s) -> s.toLowerCase())
3.         .collect(Collectors.toList());
```

Egy file-os kódbázis futtatás

Ha egy file-ból áll a kódbázis, futtatható közvetlenül:

```
java HelloWorld.java
```

Ha akarunk átadni változókat:

```
java -classpath /home/foo/java Hello.java Bonjour
```

```
javac -classpath /home/foo/java Hello.java
java -classpath /home/foo/java Hello Bonjour
```

Egy file-os kódbázis futtatás

- A filenév utáni paraméterek a futtatásnál kerülnek átadásra az alkalmazásnak
- A filenév előtti paraméterek a fordítás után adódnak át a futtatáshoz
- A filenév előtti paraméterek a compilernek is átadódnak, ha szükségesek a fordításhoz

Standard HTTP kliens

- A Java 9-ben már elindult inkubátormodulként a standardizált HTTP kliens fejlesztése
- A Java 11 SE Standard már tartalmazza ezt a fejlesztést
- Új modul: java.net.http
- Fő típusok:
  - HttpClient
  - HttpRequest
  - HttpResponse
  - WebSocket

API-k változásai

- IO változások:
  - java.io.FileReader és java.io.FileWriter: új konstruktor amelyben a Charset definiálható
  - NullStream-ek, NullWriter, NullReader
- String változások:
  - isBlank()
  - lines()
  - repeat(int)
  - strip(), stripLeading(), stripTrailing
- java.nio.file.File:
  - Charset támogatás

API-k változásai

- Predicate:
  - Predicate.not(Predicate)

Eddig:

```
1. lines.stream()
2.         .filter(s -> !s.isBlank())
```

Ezután:

```
1. lines.stream()
2.         .filter(Predicate.not(String::isBlank))
```

Nest-based access control

```
1.  public class Outer {
2.      private int outerInt;
3.      class Inner {
4.        public void printOuterInt() {
5.          System.out.println("Outer int = " + outerInt);
6.        }
7.       }
8.      }
```

Nest-based access control

```
1.  public class Outer {
2.       private int outerInt;
3.      public int access$000() {
4.        return outerInt;
5.       }
6.     }
7.      class Inner$Outer {
8.       Outer outer;
9.       public void printOuterInt() {
10.        System.out.println("Outer int = " + outer.access$000());
11.       }
12.     }
```

Nest-based access control

- 3 új metódus a java.lang.Class osztályban:
  - Class getNestHost()
  - Class[] getNestMembers()
  - boolean isNestmateOf(Class)

Dynamic Class-file Constants

- Egy új constant-pool bevezetése: CONSTANT_Dynamic
- Ezen értékek nem állítódnak be fordítási időben
- Egy bootstrap metódus segítségével runtime történik a konstansok beállítása

Epsilon Garbage Collector

- Az Epsilon Garbage Collector egy új GC, amely allokálja a memóriát, de nem szabadítja azt fel

MIÉRT?!

Epsilon Garbage Collector

- Első ránézésre két felhasználási terület lehetséges:
  - Benchmarking: Először futtatjuk az alkalmazásunkat az Epsilon GC-vel, majd ugyanezeket a teszteket elvégezzük a kívánt GC-vel és összehasonlítjuk a metrikákat
  - Nagyon rövid életű taskok elvégzésére, amikor garantálható, hogy nem fog elfogyni a heap, hiszen nincs overhead (még statisztikagyűjtés sem)
- Ha elfogyott a heap, 3 féle módon tudunk bedőlni:
  - OutMemoryError
  - Heap dump
  - Hard Fail

Flight Recorder

- Az Oracle JDK kiegészítő terméke volt, bekerült az OpenJDK-ba is, illetve az Oracle JDK-ba
  - API-kat biztosít adatok eventekként való feldolgozására, előállítására
  - Buffer mechanizmus és bináris adatok támogatása
  - Event filtering
  - OS, JVM és Java package eventek
  - Modulok: jdk.jfr, jdk.management.jfr

Low-overhead heap profiling

- Google fejlesztése
- JVM memory profiler, ami lehetővé teszi:
  - A defaultként való beállítást a kellően alacsony overhead miatt
  - Elérhetőség egy jól definiált interface-n keresztül
  - Minden allokáció monitorozható vele
  - Implementácó független
  - Élő és halott Java objektumokról is tud adatot gyűjteni

Transport Layes Security 1.3

- A TLS 1.3 támogatás kialakítása
- Jelentős fejlesztés biztonsági és performanciális szempontból is

ZCG Garbage Collector

- Low-latency GC nagy memóriaigényű (több GB) alkalmazásokhoz
- 1 generáció használata
- Alkalmazással konkurens GC
- Read barrier
- NEM ÁLTALÁNOS HASZNÁLATRA

Depricationés Unsupported feature-ök

- JavaEE modul eltávolítása
- Corba modul eltávolítása
- Nashorn Javascript Engine deprication
- Pack200 deprication