NVP – Nebenläufige und Verteilte Programme

Aufgabenblatt 15

Aufgabe 1

Betrachten Sie das Problem der Brückenüberwachung (Aufgabe 3 von Aufgabenblatt 5) und stellen Sie Ihre Lösung auf ein Szenario um, bei dem die Autos eigenständige Prozesse sind, die über Internet mit dem Brückenmonitor kommunizieren. Ein Auto das die Brücke treten will:

- baut eine Verbindung zum Server auf und sendet eine Anfrage.
- Es wartet die Antwort ab, überquert die Brücke und meldet sich ab.
- Dann schließt das Auto die Verbindung

Der Brückenmonitor soll reaktiv gestaltet werden. Verwenden Sie Akka-I/O.

Aufgabe 2

Im Gegensatz zu RMI[1] werden Klassendateien (`xy.class`) nicht bei Bedarf automatisch von einer JVM-Instanz zu einer anderen transferiert. In einer verteilten Anwendung muss darum in jedem Knoten der Code jeder lokal verwendeten Klasse lokal präsent sein: d.h. die verwendeten `class`–Dateien müssen auf dem lokalen *class path* zu finden sein.

Jede Java-Anwendung kann aber mit eigenen Klassenladern ausgestattet werden, mit denen der Prozess des Ladens von benötigtem ausführbarem Code in beliebiger Art erweitert werden kann. Es sollte darum möglich sein, auch in einem verteilten System auf Basis von Aktoren Klassencode bei Bedarf von einem andern Knoten zu laden.

Akka unterstützt das Klassenladen im Objekt `ActorSystem` mit einer `apply`-Methode, die einen Klassenlader als Parameter annimmt:

```
def apply(name: String, config: Config, classLoader: ClassLoader): ActorSystem
  /* Creates a new ActorSystem with the specified name, the specified Config, and
     specified ClassLoader */
```

Hier könnte ein eigener Klassenlader eingesetzt werden, der Klassen bei Bedarf von einer entfernten Quelle laden kann.

Der Einfachheit halber wollen wir kein “bedarfsgetriebenes” Klassenladen realisieren, sondern eines, bei dem Klassencode an einen anderen Knoten transferiert werden werden bevor er dort dann verwendet werden kann.

Ein entsprechendes System kann recht einfach implementiert werden: Eine *Registry* für Klassen

- wird über das Netz mit Bytecode gefüllt und
- wird lokal vom Klassenlader genutzt.

```
package remote_class_loading

import akka.actor.{Actor, ActorSystem, Props}

class RegistryActor(registry: Registry) extends Actor {

  def receive = {
    case RegisterRemoteMsg(name, bytes) =>
      println(s"\tRegistryActor registers $name -> $bytes")
      registry.register(name, bytes)

    case x: Any => println(s"\tRegistryActor received unknown message $x")
```

[1] https://docs.oracle.com/javase/tutorial/rmi/overview.html

```
  }
}

class Registry(system: ActorSystem, registryClassloader: RegistryClassloader) {

  var registeredClasses : Map[String, Array[Byte]] = Map()

  val registryActor = system.actorOf(Props(classOf[RegistryActor], this), name =
      "registryActor")

  println("registry actor stated with path " + registryActor.path)

  registryClassloader.setRegistry(this)

  println("registry ready to accept registrations")

  /**
    * register a class
    * @param name  the name of the class
    * @param classBytes the class code of the class
    */
  def register(name : String, classBytes : Array[Byte]) {
    println("RegistryClassLoader::register class " + name)
    registeredClasses += (name -> classBytes)
  }

}
```

```
package remote_class_loading

import java.net.{URL, URLClassLoader}
import java.security.cert.Certificate
import java.security.{AllPermission, CodeSource, Policy, ProtectionDomain}

/**
  * A class loader that finds class code in a registry
  */
class RegistryClassloader(parent: ClassLoader) extends URLClassLoader(Array[URL](),
    parent) {

  var registry: Registry = null

  def setRegistry(registry: Registry): Unit = { this.registry = registry }

  override
  protected def findClass(name: String) : Class[_] = {
    println(s"RegistryClassLoader::findClass $name")

    val cert: Array[Certificate] = null
    val myCs = new CodeSource(null, cert)
    val pc = Policy.getPolicy().getPermissions(myCs)
    pc.add(new AllPermission())
    val pd = new ProtectionDomain(myCs, pc, this, null)

    // load from parent
    var result = findLoadedClass(name)
    println(s"RegistryClassLoader::findClass $name loadedClass = $result")
    if (result == null) {
      try {
        result = findSystemClass(name)
        println(s"RegistryClassLoader::findClass $name systemClass = $result")
```

```
    } catch {
      case e: Exception => println(s"RegistryClassLoader::findClass $name systemClass
          exception = $e") /* ignore */
    }
   }
   if (result != null) {
    println(s"RegistryClassLoader::findClass $name resolved to systemClass $result")
    return result
   }

   // load from registry
   try {
    val classBytes = registry.registeredClasses(name)
    println(s"RegistryClassLoader::findClass $name registerdClass = $classBytes")
    result = defineClass(name, classBytes, 0, classBytes.length);
   } catch {
    case e: Exception =>
     println(s"RegistryClassLoader::findClass $name registerdClass exception = $e")
     throw new ClassNotFoundException(name);
   }

   println(s"RegistryClassLoader::findClass $name resolved to registered class $result")
   result
  }
}
```

Auf “der anderen Seite” muss ein *Registry Client* den Bytecode der Klassen versenden, der in einem entfernten Knoten verwendet werden soll.

```
import akka.actor.{ActorRef, ActorSelection, ActorSystem}
import akka.util.Timeout

import scala.concurrent.{Await, Future}
import scala.concurrent.duration._
import scala.language.postfixOps

class RegistryClient(RemoteSystemName: String, host: String, port: Int)(implicit
    actorSystem: ActorSystem) {

 implicit val timeout = Timeout(3 seconds)

 println("try to access remote registry actor " +
     s"akka.tcp://${RemoteSystemName}@${host}:${port.toString}/user/registryActor")

 private val registryActorS : ActorSelection =
  actorSystem.actorSelection(
    s"akka.tcp://${RemoteSystemName}@${host}:${port.toString}/user/registryActor")

 private val futureRegistryActorR : Future[ActorRef] = registryActorS.resolveOne
 private val registryActorR : ActorRef = Await.result(futureRegistryActorR , 5.seconds)

 def registerRemote(clazz: Class[_]): Unit = {
  val classesToBeSent = ClassBytesLoader.loadClass(clazz)
  for ((name, bytes) <- classesToBeSent) {
   registryActorR ! RegisterRemoteMsg(name, bytes)
  }
 }

}
```

Die Sache ist etwas komplexer, als auf den ersten Blick zu vermuten ist, da zu einer Scala-Klasse beliebig viele innere

Klassen generiert werden können, deren genaue Samen wir nicht kennen. Wir versuchen sie alle zu finden mit folgender *ClassBytesLoader* der allen eventuell relevanten Klassencode aufsammelt:

```
package remote_class_loading

import java.nio.file.{DirectoryStream, Files, Path, Paths}

object ClassBytesLoader {

  def loadClass (clazz: Class[_]) : Map[String, Array[Byte]] = {
    val basePath : Path =
        Paths.get(clazz.getProtectionDomain.getCodeSource.getLocation.toURI)
    val relName = clazz.getName().replace('.', '/')
    val fullPath = basePath.resolve(relName)
    val fileName = fullPath.getFileName()
    val fileNameHead = (fileName.toString()).split("\\.")(0)
    val parentDir = fullPath.getParent()

    var res : Map[String, Array[Byte]] = Map()
    val ds = Files.newDirectoryStream(
      parentDir,
      new DirectoryStream.Filter[Path]{
        def accept(file: Path) : Boolean =
            file.getFileName().toString().matches(fileNameHead+"(\\$.+)?\\.class")
      })
    try {
      val iter = ds.iterator()
      while (iter.hasNext()) {
        val p = iter.next()
        res += (((basePath.relativize(p)).toString().split("\\.")(0).replace('/', '.')) ->
            Files.readAllBytes(p))
      }
    } finally {
      ds.close
    }
    res
  }

}
```

Aufgabenstellung Schreiben Sie eine verteilte Echo-Anwendung, bei der der Code des Echo-Servers und der des Echo-Clients – zwei Aktoren – nur auf einem Knoten zur Verfügung stehen, der Client-Seite beispielsweise. Der Code des Server-Aktors soll von der Client-Seite zur Serverseite gesendet werden und dann soll dort ein Server-Aktor gestartet werden mit den anschließend der entfernte Client kommuniziert.