8.并发编程模型 Akka

8.1 Akka 介绍

写并发程序很难。程序员不得不处理线程、锁和竞态条件等等，这个过程很容易出错，而且会导致程序代码难以阅读、测试和维护。

Akka 是 JVM 平台上构建高并发、分布式和容错应用的工具包和运行时。Akka 用 Scala 语言写成，同时提供了 Scala 和 JAVA 的开发接口。

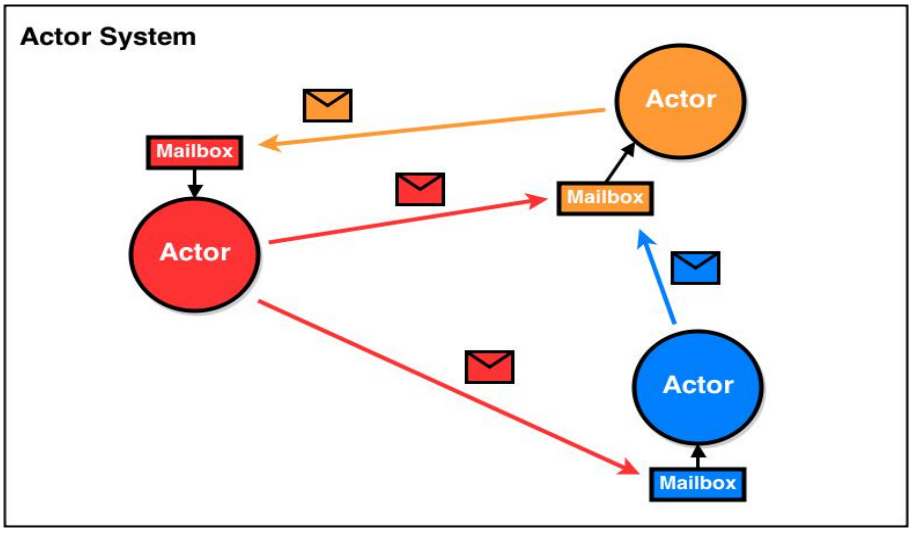
8.2 Akka 中 Actor 模型

Akka 处理并发的方法基于 Actor 模型。在基于 Actor 的系统里，所有的事物都是 Actor，就好像在面向对象设计里面所有的事物都是对象一样。但是有一个重要区别，那就是 Actor 模型是作为一个并发模型设计和架构的，而面向对象模式则不是。Actor 与 Actor 之间只能通过消息通信。

- 对并发模型进行了更高的抽象
- 异步、非阻塞、高性能的事件驱动编程模型
- 轻量级事件处理（1GB 内存可容纳百万级别个 Actor）

为什么 Actor 模型是一种处理并发问题的解决方案？

处理并发问题就是如何保证共享数据的一致性和正确性，为什么会有保持共享数据正确性这个问题呢？无非是我们的程序是多线程的，多个线程对同一个数据进行修改，若不加同步条件，势必会造成数据污染。那么我们是不是可以转换一下思维，用单线程去处理相应的请求，但是又有人会问了，若是用单线程处理，那系统的性能又如何保证。Actor 模型的出现解决了这个问题，简化并发编程，提升程序性能。



从图中可以看到，Actor 与 Actor 之前只能用消息进行通信，当某一个 Actor 给另外一个 Actor 发消息，消息是有顺序的，只需要将消息投寄的相应的邮箱，至于对方 Actor 怎么处理你的消息你并不知道，当然你也可等待它的回复。

Actor 是 ActorSystem 创建的，ActorSystem 的职责是负责创建并管理其创建的 Actor，ActorSystem 的单例的，一个 JVM 进程中有一个即可，而 Acotr 是多例的。
Pom 依赖：

```
<!-- 定义一下常量 -->
<properties>
    <encoding>UTF-8</encoding>
    <scala.version>2.11.8</scala.version>
    <scala.compat.version>2.11</scala.compat.version>
    <akka.version>2.4.17</akka.version>
</properties>

<dependencies>
    <!-- 添加 scala 的依赖 -->
    <dependency>
        <groupId>org.scala-lang</groupId>
        <artifactId>scala-library</artifactId>
        <version>${scala.version}</version>
    </dependency>

    <!-- 添加 akka 的 actor 依赖 -->
    <dependency>
        <groupId>com.typesafe.akka</groupId>
        <artifactId>akka-actor_${scala.compat.version}</artifactId>
        <version>${akka.version}</version>
    </dependency>

    <!-- 多进程之间的 Actor 通信 -->
```

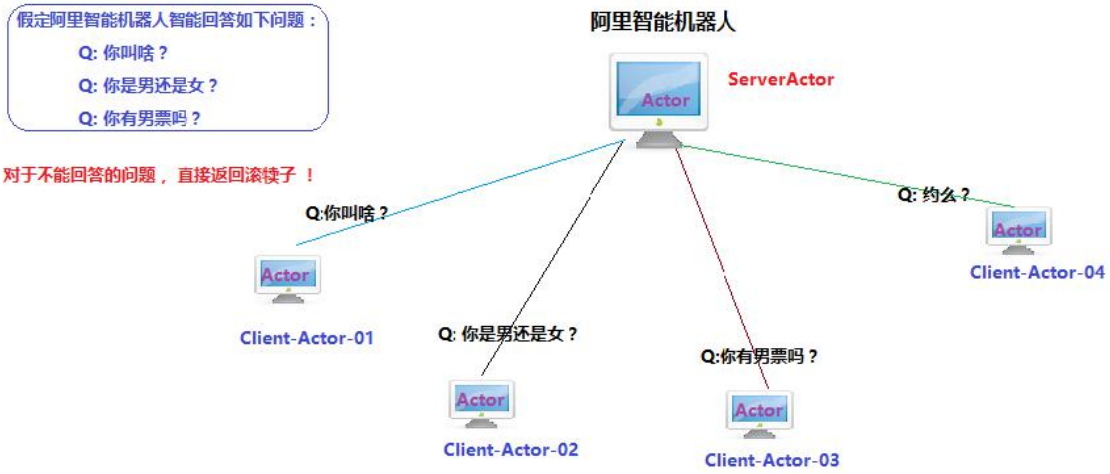
```
    <dependency>
        <groupId>com.typesafe.akka</groupId>
        <artifactId>akka-remote_${scala.compat.version}</artifactId>
        <version>${akka.version}</version>
    </dependency>
</dependencies>

<!-- 指定插件-->
<build>
    <!-- 指定源码包和测试包的位置 -->
    <sourceDirectory>src/main/scala</sourceDirectory>
    <testSourceDirectory>src/test/scala</testSourceDirectory>
    <plugins>
        <!-- 指定编译 scala 的插件 -->
        <plugin>
            <groupId>net.alchim31.maven</groupId>
            <artifactId>scala-maven-plugin</artifactId>
            <version>3.2.2</version>
            <executions>
                <execution>
                    <goals>
                        <goal>compile</goal>
                        <goal>testCompile</goal>
                    </goals>
                    <configuration>
                        <args>
                            <arg>-dependencyfile</arg>

<arg>${project.build.directory}/.scala_dependencies</arg>
                        </args>
                    </configuration>
                </execution>
            </executions>
        </plugin>

        <!-- maven 打包的插件 -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>2.4.3</version>
            <executions>
                <execution>
                    <phase>package</phase>
                    <goals>
```

```
                    <goal>shade</goal>
                </goals>
                <configuration>
                    <filters>
                        <filter>
                            <artifact>*:*</artifact>
                            <excludes>
                                <exclude>META-INF/*.SF</exclude>
                                <exclude>META-INF/*.DSA</exclude>
                                <exclude>META-INF/*.RSA</exclude>
                            </excludes>
                        </filter>
                    </filters>
                    <transformers>
                        <transformer
implementation="org.apache.maven.plugins.shade.resource.AppendingTransformer">
                            <resource>reference.conf</resource>
                        </transformer>
                        <!-- 指定 main 方法 -->
                        <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransfor
mer">
                            <mainClass></mainClass>
                        </transformer>
                    </transformers>
                </configuration>
            </execution>
        </executions>
    </plugin>
  </plugins>
</build>
```

8.3 案例--HelloActor

8.4 案例--PingPong

8.5 案例基于 Actor 的聊天模型



参数：

```
akka.actor.provider = "akka.remote.RemoteActorRefProvider"
akka.remote.netty.tcp.hostname = $host
akka.remote.netty.tcp.port = $port
```

1) 创建一个 Server 端用于服务客户端发送过来的问题，并作处理并返回信息给客户端！
2) 创建一个 Client 端，用于向服务端发送问题，并接收服务端发送过来的消息！

8.6 案例 Spark Master Worker 进程通信示例