Akka

Akka基于Actor模型，提供一个用于构建可扩展的（Scalable）、弹性的（Resilient）、快速响应的（Responsive）应用程序的平台。

# Actor

维基百科定义Actor模型，*在计算科学领域，Actor模型是一个并行计算（Concurrent Computation）模型，它把actor作为并行计算的基本元素来对待：为响应一个接收到的消息，一个actor能够自己做出一些决策，如创建更多的actor，或发送更多的消息，或者确定如何去响应接收到的下一个消息。*

Actor是Akka中最核心的概念，封装了状态和行为的对象，Actor之间可以通过交换消息的方式进行通信，每个Actor都有自己的收件箱（Mailbox）。通过Actor能够简化锁及线程管理，可以非常容易的开发出正确的并发程序和并行系统，Actor具有如下特性：

* 提供了一种高级抽象，能够简化在并发（Concurrency）/并行（Parallelism）应用场景下的编程开发
* 提供了异步非阻塞、高性能的事件驱动编程模型
* 超级轻量级事件处理（每GB堆内存几百万Actor）

实现Actor，继承akka.actor.Actor，实现receive方法，示例如下：

*import akka.actor.AbstractActor;*

*import akka.event.Logging;*

*import akka.event.LoggingAdapter;*

*public class MyActor extends AbstractActor {*

*private final LoggingAdapter log = Logging.getLogger(getContext().getSystem(), this);*

*@Override*

*public Receive createReceive() {*

*return receiveBuilder()*

*.match(String.class, s -> {*

*log.info("Received String message: {}", s);*

*})*

*.matchAny(o -> log.info("received unknown message"))*

*.build();*

*}*

*}*

Receive方法参数是PartialFunction<Object,BoxedUnit>，定义Actor处理的消息类型。

# Akka Demo

示例代码如下：

*public class HelloDemo{*

*public static void main(String[] args) {*

*ActorSystem system = ActorSystem.create("actor-demo-java");*

*ActorRef hello = system.actorOf(Props.create(Hello.class));*

*hello.tell("Bob", ActorRef.noSender());*

*try {*

*Thread.sleep(1000);*

*} catch (InterruptedException e) { // ignore }*

*system.terminate();*

*}*

*private static class Hello extends AbstractActor {*

*public Receive createReceive() {*

*return receiveBuilder()*

*.match(String.class, s -> {*

*System.out.println("Received String message:" + s);*

*}).build();*

*}*

*}*

*}*

第一部分代码是Akka应用程序基础架构：

1. 创建actor系统（ActorySystem）
2. 在system中创建一个actor（system.actorOf行，为所创建的actor返回一个actor引用）
3. 使用actor引用向actor发送消息
4. 关闭系统

其中Props对象用于Hello actor类型的配置属性，在其中设置参数。

第二段代码是Hello Actor定义，实现AbstractActor接口，实现方法createReceive，在方法中对于接受到的消息进行处理，本身仅是简单的在控制台中打印。

备注：

之所以在主程序中，将消息发送给actor之后，会有一次Thread sleep 1000形式的等待，才会关闭系统，是因为Akka actor是异步的，即使目标actor与发送方actor位于相同的JVM中，目标actor也不会立即开始执行，处理该消息的线程会将消息添加到目标actor的邮箱中。将消息添加到邮箱中会触发一个线程，以便从该邮箱中获取该消息并调用actor的receive方法来处理。

# Akka Multi-JVM

上面的例子，在同一JVM中进行消息传递，常规的使用是在不同的节点上有不同的Actor，

https://www.ibm.com/developerworks/cn/java/j-jvmc5/

http://blog.csdn.net/thkhxm/article/details/40182835

http://blog.csdn.net/csolo/article/details/52916009

http://shiyanjun.cn/archives/1168.html