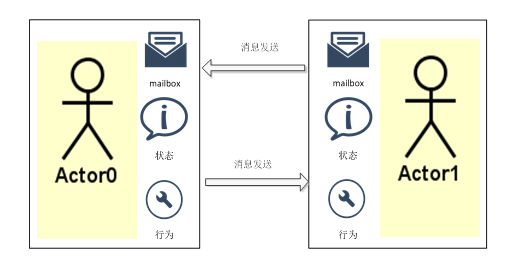
Actor编程模型

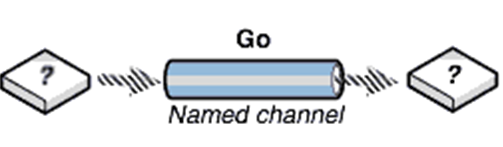
为了提高充分利用操作系统资源，使用并发编程，在同一台处理器上同时处理多个任务，即多个事件在同一时间间隔发生，充分利用处理器的每个核，以达到最高的处理性能。并发的要求是将任务切分成独立执行的片段。为了解决并发的需求，引入Thread的概念，如果需要将任务并发，将其放到单独的线程中执行，由系统负责调度。

并发模型主要有两种实现形式：同一个进程下，多个线程共享内存，由程序对读写做同步控制（锁机制）；多个进程通过进程间通信或者内存映射实现数据同步。在分布式系统架构中，多个进程之间通过协作完成同一任务，处理这种编程有两种模型：

1. Actor模型，主角是Actor，类似Worker，Actor彼此之间发送消息，不需要经过中介，消息是异步发送和处理的，每个Actor都有一个Mailbox，用于收发数据，如下图所示：



1. CSP模型，提供多个进程公用的“管道Channel”，这个Channel中存放多个任务，如下图所示：



这两种模型典型的实现是Scala的actors和Go中的goroutines，这两种独立的进程都是通过消息传递进行通信，主要区别在于在CSP消息交换是同步的，任务被推送到Channel就立即执行，如果任务执行的线程忙，则发送者暂时无法推送新任务。Actor模型是完全解耦的，进程是异步的，消息传递给Actor后不一定马上执行，可以在任意时间将消息发送给任何未经证实的接受者，Actor使用Mailbox发送及接收消息。

对于Actor模型，具有以下优势：

* 基于事件模型驱动，Actor之间的通信是异步的，即使Actor在发送消息后也无需阻塞或者等待就能够处理其他事件。
* 强隔离型，Actor中的方法不能由外部直接调用，所有的一切都是通过消息传递进行的，从而避免Actor之间的数据共享，想要观察到另一个Actor的状态变换只能通过消息传递进行询问
* 位置透明，无论Actor地址是本地还是远程对代码来说都是一样的
* 轻量性，Actor是轻量的计算单机，只需少量内存就能达到高并发

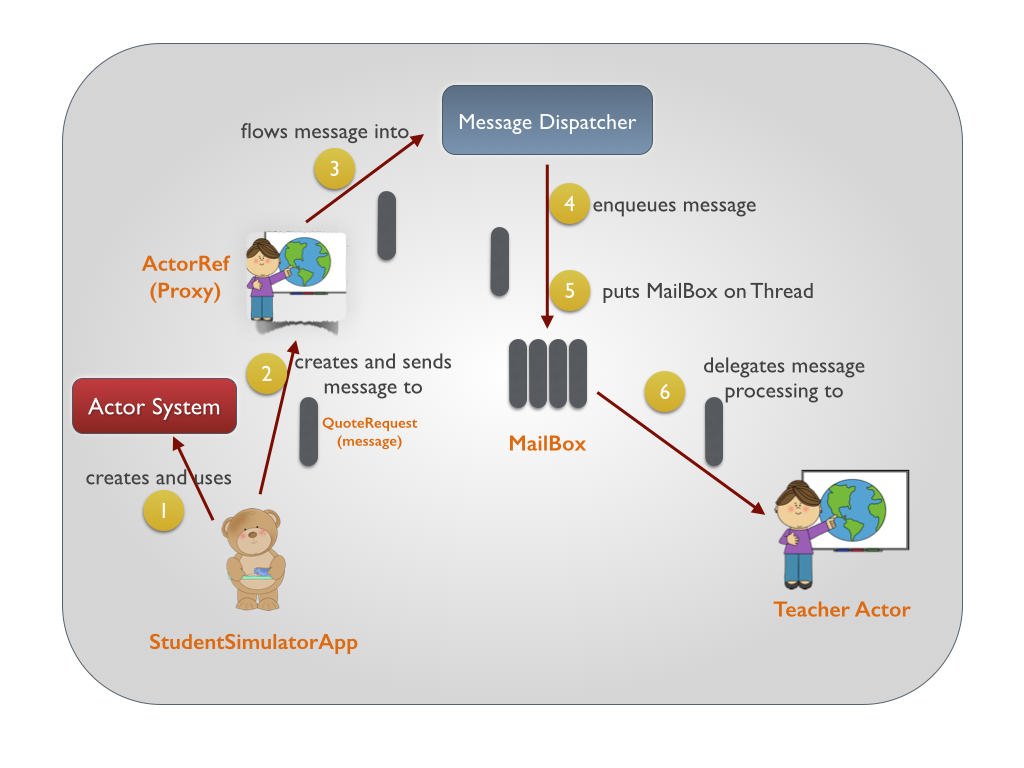
# Actor模型详解

Actor模型是一个概念模型，可以当做通用的并行计算原语，用于处理并发计算，其定义了一系列系统组件应该如何action和交互的通用规则。Actor可以作为一个容器，来存储状态、行为、Mailbox及Actory与Supervisor策略。Actor之间相互隔离，并不互信共享内存，也就是说Actor维护一个私有状态，并且这个状态不可能被另一个Actor所改变，Actor之间通过MailBox来互通有无。

* 状态（State），指Actor对象的变量信息，状态由Actor自己管理，避免并发环境下的锁和内存原子性问题
* 行为（Behavior），行为指定的Actor中的计算逻辑，通过Actor接受到的消息来改变Actor状态
* 消息（Mailbox），邮箱是Actor和Actor之间的通信桥梁，Mailbox内部通过FIFO消息队列来存储发送方Actor消息，接收方Actor从邮箱队列中获取消息。Actor的基础就是消息传递。

## 1.1 示例

下图是模拟Student与Teacher之间的邮箱通信的Actor模型：



流程如下所示：

1. StudentSimulatorApp，创建ActorSystem，ActorSystem作为顶层的Actor（类似于JAVA底层的Object），可以创建和停止Actors，甚至可以关闭整个Actor环境。通过ActorSystem的actof方法创建Actor，实际上就是在ActorSystem下创建一个子Actor。ActorSystem的创建代码如下：

*val system = ActorSystem("UniversityMessageSystem")*

1. 通过ActorSystem创建TeacherActor代理（ActorRef）,创建代码如下：

*val teacherActorRef:ActorRef = system.actorOf(Props[TeacherActor])*

ActorSystem通过actorOf创建Actor，返回类型为ActorRef的对象。ActorRef为Actor的代理，使得客户端不直接与Actor对话。这种Actor模型避免TeacherActor自定义/私有方法或变量被直接访问，所有最好将消息发送给ActorRef，由它去传递给目标Actor。

1. 发送QuotaRequest消息到代理中，通过调用”!”方法将QuoteRequest消息发送给ActorRef，代码示例如下：

*techerActorRef!QuoteRequest*

4）ActorRef将消息处理能力委派给Dispatcher，实际上创建ActorSystem和ActorRef时，Dispatcher和MailBox已经被创建。

5) Dispatcher从ActorRef中获取消息并传递给MailBox，Dispatcher封装一个线程池，之后在线程池中执行MailBox

*protected[akka] override def registerForExecution(mbox: Mailbox, ...): Boolean = {*

*...*

*try {*

*executorService execute mbox*

*...*

*}*

6）当ActorRef发送消息到调用目标的Actor的receiver方法时，MailBox中的run方法被执行，接着从消息队列中取出一条消息并传递给Actor处理

*def receive = {*

*case QuoteRequest => {*

*import util.Random*

*//从list中随机选出一条消息作为回应（这里只print并没回应学生的请求）*

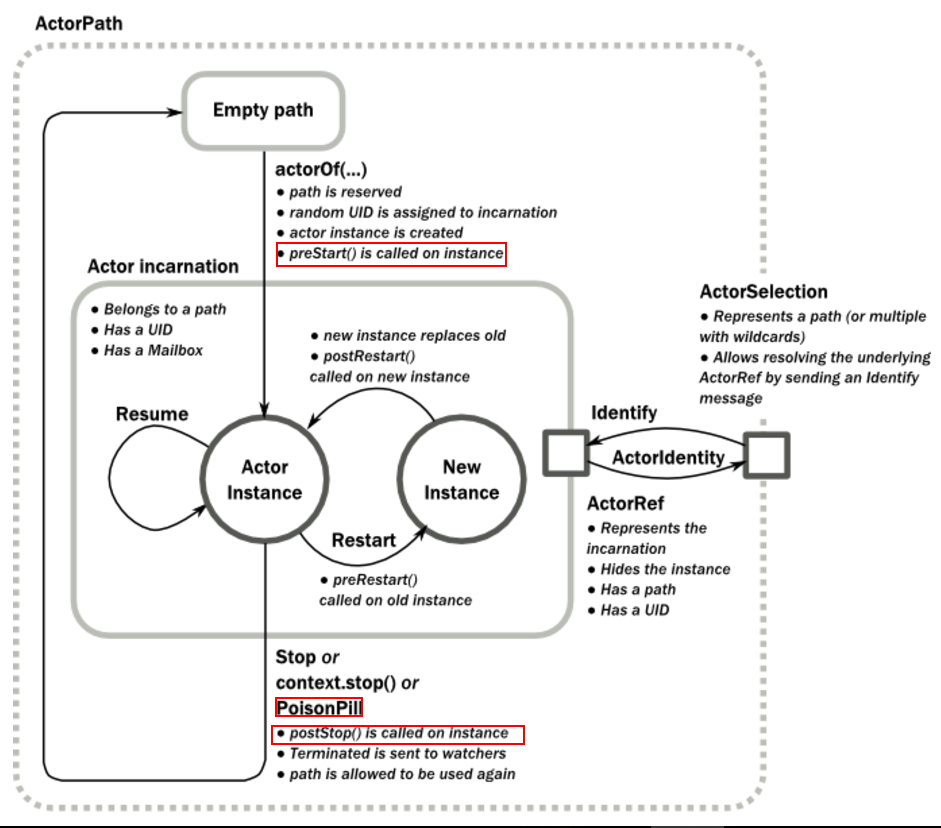
*val quoteResponse=QuoteResponse(quotes(Random.nextInt(quotes.size)))*

*println (quoteResponse)*

*}*

## 概念

1. ActorSystem，ActorSystem是Akka的一个入口，可以看做是Actor的系统工厂或者管理者，管理Actor的生命周期、停止Actor等
2. Actor，Actor中的典型方法对preStart和postStop，在启动和停止时被调用，生命周期如下：



1. ActorRef，ActorRef为Actor的代理，使得客户端不直接与Actor交互
2. MailBox，每个Actor都有一个MailBox，内部采用FIFO队列来存储消息。
3. Dispatcher，从ActorRef中获取消息并传递给MailBox，Dispatcher封装了一个线程池，在线程池中执行MailBox

https://www.cnblogs.com/MOBIN/p/7236893.html

https://en.wikipedia.org/wiki/Actor\_model

http://blog.csdn.net/zhaodedong/article/details/73441303

Actor视频：https://www.jianshu.com/p/449850aa8e82

Actor生命周期：https://zhangyi.gitbooks.io/akka-in-action/content/actor.html