# Demo地址

<https://github.com/JianFengY/AkkaDemo>

# 简介

Akka是一个由Scala编写的,能兼容Scala和JAVA的,用于编写高可用和高伸缩性的Actor模型框架。它采用了基于事件驱动的并发处理模式，把并发操作的各种复杂的东西都统一的做了封装，因此大大的简化了我们在应用系统中开发并发处理的过程，使我们主要关心的是业务逻辑的实现。

特性：

1. 垂直扩展（并发）
2. 水平扩展（远程调用）
3. 高容错

## Actor

Actor是应用创建中最小的单元，它是封装了状态和行为的对象。Actor之间并不直接通信，他们的唯一通讯方式就是交换消息，理论上来讲，每一个Actor都拥有属于自己的轻量级线程，保护它不会被系统中的其他部分影响。因此，我们在编写Actor时,就不用担心并发的问题。

## Java Actor API



(1) 这个Actor继承了AbstractActor。这是一个Java 8特有的API，利用了lambda表达式另外还有一个较老版本的UntypedActor。

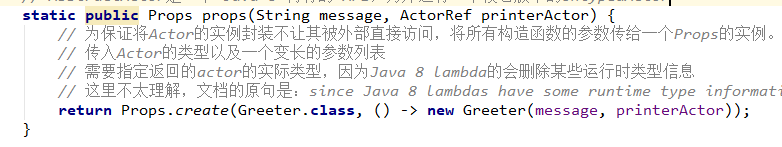
(2) 在createReceive()方法中设置初始行为，返回receiveBuilder()定义接收到消息作出响应。

(3) receiveBuilder() 提供了一些match()方法为它可以接受的所有消息提供模式匹配并且可以通过函数做出响应。

## Props

为了保证能够将Actor的实例封装起来，不让其被外部直接访问，我们将所有构造函数的参数传给一个Props的实例。Props允许我们传入Actor的类型以及一个变长的参数列表。

一种常见模式是在Actor中编写静态方法props描述如何构造Actor。例如我示例代码中的：



# 总结

文档：<https://doc.akka.io/docs/akka/current/>

按照我的理解，Akka就是一个工具包，它在JVM上实现了Actor模型，使用它可以方便地搭建分布式并发应用程序。Actor模型就类似一个对象，而它特别的地方就是不能直接调用或修改，只能通过异步的接收或发送消息进行通信。

书本上的一个通俗的例子：一个餐馆有三个Actor，客人、服务员和厨师，首先，客人向服务员下单，服务员把订单传递给厨师，厨师有空闲的时候就会获取这条消息（订单），然后开始制作菜品（处理消息），完成后，厨师就会告知服务员（发送消息），等待服务员获取，此时，厨师可以处理其他消息（订单），服务员空闲的时候就会获取这条消息并把菜送到客人的餐桌（发送消息），之后客人品尝（处理消息）。这其中每个任务都可以并发进行，因为一个 Actor 每次只处理一条消息，所以可以在Actor内部安全地保存状态。这样就不用考虑多线程的安全和锁的问题。

以上内容主要来自官方文档以及《实战Java高并发程序设计》《Akka入门与实践》这两本书。由于时间关系，并没有完全看完这些资料，只是做了简单的阅读和示例代码的编写与整理。