AKKA

**Actor Systems**

Actors相当于一群人，ActorSystem把子任务分配给他们去执行。

ActorSystem是重量级的，里面会创建很多线程，因此一个程序最好只创建一个。

## Hierarchical Structure

Actors是天然分层的。任务切割成多个子任务，需要多个Actor去处理，于是便创建子Actors,并监督。每一个Actor有且仅有一个supervisor，它就是创建它的父Actor。

actor systems 最精髓的特征是：任务被分解到足够下并分配。为了做到这些，任务需要被结构化，并且子actor知道如何去处理，发生异常如何处理。如果actor没有定义这些，它会抛出一个响应的异常给supervisor（父actor）。

一个设计难题是Supervisor需要监督哪些子actor。没有最好的解决方案，但有一些指导：

1 如果一个actor manager通过传递子任务到另外一个actor，那么这个manager应该监督这个child。

2 如果一个actor携带非常重要的数据。最好的办法为每一个request创建一个子actor，为收集响应简化状态管理。

3 如果一个actor的执行依赖另外一个actor，它应该监控这个actor的生命周期，并处理这个actor终止的通知。这个和监控actor不一样

## Configuration Container

 actor system作为整个系统的协调者。不同配置的actor system可能并存在一个jvm中。Akka没有一个全局的配置。它们之间的透明通信取决于阻塞？？

## Actor Best Practices

1 Actor之间工作应该相互独立。不应该因为外面的资源实体而阻塞。

2 在actors之间不要传递可变的对象

3 actor是行为和状态的容器。

## Blocking Needs Careful Management

**What is an Actor?**

1 Actor是一个容器，包含： [State](http://doc.akka.io/docs/akka/2.4/general/actors.html#state), [Behavior](http://doc.akka.io/docs/akka/2.4/general/actors.html#behavior), a [Mailbox](http://doc.akka.io/docs/akka/2.4/general/actors.html#mailbox), [Child Actors](http://doc.akka.io/docs/akka/2.4/general/actors.html#child-actors) and a [Supervisor Strategy](http://doc.akka.io/docs/akka/2.4/general/actors.html#supervisor-strategy)。所有这些都密封在Actor Reference中。

2 Actor有生命周期，并不自动销毁，需要程序员自己控制。

## Actor Reference

## State

# Supervision and Monitoring

要把actors看成一个有层次结构的整体。所有的动作都会同其子孙actor一致。

1 resume：会保留积累的内部状态

2 restart：会清除积累的内部状态。preRestart hook缺省会先终止所有的下属，然后再重启。但可以重写。

3 stop：

4 Escalate：

这种方式看起来不太灵活。例如：希望不同的下属应用不同的策略。

## The Top-Level Supervisors

至少有3个Actor：

### /user: The Guardian Actor

1 是所有用户创建actor的父actor。使用system.actorOf()来创建子actor

### /system: The System Guardian

这个是为了实现正确的关闭顺序。

### /: The Root Guardian

## What Restarting Means

Restart是生成一个新的，并替换旧的；对它所有的子actor有影响。

1 suspend 哪个actor，并迭代suspend所有的子actor

2 调用old actor的preRestart方法：其缺省实现是向所有的子actor发送终止消息，并调用postStop

3 等待所有的子actor终止

4 重新建一个actor

5 调用postRestart，其实现是调用preRestart

6 重启子actor

7 resume the actor

## What Lifecycle Monitoring Means

# Actor References, Paths and Addresses

## What is an Actor Reference?

Actor Reference是ActorRef的子类，代表一个Actor，并发送消息。

1 Actor 有self 字段： 获得自己标准的引用；默认的，被sender包含随着消息发过去

2 sender 方法：能获取发送者的引用

Actor Reference 有好几种类型，根据Actor System的配置：

1 Purely local actor： 不支持网络

2 Local actor reference

3 subtype of local actor references ： 用于routers

4 Remote actor references

5 several special types of actor references

## What is an Actor Path?

# Actors

[Actor Model](http://en.wikipedia.org/wiki/Actor_model) 提供了一个高并发、分布式的系统。解放了程序员在写程序时要处理锁和线程管理。

### Creating Actors with Props

两种创建方式：第一个参数是Props，第二个参数是个名字（如果重复，会抛出异常）

1 使用ActorSystem.actOf

2 使用ActorContext.actOf

ActorRef 是不可变的、可序列化的

ActorRef 是异步创建的。

### Dependency Injection

当Actor参数依赖DI框架注入的时候，必须使用

### The Inbox

1 本质是一个有序队列。 这样一个actor发送的消息按照时间先后进行处理。

里面有一个内置的傀儡actor在起作用。

2 用途：

a 接听多个消息

send和receive（是阻塞函数）

b 监控其他actor的生命周期

watch

## UntypedActor API

1 消息处理函数

2 获取自己和发送者的actor和监控策略（监控策略最好定义在Actor里面）

3 上下文：由上下文又能获得很多东西

### Actor Lifecycle

### Lifecycle Monitoring aka DeathWatch

就是调用watch函数监控一个actor；当actor停止的时候，会向它的监督者发送一个Terminated消息。

### Start Hook