



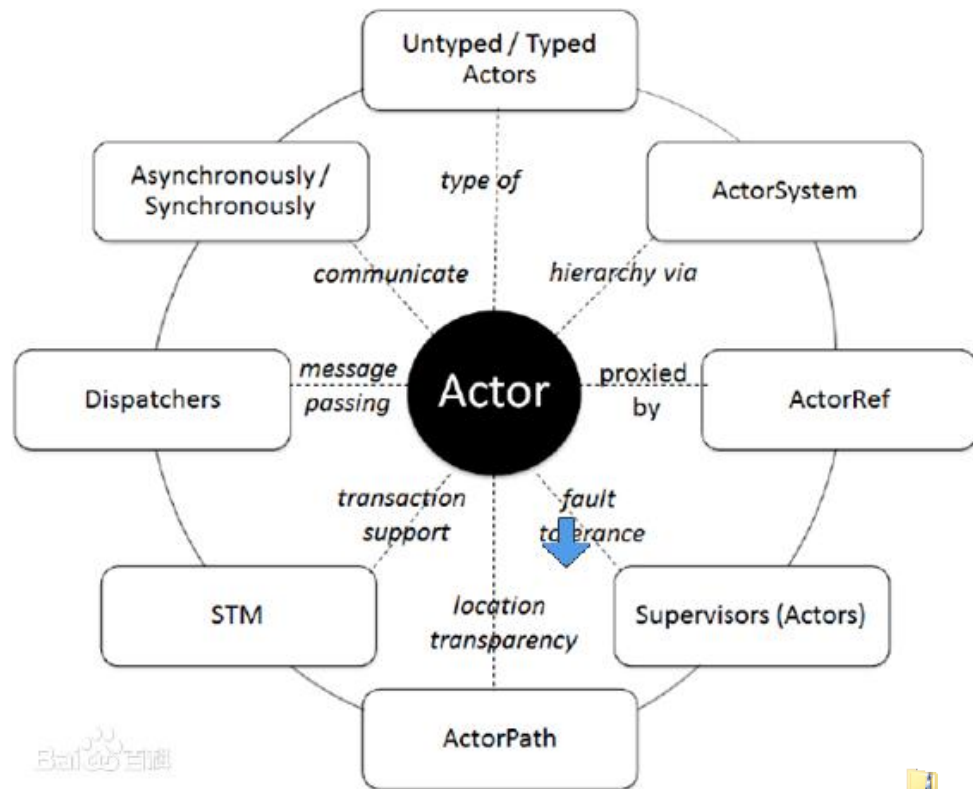
Scala核心编程

-并发编程模型 Akka

尚硅谷研究院

● Akka 介绍

- 1) Akka是JAVA虚拟机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，你可以理解成**Akka是编写并发程序的框架**。
- 2) Akka用Scala语言写成，**同时提供了Scala和JAVA的开发接口**。
- 3) Akka主要解决的问题是：可以轻松的写出高效稳定的并发程序，程序员不再过多的考虑线程、锁和资源竞争等细节。



● Akka 中 Actor 模型

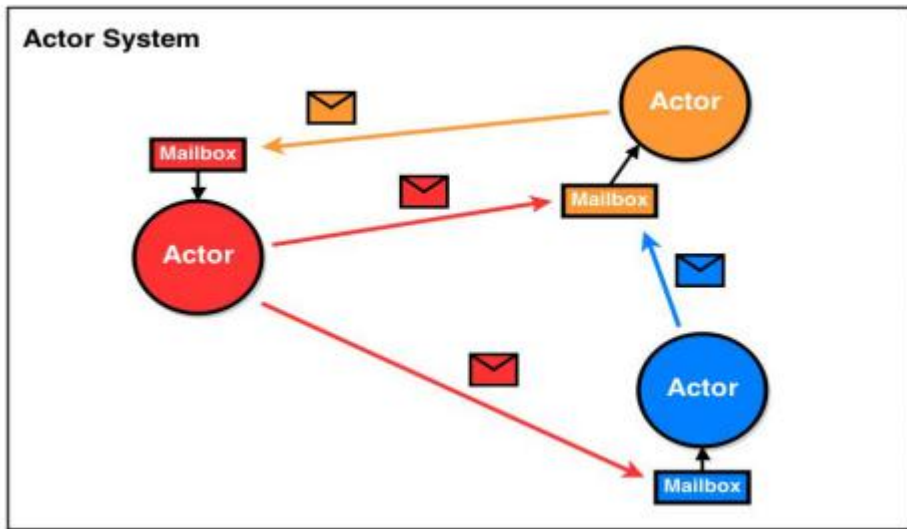
Actor 模型用于解决什么问题

- 1) 处理并发问题关键是要保证共享数据的一致性和正确性，因为程序是多线程时，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。但是当我们对关键代码加入同步条件 `synchronized` 后，实际上大并发就会阻塞在这段代码，对程序效率有很大影响。
- 2) 若是用单线程处理，不会有数据一致性的问题，但是系统的性能又不能保证。
- 3) 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。你可以这里理解：Actor 模型是一种处理并发问题的解决方案，很牛！

经典案例：售票系统

● Akka 中 Actor 模型

Actor模型及其说明



图片22.z

- 1) Akka 处理并发的方法基于 Actor 模型。(示意图)
- 2) 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。
- 3) Actor 模型是作为一个并发模型设计和架构的。Actor 与 Actor 之间**只能通过消息通信**，如图的信封。

● Akka 中 Actor 模型

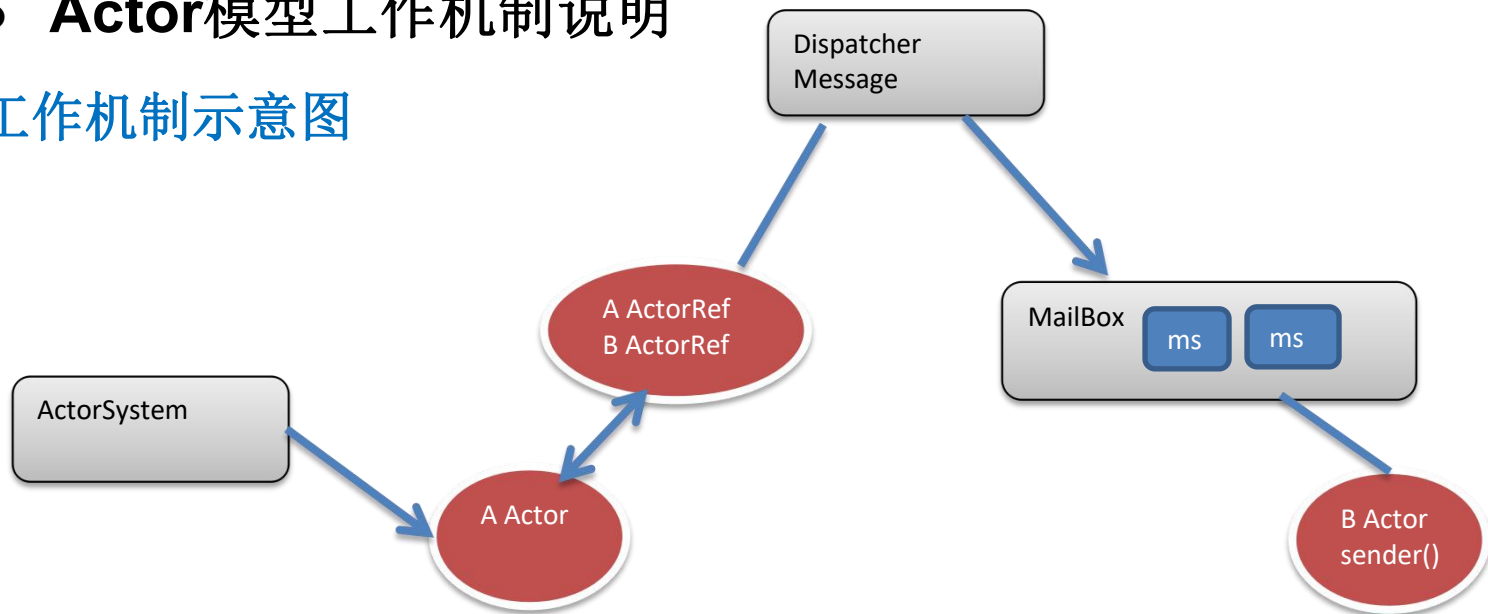
Actor模型及其说明

- 4) Actor 与 Actor 之间只能用消息进行通信，当一个 Actor 给另外一个 Actor 发消息，消息是有顺序的(消息队列)，只需要将消息投寄的相应的邮箱即可。
- 5) 怎么处理消息是由接收消息的Actor决定的，发送消息Actor可以等待回复，也可以异步处理【ajax】
- 6) ActorSystem 的职责是负责创建并管理其创建的 Actor， ActorSystem 是单例的(可以ActorSystem是一个工厂，专门创建Actor)，一个 JVM 进程中有一个即可，而 Acotr 是可以有多个的。
- 7) Actor模型是对并发模型进行了更高的抽象。
- 8) Actor模型是异步、非阻塞、高性能的事件驱动编程模型。[案例: 说明 什么是异步、非阻塞, 最经典的案例就是ajax异步请求处理]
- 9) Actor模型是轻量级事件处理（1GB 内存可容纳百万级别个 Actor），因此处理大并发性能高。



● Actor模型工作机制说明

工作机制示意图



- 1) A Actor 如果想给自己发消息，就通过A ActorRef
- 2) A Actor 想给B Actor 发消息，就需要有(持有)B ActorRef，通过B ActorRef 发消息

后有说明

B Actor 中
receive 方法 {
//1.消息接收和处理
//2. 通过sender() 方法可以得到发送消息的Actor的
ActorRef, 通过这个ActorRef， B Actor 也可以回复消息
}



● Actor模型工作机制说明

Actor模型工作机制说明(对照工作机制示意图理解)

- 1) ActorySystem创建Actor
- 2) ActorRef:可以理解成是Actor的**代理或者引用**。消息是通过ActorRef来发送,而不能通过Actor 发送消息,通过哪个ActorRef 发消息,就表示把该消息发给哪个Actor
- 3) 消息发送到Dispatcher Message (消息分发器),它得到消息后,会将消息进行分发到对应的MailBox。(注: Dispatcher Message 可以理解成是一个线程池, MailBox 可以理解成是消息队列,可以缓冲多个消息,遵守FIFO)
- 4) Actor 可以通过 receive方法来获取消息,然后进行处理。



- **Actor模型工作机制说明**

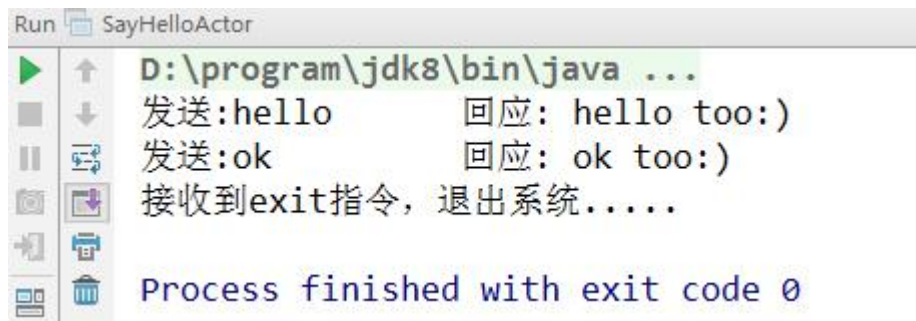
Actor间传递消息机制(对照工作机制示意图理解)

- 1) 每一个消息就是一个Message对象。Message 继承了Runnable， 因为Message就是线程类。
- 2) 从**Actor**模型工作机制看上去很麻烦，但是程序员编程时只需要编写Actor就可以了，其它的交给Actor模型完成即可。
- 3) A Actor要给B Actor 发送消息，那么A Actor 要先拿到(也称为**持有**) B Actor 的 代理对象ActorRef 才能发送消息

● Actor模型快速入门

应用实例需求

- 1) 编写一个Actor, 比如SayHelloActor
- 2) SayHelloActor 可以给自己发送消息 ,如图



```
Run SayHelloActor
D:\program\jdk8\bin\java ...
发送:hello      回应: hello too:)
发送:ok         回应: ok too:)
接收到exit指令, 退出系统.....
Process finished with exit code 0
```

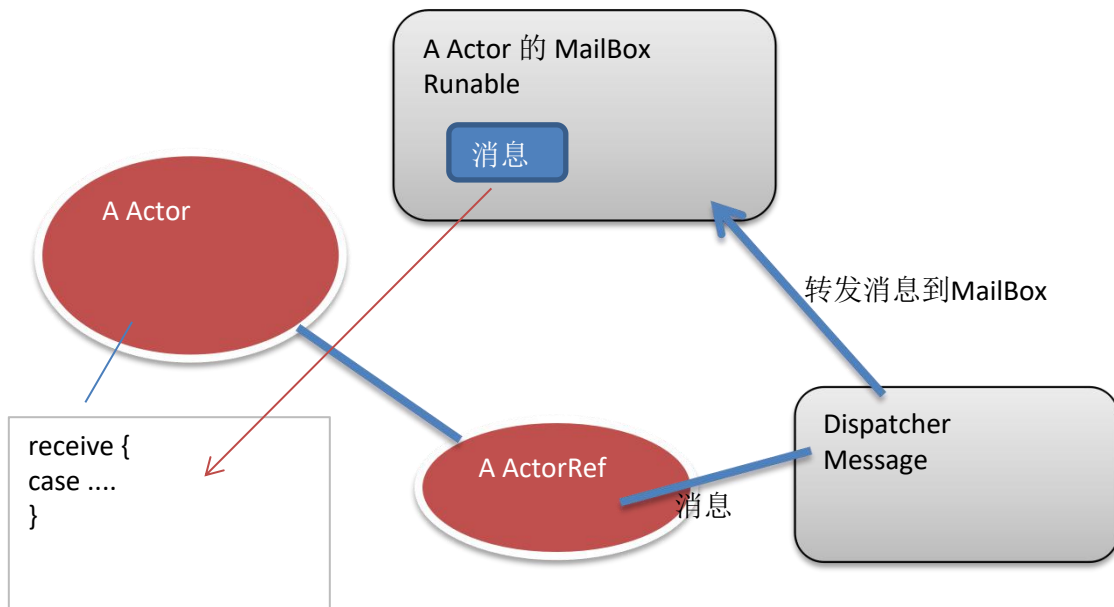


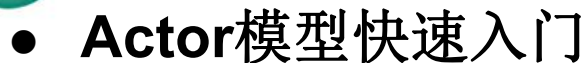
- 3) 要求使用Maven的方式来构建项目,这样可以很好的解决项目开发包的依赖关系。[scala 和 akka]

● Actor模型快速入门

Actor自我通讯机制原理图

➤ 初步理解Actor通讯机制





看老师演示



```
object SayHelloActor {
  private val actoryFactory = ActorSystem("ActoryFactory")
  private val sayHelloActorRef: ActorRef = actoryFactory.actorOf(Props[SayHelloActor], "say")
}
```



● Actor模型快速入门

Actor自我通讯机制原理图

➤ 小结和说明

当程序执行 `aActorRef = actorFactory.actorOf(Props[AActor], "aActor")`，会完成如下任务 [这是非常重要的方法]

- 1) `actorFactory` 是 `ActorSystem("ActorFactory")` 这样创建的。
- 2) 这里的 `Props[AActor]` 会使用反射机制，创建一个 `AActor` 对象，如果是 `actorFactory.actorOf(Props(new AActor(bActorRef)), "aActorRef")` 形式，就是使用 `new` 的方式创建一个 `AActor` 对象，注意 `Props()` 是小括号。
- 3) 会创建一个 `AActor` 对象的代理对象 `aActorRef`，使用 `aActorRef` 才能发送消息
- 4) 会在底层创建 `Dispatcher Message`，是一个线程池，用于分发消息，消息是发送到对应的 `Actor` 的 `MailBox`



- **Actor模型快速入门**

Actor自我通讯机制原理图

➤ 小结和说明

- 5) 会在底层创建AActor 的MailBox 对象，该对象是一个队列，可接收Dispatcher Message 发送的消息
- 6) MailBox 实现了Runnable 接口，是一个线程，一直运行并调用Actor的 receive 方法，因此当Dispatcher 发送消息到MailBox时，Actor 在receive 方法就可以得到信息.
- 7) aActorRef ! "hello", 表示把hello消息发送到A Actor 的mailbox （通过 Dispatcher Message 转发）



- **Actor**模型快速入门

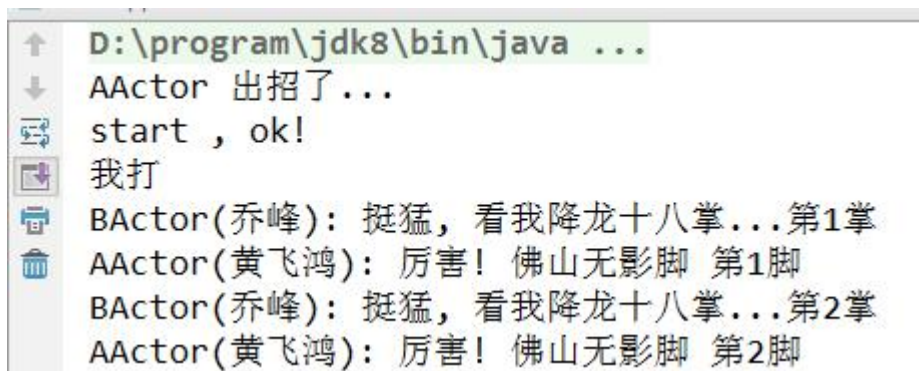
课堂练习

10min完成该项目

● Actor模型应用实例-Actor间通讯

应用实例需求

- 1) 编写2个 Actor , 分别是 AActor 和 BActor
- 2) AActor和BActor之间可以相互发送消息.



```
D:\program\jdk8\bin\java ...  
AActor 出招了...  
start , ok!  
我打  
BActor(乔峰): 挺猛, 看我降龙十八掌...第1掌  
AActor(黄飞鸿): 厉害! 佛山无影脚 第1脚  
BActor(乔峰): 挺猛, 看我降龙十八掌...第2掌  
AActor(黄飞鸿): 厉害! 佛山无影脚 第2脚
```



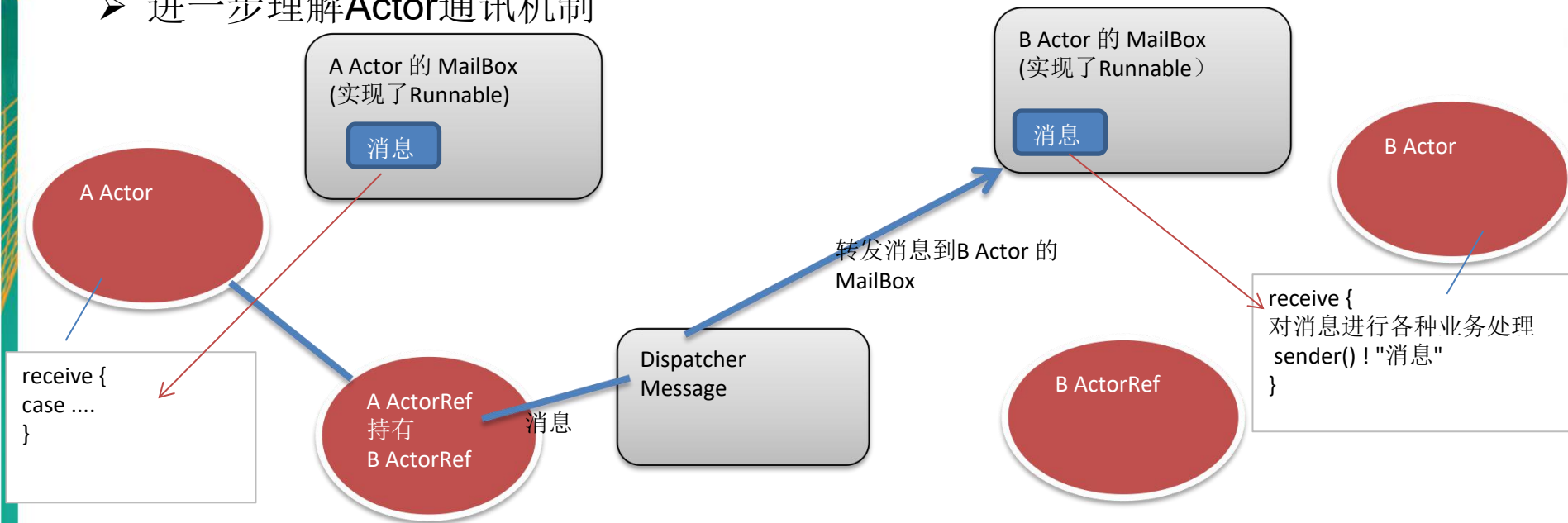
图片3.z

- 3) 加强对Actor传递消息机制的理解。

● Actor模型应用实例-Actor间通讯

两个Actor的通讯机制原理图

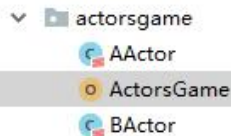
➤ 进一步理解Actor通讯机制



● Actor模型应用实例-Actor间通讯

代码实现

看老师演示



```
class AActor(bActorRef: ActorRef) extends Actor { //AActor.scala
  override def receive: Receive = {
    case "start" => {
      println("AActor(黄飞鸿) 开始游戏了")
      bActorRef ! "我打"}
    case "我打" => {
      println(s"AActor(黄飞鸿) 厉害 看我佛山无影脚~~~~ 第$attact 脚")
      Thread.sleep(1000)
      bActorRef ! "我打"}
  }
}
```

```
class BActor extends Actor { //BActor.scala
  override def receive: Receive = {
    case "我打" => {
      println(s"BActor(乔峰): 挺猛的
      看我降龙十八掌~ 第$attack 掌")
      Thread.sleep(1000)
      sender() ! "我打"}
  }
}
```

```
object ActorsGame { //ActorsGame.scala
  def main(args: Array[String]): Unit = {
    val actorFactory = ActorSystem("actorFactory")
    val bActorRef = actorFactory.actorOf(Props[BActor], "bActor")
    val aActorRef = actorFactory.actorOf(Props(new AActor(bActorRef)), "aActor")
    aActorRef ! "start"}}
}
```





● Actor模型应用实例-Actor间通讯

两个Actor的通讯机制原理图

➤ 小结和说明

- 1) 两个Actor通讯机制和Actor 自身发消息机制基本一样，只是要注意如下
- 2) 如果A Actor 在需要给B Actor 发消息，则需要持有B Actor 的 ActorRef，可以通过创建时，传入 B Actor的 代理对象(ActorRef)
- 3) 当B Actor 在receive 方法中接收到消息，需要回复时，可以通过sender() 获取到发送Actor的 代理对象。

➤ 如何理解Actor 的receive 方法被调用？

- 1) 每个Actor对应MailBox
- 2) MailBox 实现了Runnable 接口，处于运行的状态
- 3) 当有消息到达MailBox,就会去调用Actor的receive方法，将消息推送给receive

● Akka网络编程

看两个实际应用(socket/tcp/ip)

1) QQ,迅雷,百度网盘客户端.

新浪网站,京东商城,淘宝



图片2.z



图片3.z



● Akka网络编程基本介绍

Akka支持面向大并发后端服务程序，网络通信这块是服务端程序重要的一部分。

网络编程有两种：

- 1) **TCP socket编程**，是网络编程的主流。之所以叫Tcp socket编程，是因为底层是基于Tcp/ip协议的. 比如: QQ聊天 [示意图]
- 2) **b/s结构的http编程**，我们使用浏览器去访问服务器时，使用的就是http协议，而http底层依旧是用tcp socket实现的。 比如: 京东商城 【属于 web 开发范畴】

- 网络编程基础知识

网线,网卡,无线网卡

计算机间要相互通讯,必须要求网线,网卡,或者是无线网卡.



图片4.z



- 网络编程基础知识

协议(tcp/ip)

TCP/IP (Transmission Control Protocol/Internet Protocol)的简写,中文译名为传输控制协议/因特网互联协议, 又叫网络通讯协议, 这个协议是Internet最基本的协议、Internet国际互联网的基础, 简单地说, 就是由网络层的IP协议和传输层的TCP协议组成的。

- 网络编程基础知识

OSI与Tcp/ip参考模型 (推荐tcp/ip协议3卷)

应用层 (application)
表示层 (presentation)
会话层 (session)
传输层 (transport)
网络层 (ip)
数据链路层 (link)
物理层 (physical)

Osi模型 (理论)

应用层 (application) smtp, ftp, telnet http
传输层: (transport) 解释数据
网络层: (ip) 定位ip地址和 确定连接路径
链路层: (link) 与硬件驱动 对话

Tcp/ip模型 (现实)



tracert.zip



qq通讯网络底

深入理解:qq间相互通讯的案例



● 网络编程基础知识

ip地址

概述：每个internet上的主机和路由器都有一个ip地址，它包括网络号和主机号，ip地址有ipv4(32位)或者ipv6(128位). 可以通过ipconfig 来查看

```
C:\Users\Administrator>ipconfig
```

Windows IP 配置

无线局域网适配器 无线网络连接 2:

媒体状态 : 媒体已断开
连接特定的 DNS 后缀 :

无线局域网适配器 无线网络连接:

连接特定的 DNS 后缀 :
本地连接 IPv6 地址 : fe80::3c46:ea3c:8703:de88%14
IPv4 地址 : 192.168.1.7
子网掩码 : 255.255.255.0
默认网关 : fe80::1%14
192.168.1.1

- 网络编程基础知识

端口(port)-介绍

我们这里所指的端口不是指物理意义上的端口，而是特指**TCP/IP**协议中的端口，是逻辑意义上的端口。

如果把**IP**地址比作一间房子，端口就是出入这间房子的门。真正的房子只有几个门，但是一个**IP**地址的端口可以有**65535**（即： $256 \times 256 - 1$ ）个之多！端口是通过端口号来标记的。（端口号 **0**: Reserved)



- 网络编程基础知识

端口(port)-分类

- 0号是保留端口.
- 1-1024是固定端口
又叫有名端口,即被某些程序固定使用,一般程序员不使用.
22: SSH远程登录协议 23: telnet使用 21: ftp使用
25: smtp服务使用 80: iis使用 7: echo服务
- 1025-65535是动态端口
这些端口,程序员可以使用.

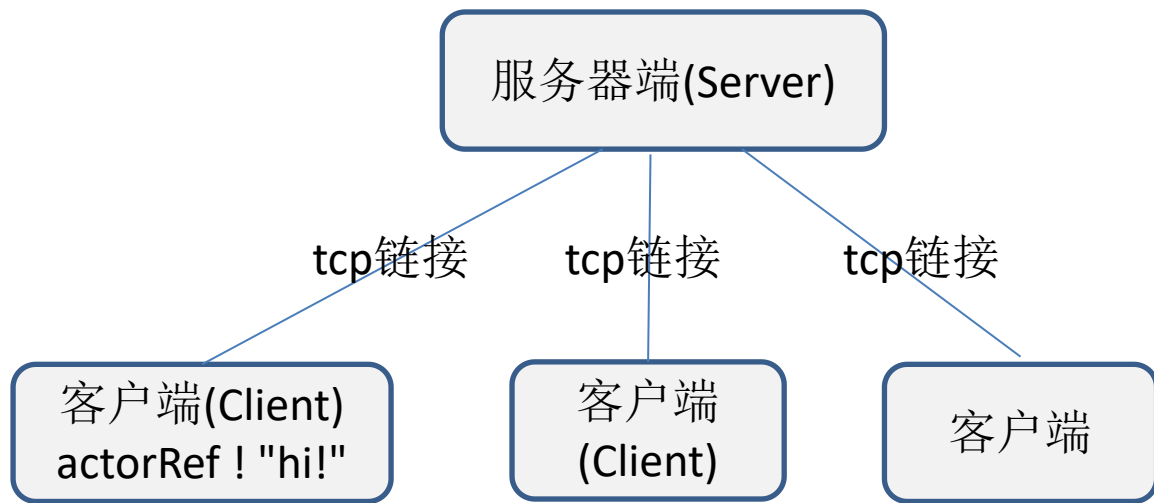
- 网络编程基础知识

- 端口(port)-使用注意

- 1) 在计算机(尤其是做服务器)要尽可能的少开端口[]
- 2) 一个端口只能被一个程序监听()
- 3) 如果使用 `netstat -an` 可以查看本机有哪些端口在监听
- 4) 可以使用 `netstat -anb` 来查看监听端口的pid,在结合任务管理器关闭不安全的端口.

- 网络编程基础知识

为了授课方法，我们将tcp socket编程，简称 socket编程。
下图为socket编程中客户端和服务器的网络分布





● Akka网络编程-小黄鸡客服

需求分析

- 1) 服务端进行监听(9999)
- 2) 客户端可以通过键盘输入，发送咨询问题给小黄鸡客服(服务端)
- 3) 小黄鸡(服务端) 回答客户的问题

界面设计

小黄鸡(服务端)



图片5.z

```
[INFO] [10/30/2018 01:09:04.399] [main] [akka.remote]
启动了在9999端口监听...
客户咨询问题 hello
[WARN] [SECURITY][10/30/2018 01:09:10.846] [client-]
客户咨询问题 大数据学费是多少
客户咨询问题 学校地址
客户咨询问题 what
客户咨询问题 可以学那些技术
```

客户

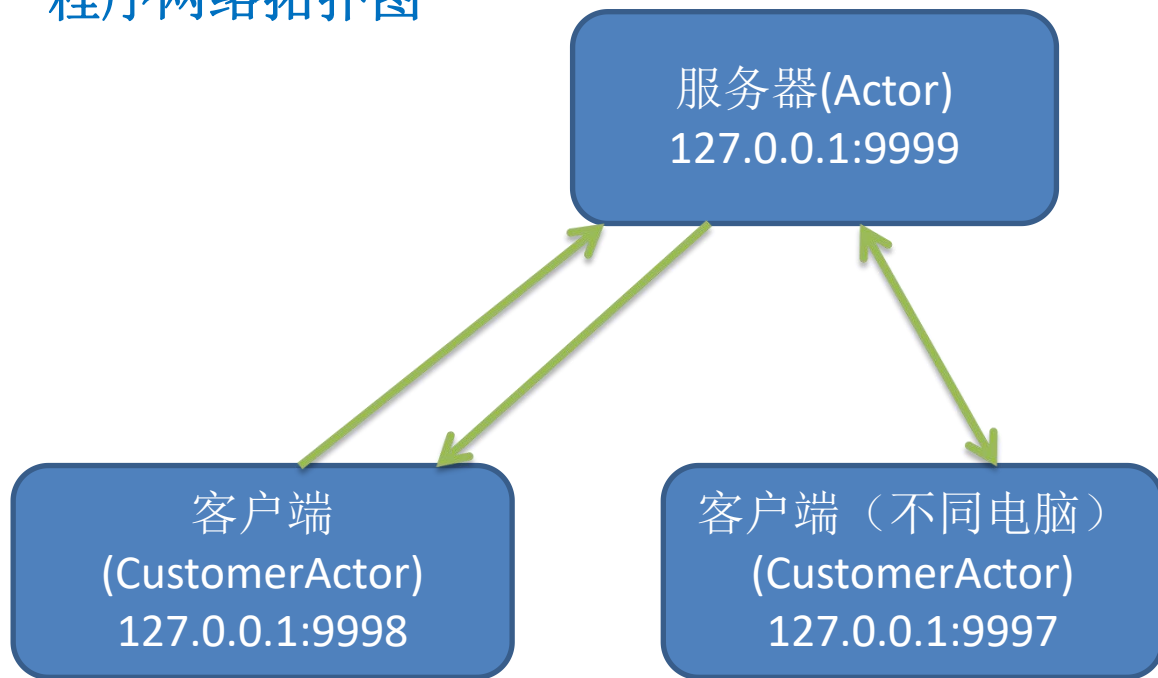


图片6.z

```
[INFO] [10/30/2018 01:09:04.399] [main] [akka.remote]
客户端start...
hello
[WARN] [SECURITY][10/30/2018 01:09:10.782] [client-]
收到小黄鸡客服(server)消息：你说啥子:)
大数据学费是多少
收到小黄鸡客服(server)消息：15000$
学校地址
收到小黄鸡客服(server)消息：北京昌平区xxx路
what
收到小黄鸡客服(server)消息：你说啥子:)|
可以学那些技术
收到小黄鸡客服(server)消息：大数据 前端...
```

- Akka网络编程-小黄鸡客服

程序网络拓扑图





- **Akka**网络编程-小黄鸡客服

程序框架图



程序框架&



● Akka网络编程-小黄鸡客服



功能实现(走代码)

```
class YellowChickenServer extends Actor {
  override def receive: Receive = {
    case "start" => println("服务器在9999端口上监听了....")
    case ClientMessage(mes) => {
      println("客户咨询问题是:" + mes)
      mes match {
        case "大数据学费是多少" => sender() ! ServerMessage("15000RMB")
        case "学校地址" => sender() ! ServerMessage("昌平区宏福大楼xxx路")
        case "可以学哪些技术" => sender() ! ServerMessage("JavaEE 大数据
Python")
        case _ => sender() ! ServerMessage("你说啥子~~")
      }
    }
  }
}

object YellowChickenServerApp extends App {
  val host = "127.0.0.1" //服务端ip地址
  val port = 9999
  //创建config对象,指定协议类型, 监听的ip和端口
  val config = ConfigFactory.parseString(
    s"""
    akka.actor.provider="akka.remote.RemoteActorRefProvider"
    akka.remote.netty.tcp.hostname=$host
    akka.remote.netty.tcp.port=$port
    """).stripMargin)

  private val serverActorSystem = ActorSystem("Server", config)
  private val yellowChickenServerActorRef: ActorRef =
    serverActorSystem.actorOf(Props[YellowChickenServer],
```

```
class CustomerActor(serverHost: String, serverPort: Int) extends Actor {
  var serverActorRefer: ActorSelection = _
  override def preStart(): Unit = {
    this.serverActorRefer =
      context.actorSelection(s"akka.tcp://Server@${serverHost}:${serverPort}")
    println("this.serverActorRefer=" + this.serverActorRefer)
  }
  override def receive: Receive = {
    case "start" => println("客户端启动了!!...")
    case mes:String => {
      println("开始咨询了")
      serverActorRefer ! ClientMessage(mes)
    }
    case ServerMessage(mes) => {
      println("收到小黄鸡咨询老师(Server): " + mes)
    }
  }
}

object CustomerActorApp extends App {
  val (host,port,serverHost,serverPort) = ("127.0.0.1",9990,"127.0.0.1",9990)
  val config = ConfigFactory.parseString(
    s"""
    akka.actor.provider="akka.remote.RemoteActorRefProvider"
    akka.remote.netty.tcp.hostname=$host
    akka.remote.netty.tcp.port=$port
    """).stripMargin)

  val clientActorSystem = ActorSystem("client", config)
  val actorRef: ActorRef = clientActorSystem.actorOf(Props[CustomerActor],
    actorRef ! "start"
  while (true) {
    val mes = StdIn.readLine()
    actorRef ! mes
  }
}
```



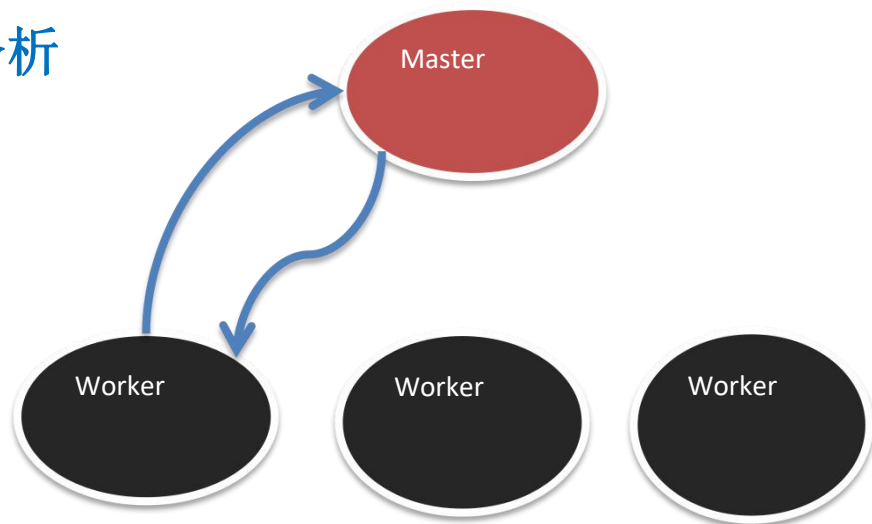

● Spark Master Worker 进程通讯项目

项目意义

- 1) 深入理解Spark的Master和Worker的通讯机制
- 2) 为了方便同学们看Spark的底层源码，命名的方式和源码保持一致.(如： 通讯消息类命名就是一样的)
- 3) 加深对主从服务心跳检测机制(HeartBeat)的理解，方便以后spark源码二次开发。

● Spark Master Worker 进程通讯项目

项目需求分析



- 1) worker注册到Master, Master完成注册, 并回复worker注册成功
- 2) worker定时发送心跳, 并在Master接收到
- 3) Master接收到worker心跳后, 要更新该worker的最近一次发送心跳的时间
- 4) 给Master启动定时任务, 定时检测注册的worker有哪些没有更新心跳,并将其从hashmap中删除
- 5) master worker 进行分布式部署(Linux系统)-> 如何给maven项目打包->上传linux



- **Spark Master Worker 进程通讯项目**

项目界面设计

我们主要是通过应用实例，来剖析Spark 的Master 和 Worker的通讯机制，因此功能比较简洁，设计的界面如下：

待...



● Spark Master Worker 进程通讯项目

实现功能1-Worker完成注册

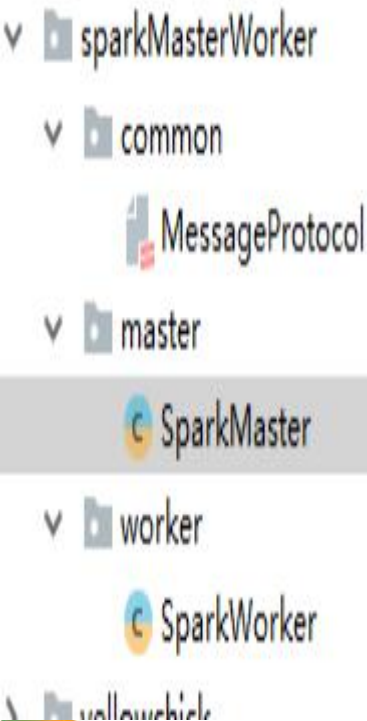
- 功能要求: worker注册到Master, Master完成注册, 并回复worker注册成功
- 思路分析(程序框架图)



程序框架图

• Spark Master Worker 进程通讯项目

➤ 代码实现



```
class SparkMaster extends Actor { //SparkMaster.scala
  val workers = collection.mutable.HashMap[String, WorkerInfo]()
  override def receive: Receive = {
    case "start" => println("master服务, 启动并开始监听端口....")
```

```
    case RegisterWorkerInfo(workerId, cpu, ram) => {
      if (!workers.contains(workerId)) {
        println(workerId + " 注册ok.... ")
        val workerInfo = new WorkerInfo(workerId, cpu, ram)
        workers += ((workerId, workerInfo))
        sender() ! RegisteredWorkerInfo
      }
    }
  }
}
```

```
object SparkMaster {
  def main(args: Array[String]): Unit = {
    val config = ConfigFactory.parseString(
      s"""
        |akka.actor.provider="akka.remote.RemoteActorRefProvider"
        |akka.remote.netty.tcp.hostname=127.0.0.1
        |akka.remote.netty.tcp.port=10001
        |""".stripMargin)
    val actorSystem = ActorSystem("sparkMaster", config)
    val masterActorRef = actorSystem.actorOf(Props[SparkMaster], "master-01")
    masterActorRef ! "start"
  }
}
```

```
// worker注册信息 //MessageProtocol.scala
case class RegisterWorkerInfo(id: String, cpu: Int, ram: Int)
// 这个是WorkerInfo, 保存在master的hashmap中
class WorkerInfo(val id: String, val cpu: Int, val ram: Int)
case object RegisteredWorkerInfo
```





• Spark Mas

➤ 代码实现

```
class SparkWorker(masterUrl: String) extends Actor {  
  var masterProxy: ActorSelection = _  
  val workerId = UUID.randomUUID().toString  
  override def preStart(): Unit = {  
    masterProxy = context.actorSelection(masterUrl)  
  }  
  override def receive: Receive = {  
    case "start" => { // 自己已就绪  
      println(workerId + " 向master发出注册信息... ")  
      masterProxy ! RegisterWorkerInfo(workerId, 1, 64 * 1024) //  
    }  
    case RegisteredWorkerInfo => {  
      println(workerId + " 向master注册成功了... ")  
    }  
  }  
}
```

```
object SparkWorker {  
  def main(args: Array[String]): Unit = {  
    val host = "127.0.0.1"  
    val port = 10002  
    val masterURL = "akka.tcp://sparkMaster@127.0.0.1:10001/user/master-01"  
    val workerName = "worker-01"  
    val config = ConfigFactory.parseString(  
      s"""  
        |akka.actor.provider="akka.remote.RemoteActorRefProvider"  
        |akka.remote.netty.tcp.hostname=127.0.0.1  
        |akka.remote.netty.tcp.port=10002  
        """).stripMargin)  
    val actorSystem = ActorSystem("sparkWorker", config)  
    val workerActorRef = actorSystem.actorOf(Props(new SparkWorker(masterURL)), workerName)  
    workerActorRef ! "start"  
  }  
}
```



● Spark Master Worker 进程通讯项目

实现功能2-Worker定时发送心跳

- 功能要求: worker定时发送心跳给Master, Master能够接收到,并更新worker上一次心跳时间
- 思路分析(程序框架图)



程序框架图



● Spark Master Worker 进程通讯项目

实现功能2-Worker定时发送心跳

➤ 代码实现

```
import scala.concurrent.duration._
case RegisteredWorkerInfo => { //SparkWorker.scala
  println(workerId + " 向master注册成功了... ")
  println(workerId + " 准备开始定时发送心跳消息给master... ")
  import context.dispatcher
  context.system.scheduler.schedule(0 millis, 3000 millis, self, SendHeartBeat)
}
case SendHeartBeat => {
  // 向master发送心跳了
  println(s"----- $workerId 发送心跳-----")
  masterProxy ! HearBeat(workerId)
}
```

```
case HearBeat(workerId) => { //SparkMaster.scala
  val workerInfo = workers(workerId)
  workerInfo.lastHeartBeatTime = System.currentTimeMillis()
  println(s"master: ${workerId} 更新了心跳时间...")
}
```

```
//MessageProtocol.scala
case object SendHeartBeat
case class HearBeat(id: String)
class WorkerInfo(val id: String, val
  cpu: Int, val ram: Int){
  // 新增
  var lastHeartBeatTime: Long = _
}
```




- **Spark Master Worker 进程通讯项目**

实现功能3-Master启动定时任务，定时检测注册的worker

- 功能要求：Master启动定时任务，定时检测注册的worker有哪些没有更新心跳，已经**超时的**worker，将其从hashmap中**删除**掉
- 思路分析(程序框架图)



● Spark Master Worker 进程通讯项目

实现功能3-Master启动定时任务，定时检测注册的worker

➤ 代码实现

```
case "start" => { //SparkMaster.scala
    println("master服务，启动并开始监听端口....")
    self ! StartTimeoutWorker
}
// 开启定时器，每隔一定时间检测是否有worker的心跳超时
case StartTimeoutWorker => {
    import context.dispatcher // 使用调度器时候必须导入dispatcher
    context.system.scheduler.schedule(0 millis, 9000 millis, self, RemoveTimeoutWorker)
}
case RemoveTimeoutWorker => {
    val workerInfos = workers.values
    val currentTime = System.currentTimeMillis()
    // 过滤心跳超时的worker
    workerInfos
        .filter(workerInfo => currentTime - workerInfo.lastHeartBeatTime > 6000)
        .foreach(workerInfo => workers.remove(workerInfo.id))
    println(s"-----还剩 ${workers.size} 存活的Worker-----")
}
```



sparkMasterWorker3.0.zip

```
//MessageProtocol.scala
//master给自己发送一个触发检查超时worker的信息
case object StartTimeoutWorker
// master给自己发消息，检测worker,对于心跳超时的。
case object RemoveTimeoutWorker
```

● Spark Master Worker 进程通讯项目

实现功能4-Master,Worker的启动参数运行时指定

- 功能要求: Master,Worker的启动参数运行时指定, 而不是固定写在程序中的。
- 代码实现:

//SparkMaster.scala

```
def main(args: Array[String]): Unit = {  
  // 检验参数  
  if(args.length != 3) {  
    println("请输入参数: host port masterName")  
    sys.exit() // 退出程序  
  }  
  val host = args(0)  
  val port = args(1)  
  val masterName = args(2)  
}
```



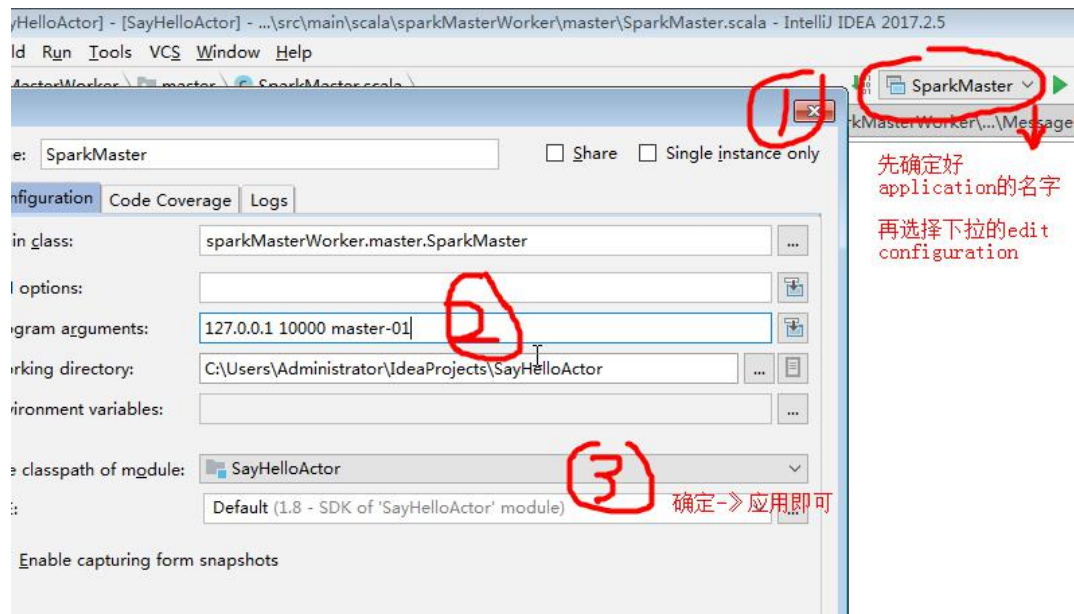
//SparkWorker.scala

```
def main(args: Array[String]): Unit = {  
  if(args.length != 4) {  
    println("请输入参数: host port workerName masterURL")  
    sys.exit() // 退出程序  
  }  
  //定义连接master相关变量, 后面改成参数输入(这样更加灵活)  
  val host = args(0) //"127.0.0.1"  
  val port = args(1) //10002  
  // "akka.tcp://sparkMaster@127.0.0.1:10001/user/master-01"  
  val masterURL = args(2)  
  val workerName = args(3) //"worker-01"  
}
```

● Spark Master Worker 进程通讯项目

实现功能4-Master,Worker的启动参数运行时指定

➤ 运行项目



图片2.z

说明：再点击确认->应用即可保存设置，如果需要运行第二个worker服务，则需要修改参数，再运行。



谢谢！ 欢迎收看！