数据结构：

1、客户端，Proxy，UnicastRef，LiveRef，ObjId

2、客户端，Remote，UnicastServerRef，Target

3、通信对象：ServerSocket，TcpEndPoint，TCPTransport

RemoteObject：远程对象的Object对象行为，持有RemoteRef

RemoteRef：远程对象的句柄，持有LiveRef

LiveRef：看样子是持有通信信息的对象（ObjId，TcpEndPoint）

Proxy：一个实现了远程服务接口并持有远程对象句柄的代理对象。

TcpEndPoint：

a1：一个终端的抽象，地址、端口、client和server的socket创建工厂构成这个终端。

TCPTransport：

a1：通信抽象，与终端一对一对应（虽然维持了一个TcpEndPoint列表，但只有最后一个是使用的，列表由TcpEndPoint自己维护（猜想是修正终端状态））。

a2：在服务端，持有一个TcpEndPoint对TCPChannel的映射表，因为服务端有可能跟多个终端通信。这里是很有意思的一点，TCPChannel持有TcpConnection的一个列表，TcpConnection持有真正的socket，为什么socket可以重用呢？因为TCPChannel持有的所有连接都是同一个终端的，socket自然可以重用。

a3：该对象更像是一个本地终端与众多远程终端通信的管理类，endpoint类有两种创建方式，一种是直接构造（transport对象为null），一种是getLocalEndpoint方法（持有transport对象）。前一种方式是服务端模拟客户端终端以及客户端模拟服务端终端用的，后一种则是服务端发布服务创建所需终端所用。前一种方式有两点需要注意，第一：服务端模拟客户端终端一般都在与客户端通信过程中，此时双方的通信由服务端对应终端的transport对象管理，模拟的客户端终端自然不需要transport对象；第二：客户端模拟服务端终端时，由于客户端获取服务必然需要拿到一个服务端注册表对象（默认0端口终端），也就是说0端口的终端必然存在，所以客户端与该服务端的所有通信交由0端口的transport对象管理（方便统一管理？或者觉得客户端与服务端通信构造的终端不多？）

TCPConnection：

a1：不一定要持有socket，直接持有输入输入流in，out也能跟客户端通信。

Q:

1. 注册表创建为什么不会hang住监听请求。

A:

a1:目前看debug是registry创建的线程是守护线程，当主线程（main方法）执行完后，又没有创建其他的用户线程（非守护线程）时守护线程会自动退出执行。

a2：原以为创建远程服务对象使用的不是守护线程，但看debug都是守护线程，但额外创建了一个DestroyJavaVM非守护线程，具体作用不明（猜想是分布式垃圾回收线程）。

2、假如有两个服务在相同的端口监听，是创建多个serverSoket？还是怎样

A:

a1:目前估计由TcpEndPoint.localEndPoints队列控制

a2：同key（地址，端口，serverSocketFactory， clientSocketFactory全部相同）使用同一个tcpTransport对象，该对象缓存一个serverSocket对象，只创建一次（但假如地址端口相同但两个工厂不同呢？）

1. exported到底做了什么？TcpEndPoint等对象的作用是什么？一个请求服务的请求到达服务端怎么找到服务？一个调用请求到达服务端怎么找到对应的服务对象并进行调用？

A：

a1:要想知道exported做了什么首先需要知道rmi一次的远程调用的执行过程。rmi的所有远程服务对象在导出的时候都会组装一个Target对象并放入ObjectTable缓存，在ObjectTable中持有两条队列，分别是Map<ObjectEndPoint,Target>以及Map<WeakIml,Target>，这也意味着有两种方式可以获取Target，分别是通过key值ObjectEndPoint（比较的是ObjId以及Transport对象，终端EndPoint以及Transport其实相当于一一对应的，所以客户端请求到服务端后完全可以重新组装一个key获得Target），以及通过key值WeakIml（比较的是内部引用的远程对象）。Target有几个比较重要的属性，ObjId，持有真正远程服务对象的weakIml，要传输到客户端的stub（Proxy代理对象），dispatcher（调度者，也就是unicastServerRef，该类不止定义了exported行为，同时也定义了远程对象响应方法调用的行为）。

a2：本来socketServer就是在Transport的一个属性，一个socket accept的时候肯定能拿到对应的Transport对象，也就拿到了对应的Endpoint对象，然后从socket输入流中读出相应的参数（各种tcp、http协议参数，objid，方法hash值，参数列表等等），然后根据objId跟Transport对象可组装ObjectEndPoint对象（Target对应的key值），从ObjectTable中获取Target，然后获取调度者dispatcher，真正的远程服务对象，调度者调用dispatch方法（也就是其定义的远程服务对象相应方法调用的行为），并传入真正的远程服务对象作为参数（看代码似乎还是组装了一个RemoteCall对象作为参数传入,作用尚不明确），然后就是一般的方法调用（骨架调用--registry或者直接调用--远程服务对象），并把结果或者异常写入输出流。至此，一次远程服务调用结束。

4、tcp？http？

5、LiveRef以及TcpEndPoint等对象不能序列化，怎么传输到客户端？

A：

a1：客户端请求注册表对象的时候其实没有向服务端发命令，只是简单的用给定的端口地址以及工厂对象在本地构建一个Registry的代理对象，因为通信只要知道端口地址就可以了。。。虽然如此，但服务端还是需要创建注册表对象用于接收处理客户端的服务请求。

5.1、同理，远程对象的动态代理同样持有LiveRef跟TCPEndPoint对象，怎么传输到客户端？

A：

a1：目前看注册表对象导出跟一般的服务对象导出的处理不一样，注册表对象导出创建的是一个registryiml\_stub类的对象。

a2：UnicastRef实现了Externalizable接口，重写了writeExternal方法，把必要的信息写入steam流中（隐藏必要的信息，只保留通信必要的信息），所以可以序列化传输到客户端。

6、服务端线程模型？

A：

a1：同key（TcpEndPoint对象，同端口地址以及工厂）共用一个TCPTransport对象，该对象保持一个serverSocket对象，用一个线程跑serverSocket.accept()，接收到请求后扔给*connectionThreadPool*线程池处理（静态变量，也就是说rmi不同服务都是用这一个线程池处理请求）

1. 通信模块数据结构？

A：

a1：TCPEndPoint跟TCPTransport（目前看是多个终端对应的一个transport对象，两者关系及作用还有待探讨），transport对象持有channelTable（TCPChannel跟终端映射对应），TCPChannel持有TcpConnection队列，而TcpConnetion才持有真正的socket对象。

a2：持有TcpConnection队列是为多线程调用提供支持，客户端调用rmi时首先尝试从TcpChannel请求一个TCPConnection对象，channel对象首先从freeList提取，假如不成功则newConnection（这里面是有讲究的，如果new的connection不需要多路复用，则直接创建。否则会首先检查创建的connection是否超过最大的连接数）。

a3:要清楚，服务端可以在一个终端EndPoint上发布多个服务，这意味着服务器的一个终端完全有可能被客户端的多个终端EndPoint连接。可以看到服务端的EndPoint跟Transport是一一对应的，而Transport持有一个Map<EndPoint,Reference<TcpChannel>>队列，而这个做key值的EndPoint其实是客户端的终端，而做value值的TCPChannel持有一个List<TcpConnection>队列，这个connection队列其实都是跟同一个客户端EndPoint连接的连接。意思是同一个服务终端上发布的服务跟某一个客户端通信持有一个也是唯一一个TCPChannel对象，但TCPConnection对象可以由多个，socket也是多个。

a4：由a2跟a3推断，channel只是一个管理connection的类，甚至可以看成是一个工具类（对一个客户端终端而言）。所以7.1中a1的疑问就可以有部分的解答了。重用channel很容易理解，毕竟一个工具类没有必要重复创建对象，服务端跟客户端通信最直观的对象应该是connection（封装了socket以及一些具体通信信息），所以假如rmi有做socket的多路复用（或者应该叫长连接），那么需要重用的就应该是connection对象。

7.1、如何实现socket or connection重用（甚至channel重用）？

A：

a1：要想解答这个问题，首先要了解真正被重用的对象是什么！现在看来似乎不是socket，不是connection，channel倒是有重用（但其最终使用的还是connection，connection最终使用的还是socket，假如底层的最重要的socket最耗费资源的对象都不重用，那重用上层对象有什么意义？）。

a2：在客户端，调用一个远程方法其实就是跟远程主机的一次通信。流程是：首先从channel中newConnection（如果freeList中有空闲connection直接拿出来用），用完之后free（如果可重用，重新放回freeList中），如此实现connection重用socket重用（但socket是双方的，假如服务端单独关闭socket？）。

a3:如果重用socket，这里有一个问题，BIO不同NIO，BIO监听socket通道是否有消息到达是会阻塞的（这也就是为什么说BIO是以socket为线程单位，一个socket一个线程的原因）。这里可以分客户端服务器两方面讨论。首先客户端，因为客户端发信息读信息是在一个方法中实现的，同时也只有向服务器发送信息后才需要监听服务器返回，换句话说客户端是主动发消息方，所以客户端socket是可以重用的。但服务器不同，服务器是被动接收方，需要监听客户端消息到达处理然后返回，这样客户端的socket就需要阻塞一个线程监听该socket上的消息，。

8、要使远程服务对象具备JRMP（java remote method protocol）的能力，需要对远程服务对象进行导出exported，有两种方式。一种是继承UnicastRemoteObject，另一种是使用UnicastRemoteObject的静态方法exported。这两种有什么不同？

A：

a1：无论是继承还是调用方法，最后都会调用到UnicastServerRef的exported方法（此方法定义了远程服务对象导出的行为），但是两者还是有区别的。通过继承继而导出服务绑定到注册表的将是真正的远程服务对象，但是通过静态方法导出服务绑定在注册表的其实是一个代理对象，也就是Proxy，该对象可以直接序列化。当RegistryIml的lookup方法调用时是直接在binds列表上拿到的对应的绑定对象，然后直接序列化。所以通过继承导出服务的将在序列化时多做一步工作，从ObjectTable里拿到对应的Target，然后拿到具体的stub（也就是Proxy），然后真正序列化写出就是这个代理对象，而不是远程服务对象。具体看：MarshalOutputStream.replaceObject

1. 分布式垃圾收集算法

10、安全策略