1. **安装打包**

**1.模块包说明**：

\*lion-boot、lion-tester会被打成可执行的、带依赖、带配置文件的jar包

\*lion-api、lion-cacher、lion-client、lion-common、lion-core、lion-monitor、lion-network、lion-registion、lion-tools被打成普通的jar包，其中lion-tools也会携带配置文件

**2.配置说明：**

\*/conf/reference.conf是默认的全部配置，当在模块中resources里有名为lion.conf的配置文件，则lion.conf会覆盖reference.conf里的相关配置参数。

**3.打包：**

\*在打包可执行的lion-boot、lion-tester前，先mvn install项目所有模块到本地

\*lion-boot打包需要根据pom/profiles节点并结合maven-assembly-plugin插件进行打包，命令为：mvn clean package -P zip -Dmaven.test.skip=true

包为：lion-release-1.0.0-SNAPSHOT.tar.gz

\*lion-tester打包需要根据pom/profiles节点并结合maven-assembly-plugin插件进行打包，命令为：mvn clean package -P test -Dmaven.test.skip=true

包为：lion-tester-1.0.0-SNAPSHOT-jar-with-dependencies.jar

1. **部署环境**

\*Zookeeper部署在245

\*Redis集群部署在237

\*lion-boot因为初步判断connection会占用较多的内存，所以也部署在245

\*lion-tester在物理机部署启动或者直接虚拟机本地启动（方便修改测试代码）

1. **测试性能**
2. **开放端口**（详见reference.conf相关参数）：

\*开启一个端口：

firewall-cmd --zone=public --add-port=80/tcp --permanent （--permanent永久生效，没有此参数重启后失效）

firewall-cmd --reload

\*服务本地监控端口：3002

\*ConnectionServer端口：3000

1. **启动zookeeper和redis集群**
2. **运行：**

lion-boot运行：

\*带配置：Java -jar -Dlion.conf=Z:\dabai-vip\lion-boot\src\main\resources\lion.conf bootstrap.jar

\*带jvm参数：java -Xms1300m -Xmx1300m -XX:PermSize=1000m -XX:MaxPermSize=1000m -XX:NewSize=200m -XX:MaxNewSize=300m -jar bootstrap.jar

1. **监控查看：**

\*使用telnet 192.168.1.245 3002 连接目标

\*使用help查看命令帮助，使用count conn查看总的连接数。

1. **测试实例：**

\*通过jar或者idea里运行lion-tester进行测试，分别启动1k、2k、6k个连接进行测试。

1. **性能分析：**

\*并发连接测试：实际并发连接测试最高到13650连接后，服务端和客户端日志lion.log里开始出现大量报错。 其中有linux文件句柄、connection读写超时次数、线程池参数、redis连接数、zookeeper连接数会限制并发数。1核4g虚拟机中，将这些瓶颈因素调节到6w并发，则发现可以达到6w并发，但是在3w左右会发现服务响应速度明显偏慢，在2w并发下可以达到一个比较不错的响应速度。由于客户端没建立一个连接需要使用一个端口，所以最多只能建立65535个连接，我只测试了6w并发。

注意：客户端建立连接只需要使用同一个bootstrap进行connect()即可，以免bootstrap实例过多，导致jvm内存溢出。

\*非并发连接测试：并发连接会有上述各种瓶颈限制，但在达到最高连接数限制后，客户端限流进行连接建立还是可以的，也就是最大可连接数实际上和服务器的内存成正相关。

\*jvm调优：建立connection实际上，新生代产生的对象较小。所有建立的connection都暂存在内存中，最终归档到老年代。所以老年代的内存相对要设置较大。而在消息发送接收过程中使用的是netty的ByteBuf，占用的是堆外内存，对jvm堆内内存并无影响。

直接缓冲ByteBuf：channel.alloc().directBuffer()

堆内缓冲ByteBuf：channel.alloc().heapBuffer()

\*jvm调优：假如业务系统要求并发要求在2w，则我们通过测试堆内存只占75M则以最小分配原则设置个100M，否则堆内存分配过多也是浪费，关键还在于每个线程栈内存需要大概1M，如果堆内存分配过大，则线程创建的数量将相应减少。

线程栈查看可以通过命令：jinfo -flag ThreadStackSize <PID>

1. **附注**

\*每台服务器能创建多少线程？

详见blogs《一台java服务器可以跑多少个线程？》

-默认情况下，一个线程的栈要预留1M的内存空间（默认线程栈），因此2G的内存也只能创建2k个线程，实际在1.6k左右。将默认栈的大小改成512K，这样理论上最多就可以开4096个线程。

-另外操作系统也会单进程能创建线程数的限制，如pid\_max、thread-max等