

1. tomcat做过哪些优化

- 优化JVM，优化JVM的根本目的是尽可能的减少GC，不论是FULL GC还是MINOR GC，减少GC，特别是FULL GC的目的是尽量减少STW（Stop The World），SWT一旦发生，在客户端的影响就是，用户会发现卡顿
 - 优化堆内存
 - 优化年轻代和老年代的比例
 - 优化GC策略
- tomcat优化
 - 根据情况，高并发情况下，适当增加线程池数量
 - `<Connector port="8080" protocol="HTTP/1.1" maxThreads="200" />`
 - 使用NIO模型，提高并发能力，新版tomcat默认模型
 - `protocol="org.apache.coyote.http11.Http11NioProtocol"`
 - 启用长连接
 - `connectionTimeout="20000" keepAliveTimeout="60000"`
 - `maxKeepAliveRequests="100"`
 - 启用GZIP

```
<Connector port="8080" protocol="HTTP/1.1"
maxThreads="500"
acceptCount="200"
connectionTimeout="20000"
compression="on"
compressionMinSize="2048"
compressableMimeType="text/html,text/xml,text/plain,text/
css,application/json,application/javascript"
URIEncoding="UTF-8"
protocol="org.apache.coyote.http11.Http11NioProtocol"
redirectPort="8443" />
```

- 内核优化
 - 上调文件描述符数量

```
ulimit -n 65535
```

2. 什么是OOM，Java程序如何解决OOM问题

- OOM

OOM即Out Of Memory, "内存用完了", 的情况在java程序中比较常见。系统会选出一个进程将之杀死, 在日志messages中看到类似下面的提示

```
# Jul 10 10:20:30 kernel: Out of memory: Kill process 9527(java) score 88 or sacrifice child
```

- 出现OOM的原因

- `java.lang.OutOfMemoryError: Java heap space`: 堆内存不足, 通常是内存泄漏或应用程序创建了太多对象
- `java.lang.OutOfMemoryError: GC overhead limit exceeded`: GC运行频率过高且无法释放足够内存
- `java.lang.OutOfMemoryError: Metaspace`: Metaspace空间不足, 通常与类加载器泄漏或类的数量超出有关
- `java.lang.OutOfMemoryError: Unable to create native thread`

- 解决方法

- 检查JVM启动参数:
 - `java -Xms` 和 `java -Xmx`, 适当增加堆内存设置, 例如 `-Xmx4g`
 - 调整eden区和幸存区的内存比例, 增大青年代的堆内存大小
- 缓存清理
- 适当增加线程数
- 更改垃圾回收器, 比如高并发场景下, 将CMS改为G1

3. 实验题: Tomcat基于MSM实现session共享

```
# 下载安装memcached
apt update; apt install memcached -y

# 查看memcached的状态
systemctl status memcached.service

# 启动脚本和配置文件
[root@ubuntu ~]# systemctl cat memcached.service | grep Exec
ExecStart=/usr/share/memcached/scripts/systemd-memcached-wrapper
/etc/memcached.conf

# nginx节点测试 Memcached远程访问

# 安装python包管理器pypi
apt install -y python3-pip

# 安装python客户端工具
pip install python-memcached

# 写一个memcache测试脚本
```

```

vim test.py

#!/usr/bin/python3

import memcache
client = memcache.Client(['10.0.0.150:11211', '10.0.0.151:11211'], debug=True)

for i in client.get_stats('items'):
    print(i)

print('-' * 35)

for i in client.get_stats('cachedump 5 0'):
    print(i)

# 执行python脚本
[root@mystical ~] $python3 test.py
('10.0.0.150:11211 (1)', {})
('10.0.0.151:11211 (1)', {})
-----
('10.0.0.150:11211 (1)', {})
('10.0.0.151:11211 (1)', {})

# 显示无任何数据

# 配置（150的session写到151的memcached；151的session写到150的memcached）
# tomcat节点配置session共享

# 10.0.0.150

# 下载相关jar包
cd /usr/local/tomcat/lib

wget https://repo1.maven.org/maven2/org/ow2/asm/asm/5.2/asm-5.2.jar
wget https://repo1.maven.org/maven2/com/esotericsoftware/kryo/3.0.3/kryo-3.0.3.jar
wget https://repo1.maven.org/maven2/de/javakaffee/kryo-serializers/0.45/kryo-serializers-0.45.jar
wget https://repo1.maven.org/maven2/de/javakaffee/msm/memcached-session-manager/2.3.2/memcached-session-manager-2.3.2.jar
wget https://repo1.maven.org/maven2/de/javakaffee/msm/memcached-session-manager-tc9/2.3.2/memcached-session-manager-tc9-2.3.2.jar
wget https://repo1.maven.org/maven2/com/esotericsoftware/minlog/1.3.1/minlog-1.3.1.jar
wget https://repo1.maven.org/maven2/de/javakaffee/msm/msm-kryo-serializer/2.3.2/msm-kryo-serializer-2.3.2.jar
wget https://repo1.maven.org/maven2/org/objenesis/objenesis/2.6/objenesis-2.6.jar
wget https://repo1.maven.org/maven2/com/esotericsoftware/reflectasm/1.11.9/reflectasm-1.11.9.jar
wget https://repo1.maven.org/maven2/net/spy/spymemcached/2.12.3/spymemcached-2.12.3.jar

# 修改配置
cat /usr/local/tomcat/conf/context.xml

```

```
# 添加下列内容
# 10.0.0.150
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
memcachedNodes="m1:10.0.0.150:11211,m2:10.0.0.151:11211" failoverNodes="m1"
requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"
transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory"/>

# 10.0.0.151
<Manager className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
memcachedNodes="m1:10.0.0.150:11211,m2:10.0.0.151:11211" failoverNodes="m2"
requestUriIgnorePattern=".*\.(ico|png|gif|jpg|css|js)$"
transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory"/>

# 150和151都重启tomcat服务
systemctl restart tomcat
```

4. 基于客户端的分布式集群和基于服务端的分布式集群有什么区别

客户端的分布式集群

客户端的分布式集群是指客户端负责将请求分发到不同的服务器节点，并自行管理分布式逻辑，而服务器是无状态的，客户端决定要与哪个服务器进行交互。

Memcached是客户端分布式集群

1. 无中心协调：Memcached服务器本身不维护关于其他服务器状态或数据存放位置的信息，它们独立运行，不需要彼此通信来处理缓存数据
2. 客户端决策：数据在哪里存储（哪台Memcached服务器）由客户端决定。客户端使用一致性hash等算法选择数据应该存储在哪个服务器上，这种方法减少了单点故障的风险，因为没有中央节点是关键的
3. 简单性和性能：这种模式使Memcached非常简单和高效。服务器仅仅响应客户端的请求，所有智能决策都由客户端完成。这减轻了服务器的计算负担，使系统能更快地响应。

服务端的分布式集群

服务端的分布式集群是指由服务器来统一管理请求的分发和任务的调度，客户端只和一个负载均衡器或网关进行通信，后端的分发逻辑和集群拓扑结构对客户端透明。

Web应用：前端请求通过 Nginx（或负载均衡）转发到后端的多个应用实例

5. 使用jvisualvm观察下面的Java程序在堆中的eden区，幸存区和老年区之间GC的过程，并截图

```
// HeapOom.java
import java.util.ArrayList;
import java.util.List;
public class HeapOom {
```

```

public static void main(String[] args) {
    List<byte[]> list = new ArrayList<byte[]>();
    int i = 0;
    boolean flag = true;
    while(flag) {
        try {
            i++;
            list.add(new byte[1024 * 1024]); // 每次增加一个1M大小的数组对象
            Thread.sleep(1000);
        } catch (Throwable e) {
            e.printStackTrace();
            flag = false;
            System.out.println("count="+i); // 记录运行次数
        }
    }
}

// 使用如下指令编译代码执行
// javac HeapOom.java
// java HeapOom

```

使用jvisualvm监控内存

jvisualvm一款图形化的内存监控工具，在jdk-8u361之前的版本中是内置的组件，但在之后的JDK版本中已经取消了该组件，要单独下载并配置

```

# 安装依赖
apt install libxrender1 libxrender1 libxtst6 libxi6 fontconfig -y

wget https://github.com/oracle/visualvm/releases/download/2.1.8/visualvm_218.zip

unzip visualvm_218.zip

# 在windows中开启Xmanager - Passive
export DISPLAY=10.0.0.1:0.0

# 执行，在windows中能看到GUI界面，在GUI中点击Tools菜单，Plugins，然后安装VisualGC插件

```

测试程序

```

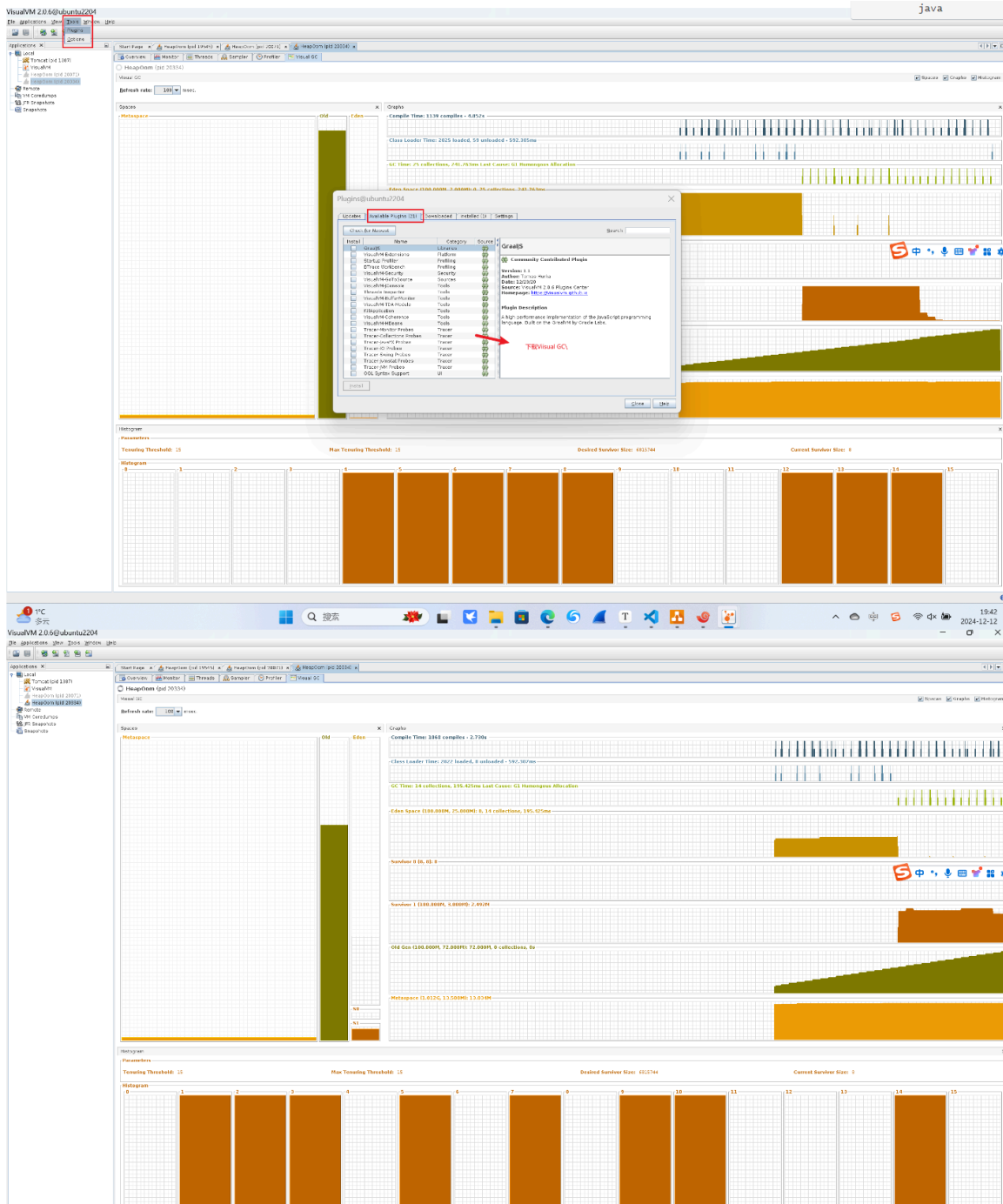
// HeapOom.java
import java.util.ArrayList;
import java.util.List;
public class HeapOom {
    public static void main(String[] args) {
        List<byte[]> list = new ArrayList<byte[]>();
        int i = 0;
        boolean flag = true;
        while(flag) {
            try {

```

```

        i++;
        list.add(new byte[1024 * 1024]); // 每次增加一个1M大小的数组对象
        Thread.sleep(1000);
    } catch (Throwable e) {
        e.printStackTrace();
        flag = false;
        System.out.println("count="+i); // 记录运行次数
    }
}
}
}

```



6. 你在工作中监控过java程序的哪些指标

[查看资源请求情况详情](#)

Version 4.1.4 running on ubuntu2204, UP for 0 days 2 hours 1 minutes										Installed applications	
Applications Data Sources Deployment Logs Threads Cluster System Connectors Certificates Quick check											
Application statistics											
请求 session sessiontimeout 调用的JSP文件											
What are those abbreviations? estimate sessions size (could be slow)											
NAME	STATUS	DESCRIPTION	REQ.	SESS.	S.ATTR	C.ATTR	SESS.TIMEOUT	JSP	JDBC USAGE	CLSTRED.?	SER.?
/	running	Welcome to Tomcat	0	0	0	9	30			no	yes
/examples	running	Servlet and JSP Examples	0	0	0	10	30			no	yes
/host-manager	running	Tomcat Host Manager Application	0	0	0	10	30			no	yes
/manager	running	Tomcat Manager Application	0	0	0	10	30			no	yes
/docs	running	Tomcat Documentation	0	0	0	9	30			no	yes
/probe	running	PSI Probe for Apache Tomcat v4.1.4	948	82	246	16	15			no	yes

通过request查看

Version 4.1.4 running on ubuntu2204, UP for 0 days 1 hours 57 minutes											
Applications Data Sources Deployment Logs Threads Cluster System Connectors Certificates Quick check											
Servlet mappings											
SERVLETS 资源路径 类 请求数 (可以看作点击率)											
APP	NAME	AVAIL	STARTUP	LOAD TIME	REQ	PROC TIME	ERR	MIN TIME	MAX TIME	MULT THRD	
/probe	probe org.apache.catalina.servlets.DefaultServlet	yes	0	235	822	0:00:29.569	0	2	1464	yes	
/probe	default org.apache.catalina.servlets.DefaultServlet	yes	1	9	50	0:00:00.582	0	1	87	yes	
/probe	jsp org.apache.jasper.servlet.JspServlet	yes	3	21	1	0:00:06.598	1	6598	6598	yes	
/	default org.apache.catalina.servlets.DefaultServlet	yes	1	0	0	0:00:00.000	0	0	0	yes	
/	jsp org.apache.jasper.servlet.JspServlet	yes	3	1	0	0:00:00.000	0	0	0	yes	
/examples	RequestInfoExample RequestInfoExample	yes	no	0	0	0:00:00.000	0	0	0	yes	
/examples	async3 async.Async3	yes	no	0	0	0:00:00.000	0	0	0	yes	
/examples	numberwriter nonblocking.NumberWriter	yes	no	0	0	0:00:00.000	0	0	0	yes	
/examples	async2 async.Async2	yes	no	0	0	0:00:00.000	0	0	0	yes	
/examples	async1 async.Async1	yes	no	0	0	0:00:00.000	0	0	0	yes	
/examples	jsp org.apache.jasper.servlet.JspServlet	yes	3	3	0	0:00:00.000	0	0	0	yes	
/examples	async0 async.Async0	yes	no	0	0	0:00:00.000	0	0	0	yes	
/examples	bytecounter nonblocking.ByteCounter	yes	no	0	0	0:00:00.000	0	0	0	yes	
/examples	responseTrailer Trailers.ResponseTrailers	yes	no	0	0	0:00:00.000	0	0	0	yes	
/examples	ServletToJsp ServletToJsp	yes	no	0	0	0:00:00.000	0	0	0	yes	
/examples	RequestParamExample RequestParamExample	yes	no	0	0	0:00:00.000	0	0	0	yes	
/examples	RequestHeaderExample RequestHeaderExample	yes	no	0	0	0:00:00.000	0	0	0	yes	
/examples	default org.apache.catalina.servlets.DefaultServlet	yes	1	11	0	0:00:00.000	0	0	0	yes	
/examples	CookieExample CookieExample	yes	no	0	0	0:00:00.000	0	0	0	yes	

Session

Toggle Expire Session search										estimate sizes Show all	
SESSIONS 点击具体的资源的session的数值即可进入, 查看具体的session信息, 人员, 地域, 访问时间, IP等											
96 items found, displaying 1 to 50. [First/Prev] 1 2 [Next/Last]											
SESSION ID	LAST IP	IDLE TIME	AGE	EXPIRY TIME	OBJECT COUNT	SER.					
F2739804723EBEBC5CAEAE8BCF034C16-m2	10.0.0.1	0:05:21.742	0:05:21.745	Wed Dec 11 23:54:09 CST 2024	3	yes					
73F388CD94E9EF429CF99190E9688EF6-m2	10.0.0.1	0:05:32.462	0:05:32.464	Wed Dec 11 23:53:58 CST 2024	3	yes					
FCB92E9F7590C28E3AF8551080117230-m2	10.0.0.1	0:08:35.046	0:08:35.055	Wed Dec 11 23:50:55 CST 2024	3	yes					
363A19FDEBA73625B8EF5C2D3DE3EEB-m2	10.0.0.1	0:09:26.322	0:09:26.336	Wed Dec 11 23:50:04 CST 2024	3	yes					
FFF139788861502479E6819164C13AD4-m2	10.0.0.1	0:07:54.744	0:07:54.755	Wed Dec 11 23:51:36 CST 2024	3	yes					
9395685C0D83F7E669667E1B45AAEAF-m2	10.0.0.1	0:05:33.167	0:05:33.174	Wed Dec 11 23:53:57 CST 2024	3	yes					
58927E451CD67DC88F0DA69951AD3ED2-m2	10.0.0.1	0:00:23.858	0:00:23.867	Wed Dec 11 23:59:06 CST 2024	3	yes					
3A1E7A98CB0868D7918F487DB1C7C24-m2	10.0.0.1	0:05:33.106	0:05:33.109	Wed Dec 11 23:53:57 CST 2024	3	yes					
3B0CF00EE5C34DE6C9D1706AE69A01B-m2	10.0.0.1	0:07:00.856	0:07:00.876	Wed Dec 11 23:52:29 CST 2024	3	yes					
F6F6A0E12E6DFC6AE68AEF3DEED3AC-m2	10.0.0.1	0:07:25.878	0:07:25.882	Wed Dec 11 23:52:04 CST 2024	3	yes					
C092746D49158D6D83D7913D3A0DBD55-m2	10.0.0.1	0:06:30.867	0:06:30.879	Wed Dec 11 23:52:59 CST 2024	3	yes					
003C5B228A0CEDB6D4CE02799D3064C8-m2	10.0.0.1	0:06:00.992	0:06:00.996	Wed Dec 11 23:53:29 CST 2024	3	yes					
337AD093DB921185C2A6291EFB861460-m2	10.0.0.1	0:05:48.783	0:05:48.825	Wed Dec 11 23:53:42 CST 2024	3	yes					
23538484244671FB41B2AA72BF770076-m2	10.0.0.1	0:02:24.923	0:02:24.935	Wed Dec 11 23:57:05 CST 2024	3	yes					
34F080D4A813E6AAC7332C388857A237-m2	10.0.0.1	0:09:26.318	0:09:26.342	Wed Dec 11 23:50:04 CST 2024	3	yes					
C6B5ABC3AFB6568C5B2EB86D666696E0-m2	10.0.0.1	0:06:24.849	0:06:24.863	Wed Dec 11 23:53:05 CST 2024	3	yes					
77F2FC4F4109138933E9D3908D89C19C-m2	10.0.0.1	0:05:34.571	0:05:34.580	Wed Dec 11 23:53:56 CST 2024	3	yes					
195BFDCDFB0DBAE15A3019D47892250-m2	10.0.0.1	0:08:09.844	0:08:09.868	Wed Dec 11 23:51:20 CST 2024	3	yes					
19ED022D12646FCDOCA25AE0D69C9F86-m2	10.0.0.1	0:08:24.986	0:08:24.991	Wed Dec 11 23:51:05 CST 2024	3	yes					
FA02B4B1581A177B97908CC5518CE67-m2	10.0.0.1	0:04:24.991	0:04:25.009	Wed Dec 11 23:55:05 CST 2024	3	yes					
C1482975A4BD1187407EBADA43DBF480-m2	10.0.0.1	0:08:30.019	0:08:30.028	Wed Dec 11 23:51:00 CST 2024	3	yes					
217F83E75458EB28CE962234874030C-m2	10.0.0.1	0:05:34.527	0:05:34.527	Wed Dec 11 23:53:56 CST 2024	3	yes					
65883A9942B7A11807B8B63ED0E5A55-m2	10.0.0.1	0:05:27.836	0:05:27.838	Wed Dec 11 23:54:02 CST 2024	3	yes					
11063D9D870DA19F3B0468F2F68FD75-m2	10.0.0.1	0:05:55.767	0:05:55.769	Wed Dec 11 23:53:35 CST 2024	3	yes					
E3E7D6E7813EB8C53C281913E2ED2AA5-m2	10.0.0.1	0:05:32.412	0:05:32.416	Wed Dec 11 23:53:58 CST 2024	3	yes					
E05BF227B36E6C41BE60554499D98-m2	10.0.0.1	0:08:50.175	0:08:50.191	Wed Dec 11 23:50:40 CST 2024	3	yes					
9B734346FCB1341BF999008715E151D-m2	10.0.0.1	0:05:39.394	0:05:39.394	Wed Dec 11 23:53:51 CST 2024	3	yes					
1C7E5036CF514DF4EBC6A34F9790F6-m2	10.0.0.1	0:10:55.882	0:10:55.891	Wed Dec 11 23:48:34 CST 2024	3	yes					
EC97E8C49F8AC31F7E23C192C19D8A3A-m2	10.0.0.1	0:07:42.851	0:07:42.871	Wed Dec 11 23:51:47 CST 2024	3	yes					
F51F40CD95D1695BE680612BC9F95B28-m2	10.0.0.1	0:05:32.393	0:05:32.411	Wed Dec 11 23:53:58 CST 2024	3	yes					
8B7AE1EA70A70A807CAAB6C728320385-m2	10.0.0.1	0:05:21.759	0:05:21.759	Wed Dec 11 23:54:09 CST 2024	3	yes					

线程池数据 (连接数)

- 根据下列指标即可看出线程池是否够用, 是否需要增加线程池的线程数量

10.0.0.202:8082/probe/threads.htm									
应用 Learn VM while p... Google https://twitter.co... 仪表板 数据结构基础 Cou... Computer Science... JS官方文档 openai/openai-co... Python零基础需... Shell-官方查询帮助... 力扣 (LeetCode) ...									
Version 4.1.4 running on ubuntu2204.wang.org, UP for 0 days 2 hours 20 minutes									
Applications Data Sources Deployment Logs Threads Cluster System Connectors Certificates Quick check									
Threads Pools What are those abbreviations?									
ID	NAME	EXEC. POINT	STATE	IN-NATIVE	SUSP.	WC	BC		
1	main	java.net.PlainSocketImpl.socketAccept (native code)	RUNNABLE	true	false	1	0		
12	AsyncFileHandler-1	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	29	0		
15	logback-1	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	108	0		
17	Probe_Quartz-1	java.lang.Object.wait (native code)	TIMED_WAITING	false	false	16219	270		
18	Probe_Quartz-2	java.lang.Object.wait (native code)	TIMED_WAITING	false	false	16223	271		
19	Probe_Quartz-3	java.lang.Object.wait (native code)	TIMED_WAITING	false	false	16218	288		
2	Reference Handler	java.lang.ref.Reference.waitForReferencePendingList (native code)	RUNNABLE	false	false	0	4		
20	Probe_Quartz-4	java.lang.Object.wait (native code)	TIMED_WAITING	false	false	16212	283		
21	Probe_Quartz-5	java.lang.Object.wait (native code)	TIMED_WAITING	false	false	16206	271		
22	ProbeScheduler_QuartzSchedulerThread	java.lang.Object.wait (native code)	TIMED_WAITING	false	false	874	638		
23	Log4j2-TF-5-Scheduled-1	jdk.internal.misc.Unsafe.park (native code)	TIMED_WAITING	false	false	270	0		
24	mysql-cj-abandoned-connection-cleanup	java.lang.Object.wait (native code)	TIMED_WAITING	false	false	1619	2		
25	HikariPool-1 housekeeper	jdk.internal.misc.Unsafe.park (native code)	TIMED_WAITING	false	false	273	0		
27	TokenManager	java.lang.Object.wait (native code)	TIMED_WAITING	false	false	55	1		
28	ForkJoinPool.commonPool-worker-3	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	90	0		
29	jboot-scheduler-thread-1	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	107	0		
3	Finalizer	java.lang.Object.wait (native code)	WAITING	false	false	67	66		
30	jboot-scheduler-thread-2	jdk.internal.misc.Unsafe.park (native code)	TIMED_WAITING	false	false	117	0		
31	jboot-scheduler-thread-3	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	136	0		
32	jboot-scheduler-thread-4	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	98	0		
33	jboot-scheduler-thread-5	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	82	0		
34	jboot-scheduler-thread-6	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	117	0		
35	jboot-scheduler-thread-7	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	216	0		
36	jboot-scheduler-thread-8	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	362	0		
37	ContainerBackgroundProcessor[StandardEngine[Catalina]]	java.lang.Thread.sleep (native code)	TIMED_WAITING	false	false	809	0		
38	http-nio-8082-exec-1	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	73	69		
39	http-nio-8082-exec-2	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	73	20		
40	http-nio-8082-exec-3	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	73	63		
41	http-nio-8082-exec-4	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	74	55		
42	http-nio-8082-exec-5	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	75	3		
43	http-nio-8082-exec-6	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	76	6		
44	http-nio-8082-exec-7	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	74	28		
45	http-nio-8082-exec-8	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	76	1		
46	http-nio-8082-exec-9	sun.management.ThreadImpl.getThreadInfo1 (native code)	RUNNABLE	false	false	76	7		
47	http-nio-8082-exec-10	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	73	66		
48	http-nio-8082-Poller	sun.nio.ch.EPoll.wait (native code)	RUNNABLE	true	false	0	181		
49	http-nio-8082-Acceptor	sun.nio.ch.ServerSocketChannelImpl.accept0 (native code)	RUNNABLE	true	false	0	0		
50	http-nio-8082-AsyncTimeout	java.lang.Thread.sleep (native code)	TIMED_WAITING	false	false	8083	0		
52	Java2D Disposer	java.lang.Object.wait (native code)	WAITING	false	false	15	14		
54	logback-3	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	47	0		
55	lboot-scheduler-thread-9	jdk.internal.misc.Unsafe.park (native code)	WAITING	false	false	91	0		

http线程池，当前一共10个
只有一个在使用RUNNABLE
如果都在使用，就表示繁忙。
此时可能需要着增加连接数

内存数据

Version 4.1.4 running on ubuntu2204, UP for 0 days 2 hours 19 minutes [这里可以查看系统资源](#) System information

Applications | Data Sources | Deployment | Logs | Threads | Cluster | **System** | Connectors | Certificates | Quick check

MEMORY UTILIZATION

CURRENT MEMORY USAGE 16% [Advise Garbage Collection](#)

FREE: 40.86 MB TOTAL: 118.00 MB MAX: 482.00 MB

OS INFORMATION

JVM: [Java\(TM\) SE Runtime Environment 11.0.22+7-Ubuntu-22.04-222 Java HotSpot\(TM\) 64-Bit Server VM](#)

OS: Linux (unknown) amd64 5.15.0-125-generic

PROCESSORS: 2

CURRENT TIME: Wed Dec 11 23:56:23 CST 2024

WORKING DIR: /

CONTAINER INFORMATION

CONTAINER: Apache Tomcat/9.0.89

CATALINA_BASE: /usr/local/apache-tomcat-9.0.89

CATALINA_HOME: /usr/local/apache-tomcat-9.0.89

APPLICATION_BASE: /usr/local/apache-tomcat-9.0.89/webapps

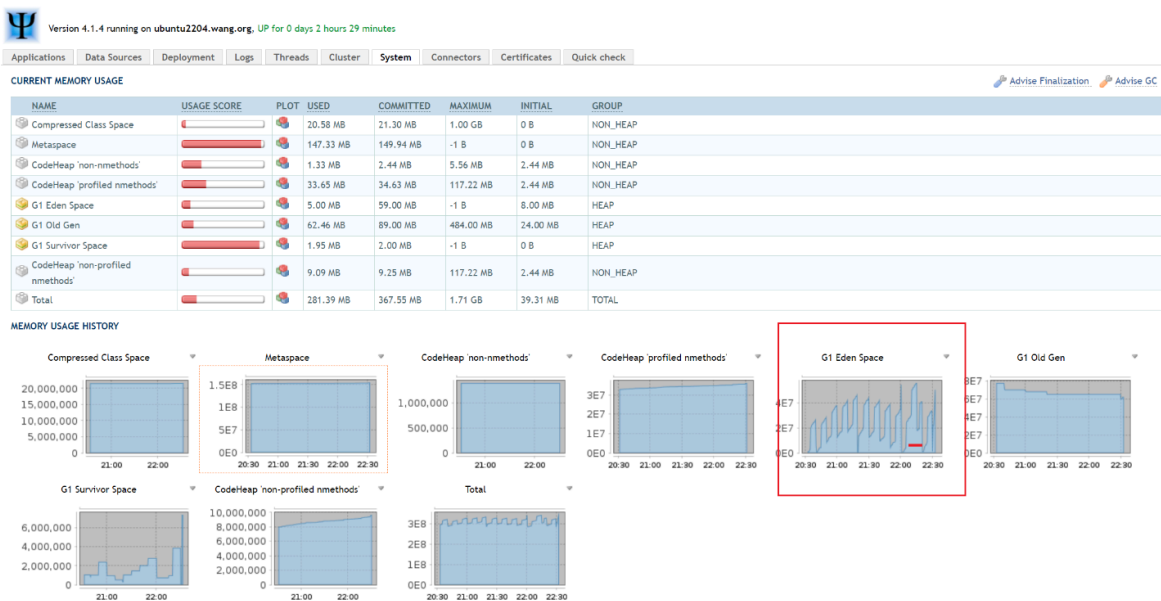
CONFIGURATION_BASE: /usr/local/apache-tomcat-9.0.89/conf/localhost

SYSTEM

- Overview
- Memory utilization**
- System properties
- OS information
- OS/Hardware Info (Slow)
- Wrapper control
- Trust Store

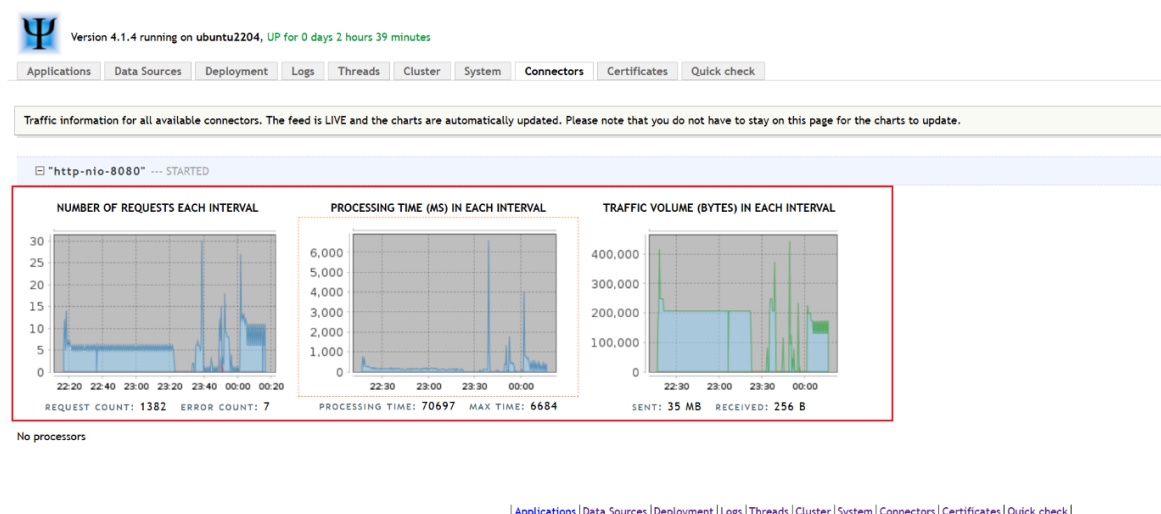
点击这里可以查看内存变化

10.0.0.202:8082/probe/memory.htm



连接器Connector

- 观察 (http) 连接器里的点击率和吞吐量



7. java对象进入老年代的原因（最少说3种）

三种对象从青年代移动到老年代的原因

- 对象的年龄超过阈值（默认15，具体看JVM实现）
 - 当对象在 Survivor 区的年龄超过了阈值，JVM 会将对象移动到老年代
 - 过程解释

- 在年轻代中，内存划分为 Eden、Survivor from (S0) 和 Survivor to (S1) 三部分。
- 新创建的对象首先分配在 Eden 区
- 当 Eden 区内存满时，Minor GC 触发
- 存活的对象将被移入 Survivor S0 区。
- 在接下来的 GC 中，存活的对象会从 S0 → S1，它们的“年龄”会增加 1。
- 当对象的年龄达到某个阈值 (默认为15)，JVM 就会将该对象移入老年代。
- **年龄阈值**
 - JVM 中的对象年龄由 MaxTenuringThreshold 决定，默认值是15。
 - 也就是说，如果一个对象的年龄达到 15 (在 S0/S1 之间存活了 15 次 GC)，这个对象就会被移到老年代
- **修改阈值**

```
-XX:MaxTenuringThreshold=10
```

```
# 将对象的“最大生存代数”调整为 10。也就是说，当对象在 Survivor 区经历了 10 次 GC 后，它就会被移入老年代。
```

- 对象的年龄超过阈值
 - 大对象 (通常是大数组或大字符串) 会直接分配到老年代，而不会经历年轻代的过程。
 - **为什么要直接进入老年代?**
 - 如果一个大对象 (比如一个大数组或大字符串) 被分配到 Eden 区，可能会很快填满 Eden 区，频繁触发 GC。这种频繁的 GC 会导致性能问题。为了解决这个问题，JVM 提供了一种机制，大对象直接进入老年代。
 - **如何定义大对象的大小?**
 - JVM 使用 PretenureSizeThreshold 参数来决定大对象的最小大小。
 - 只要对象的大小大于 PretenureSizeThreshold，它就会被直接分配到老年代。
 - **如何设置大对象的阈值?**

```
# 表示大于 1MB 的对象将直接分配到老年代。
```

```
-XX:PretenureSizeThreshold=1M
```

- 特殊情况：**Survivor 区满了的情况**
 - 如果 Survivor 区满了，存活的对象也会被直接送入老年代。
 - **为什么会发生?**
 - 在 GC 过程中，如果 Survivor 区中的空间不足以存放所有 Eden 和 Survivor 中存活的对象，那么一部分对象会直接被分配到老年代。
 - 这有点像应急转移的机制。
 - 这种情况在高并发场景中常见，比如网络高并发请求，瞬时大量对象创建导致 Survivor 区不够用。

堆满时，老年代如何分配?

- 当老年代也满了时，JVM 将触发 Full GC。
- Full GC 触发后，如果无法回收老年代中的对象，JVM 会抛出：

```
java.lang.OutOfMemoryError: Java heap space
```

8. 创建Nexus服务，并配置Maven、apt、yum的私有仓，并在客户端测试，看是否生效

Nexus安装和配置Maven仓库

实验准备

- 设备1
 - IP: 10.0.0.150
 - 部署服务: java, Maven
 - 备注: 将Maven中的仓库配置指向自建的私有仓库
- 设备2
 - IP: 10.0.0.182 (重点)
 - 部署服务: Nexus Server
 - 内存需要4G以上，实现Maven的仓库，apt仓库，yum仓库

安装Nexus，并配置Maven仓库

```
# 安装jdk-11
dpkg -i jdk-11.0.23_linux-x64_bin.deb

wget https://download.sonatype.com/nexus/3/nexus-3.67.1-01-java11-unix.tar.gz

tar xf nexus-3.67.1-01-java11-unix.tar.gz -C /usr/local/

ln -sv /usr/local/nexus-3.67.1-01/ /usr/local/nexus

# 主配置文件，指定监听的端口号和IP
## DO NOT EDIT - CUSTOMIZATIONS BELONG IN $data-dir/etc/nexus.properties
##
# Jetty section
application-port=8081
application-host=0.0.0.0
nexus-args=${jetty.etc}/jetty.xml,${jetty.etc}/jetty-http.xml,${jetty.etc}/jetty-requestlog.xml
nexus-context-path=/

# Nexus section
nexus-edition=nexus-pro-edition
nexus-features=\
    nexus-pro-feature

nexus.hazelcast.discovery.isEnabled=true

# 主要文件
[root@mystical /usr/local/nexus] $tree bin
bin
├─ contrib
└─ karaf-service.sh
```

```

|   ├── karaf-service-template.conf
|   ├── karaf-service-template.init
|   ├── karaf-service-template.init-debian
|   ├── karaf-service-template.init-redhat
|   ├── karaf-service-template.solaris-smf
|   ├── karaf-service-template.systemd
|   ├── karaf-service-template.systemd-instances
|   ├── karaf-service-win.exe
|   └── karaf-service-win.xml
└─ nexus                # 可执行二进制脚本文件，主程序
└─ nexus.rc              # 配置运行时用户身份
└─ nexus.vmoptions       # 配置服务启动时的Java选项

```

可以使用nexus.rc 指定执行程序的身份

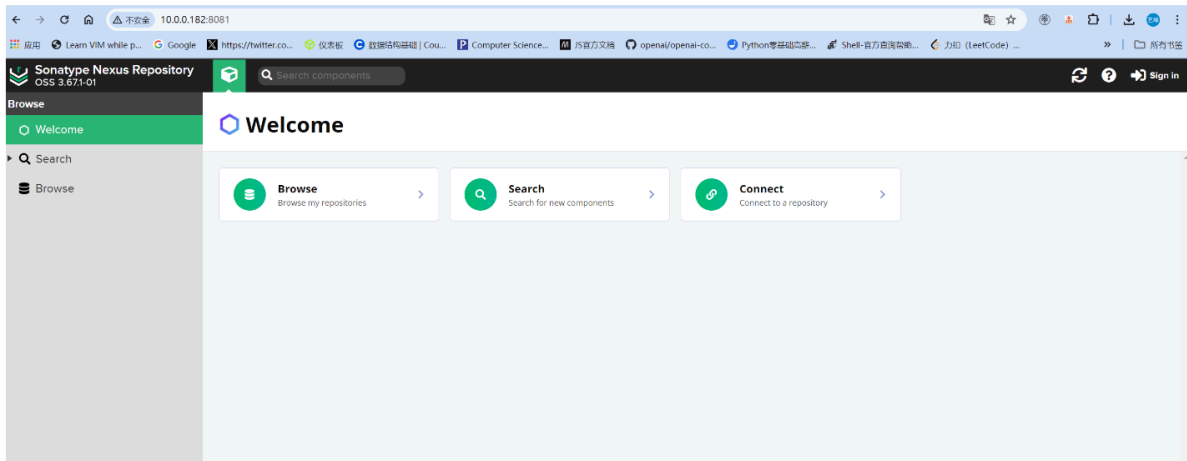
```
run_as_user="root"
```

将二进制执行文件加入全局变量

```
ln -sv /usr/local/nexus/bin/nexus /usr/bin/
```

启用nexus

在浏览器访问10.0.0.182:8081



配置服务脚本，使用systemctl 来进行服务管理

参考: <https://help.sonatype.com/en/installation-and-upgrades.html>

```

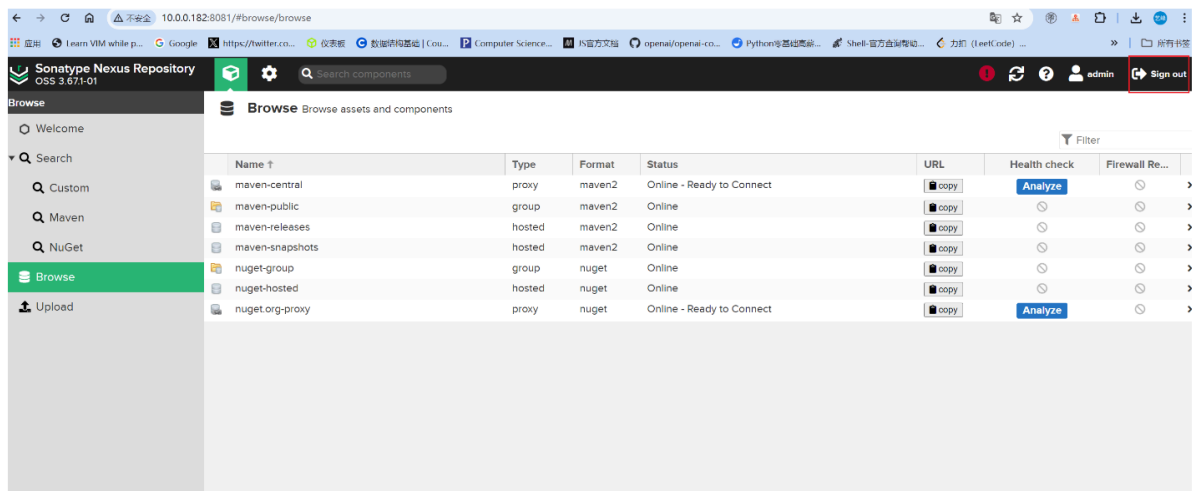
cat lib/systemd/system/nexus.service
[Unit]
Description=nexus service
After=network.target

[Service]
Type=forking
LimitNOFILE=65536
ExecStart=/usr/local/nexus/bin/nexus start
ExecStop=/usr/local/nexus/bin/nexus stop
User=root
#User=nexusvim v
Restart=on-abort

[Install]

```

wantedBy=multi-user.target



首次登录，用户名为admin; 密码在/usr/local/sonatype-work/nexus3/admin.password文件中

首次登录成功后，可以修改新的密码

首次登录成功后修改密码，并选择支持匿名访问，做成公开库

proxy 代理仓，如果nexus服务器上没有，则先去maven的官方仓库下载回来

hosted 本地仓，如果nexus服务器上没有，不会去公网下载

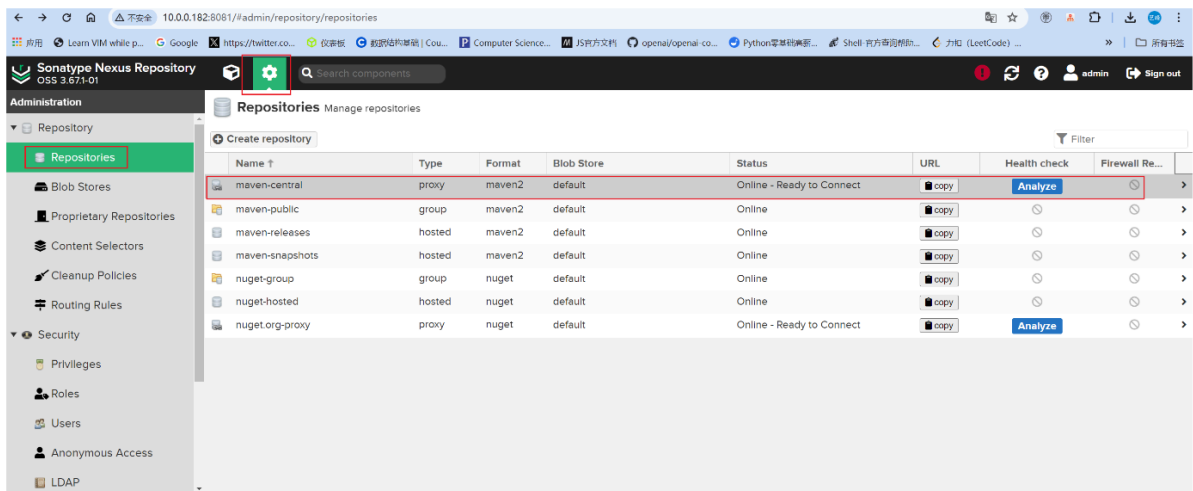
group 多个仓库集合，及支持maven仓库，yum和apt不能合并

https://maven.aliyun.com 也有集合

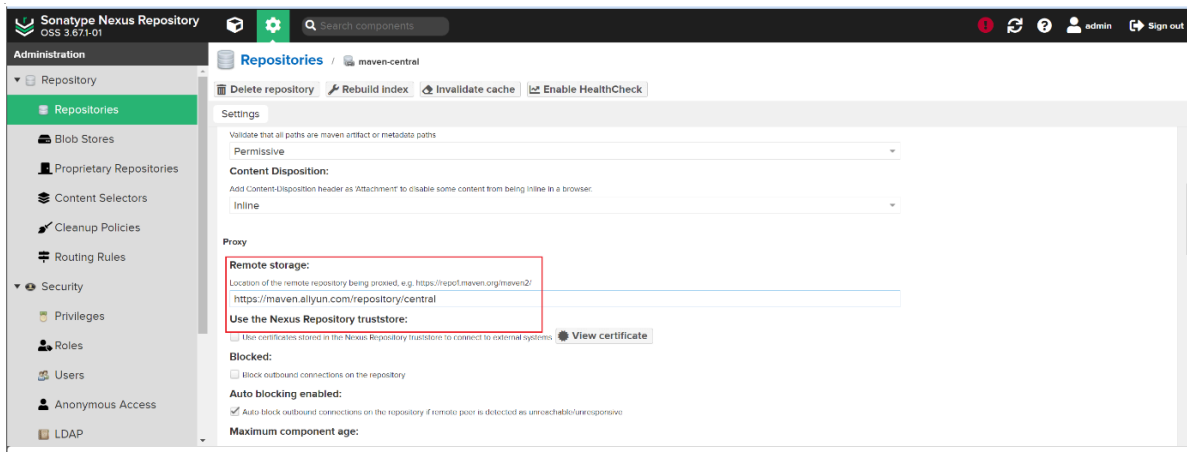
配置maven-central的公网地址以为国内阿里云

https://maven.aliyun.com/repository/central

• 点击



• 添加阿里的镜像仓库



在Maven主机上测试自建仓库(10.0.0.150)

```
cat /usr/local/maven/conf/setting.xml
```

```
<mirror>
  <id>nexus-maven</id>
  <mirrorOf>*</mirrorOf>
  <name>自建仓库</name>
  <url>http://10.0.0.182:8081/repository/maven-central/</url>
</mirror>
```

#在nexus 服务器上，对应的包文件放在此目录中

```
[root@ubuntu ~]# du -sh /usr/local/sonatype-work/nexus3/blobs/default/
56M /usr/local/sonatype-work/nexus3/blobs/default/
#但并不是可读的友好格式，这是基于安全方面的考虑，保有在 web 页面可见
[root@ubuntu ~]# tree /usr/local/sonatype-work/nexus3/blobs/default/
```

Nexus配置Apt仓库

1. 创建目录/data/blobs
2. 进入nexus配置界面，在Blob Stores页面点击Create Blob Store，创建一个存储，Type选择File，Name填Ubuntu2204，Path填/data/blobs/ubuntu2204，点击保存
3. 在Repositories页面点击Create repository,选择apt(proxy)，在新页面中Name填Ubutun2204，Distribution填jammy,Remote storage 填<https://mirror.aliyun.com/ubuntu/>，Blob store 选择ubuntu2204，点击保存
4. 在Browse页面选择刚创建的ubuntu2204，点击copy，复制原地址，然后修改一台ubuntu主机的/etc/apt/sources.list,将源指向自建的nextus服务器中的原地址
5. 测试

Nexus配置yum仓库

同上