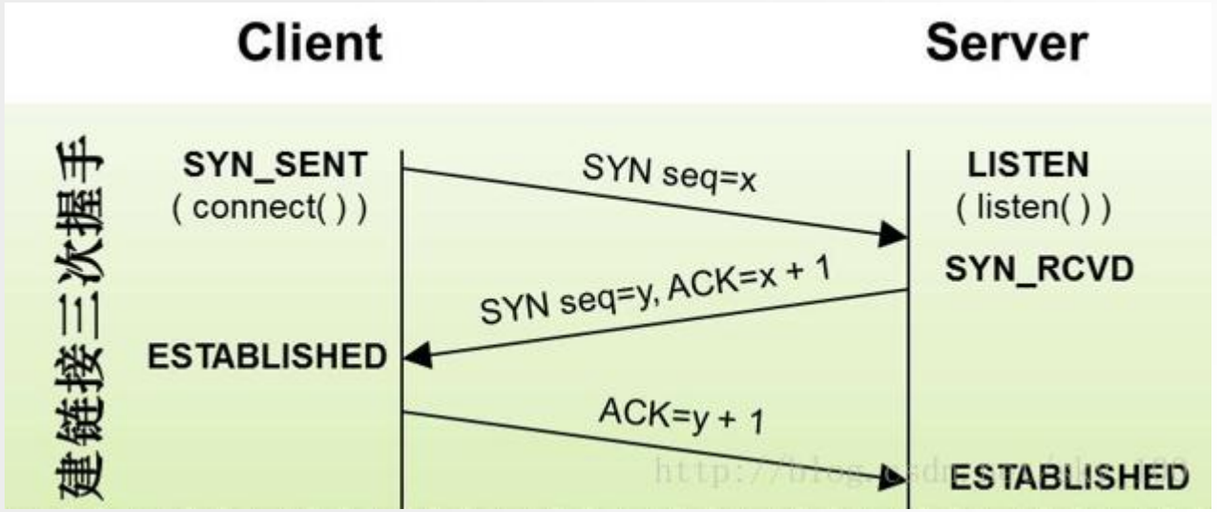
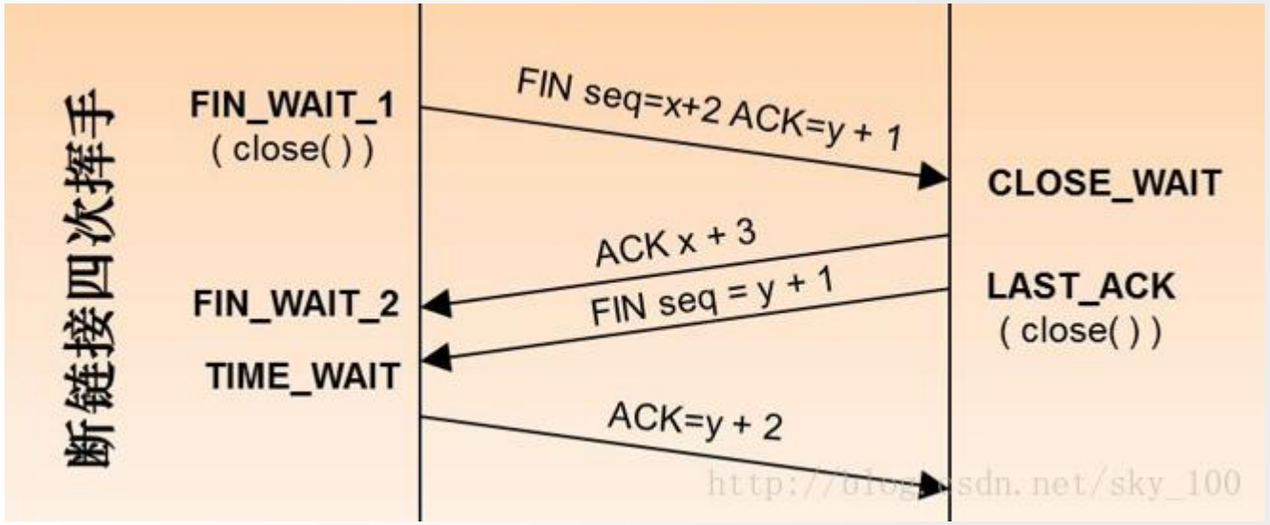
目录 /CONTENTS

01

工作原理

WORKING PRINCIPLE






```
netstat -n | awk '/^tcp/ {++S[$NF]} END {for(a in S) print a, S[a]}'
```

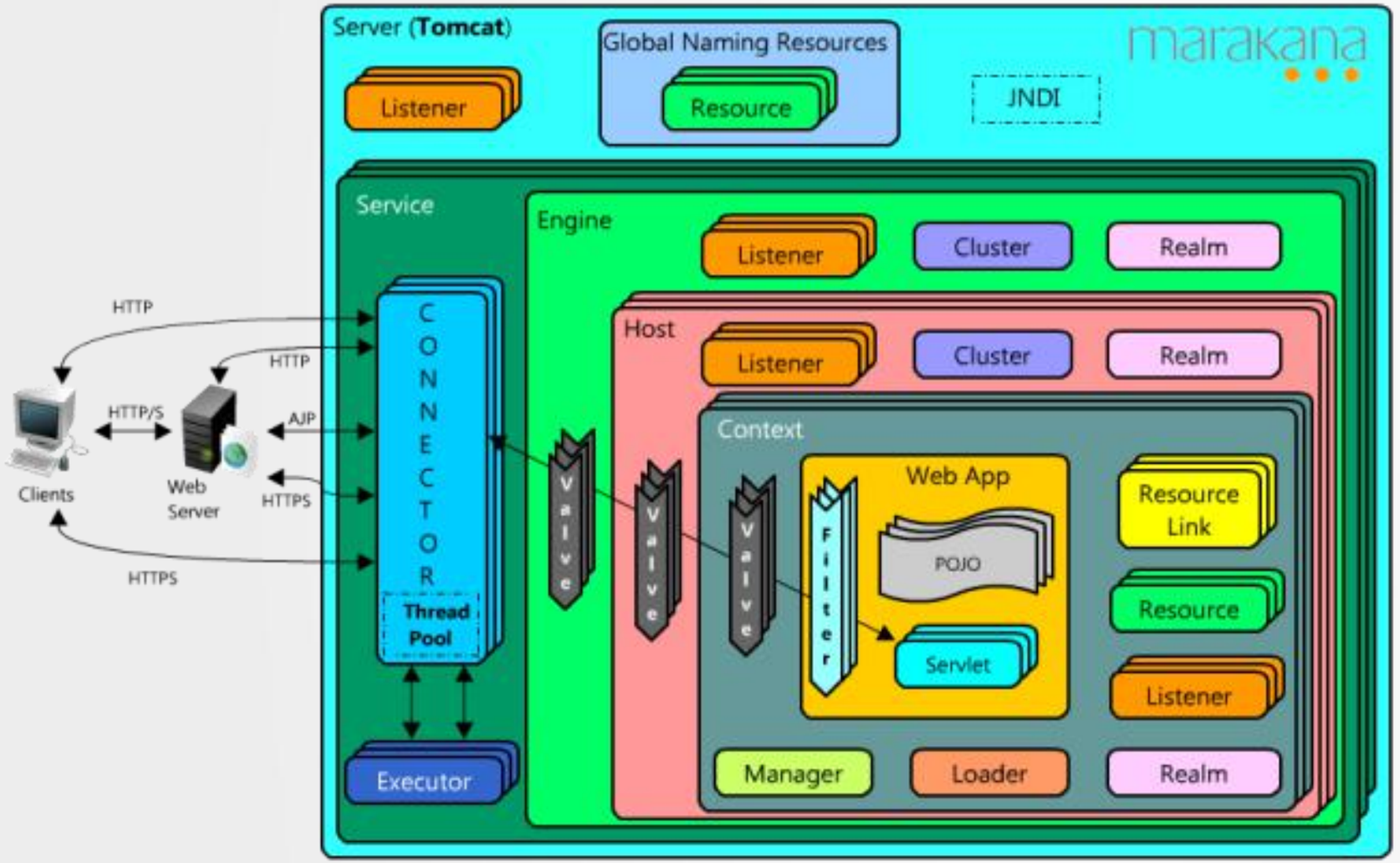
返回结果一般如下：
LAST_ACK 5 （正在等待处理的请求数）
SYN_RECV 30
ESTABLISHED 1597 （正常数据传输状态）
FIN_WAIT1 51
FIN_WAIT2 504
TIME_WAIT 1057 （处理完毕，等待超时结束的请求数）

其他参数说明：
CLOSED：无连接是活动的或正在进行
LISTEN：服务器在等待进入呼叫
SYN_RECV：一个连接请求已经到达，等待确认
SYN_SENT：应用已经开始，打开一个连接
ESTABLISHED：正常数据传输状态
FIN_WAIT1：应用说它已经完成
FIN_WAIT2：另一边已同意释放
ITMED_WAIT：等待所有分组死掉
CLOSING：两边同时尝试关闭
TIME_WAIT：另一边已初始化一个释放
LAST_ACK：等待所有分组死掉

Tomcat内部结构

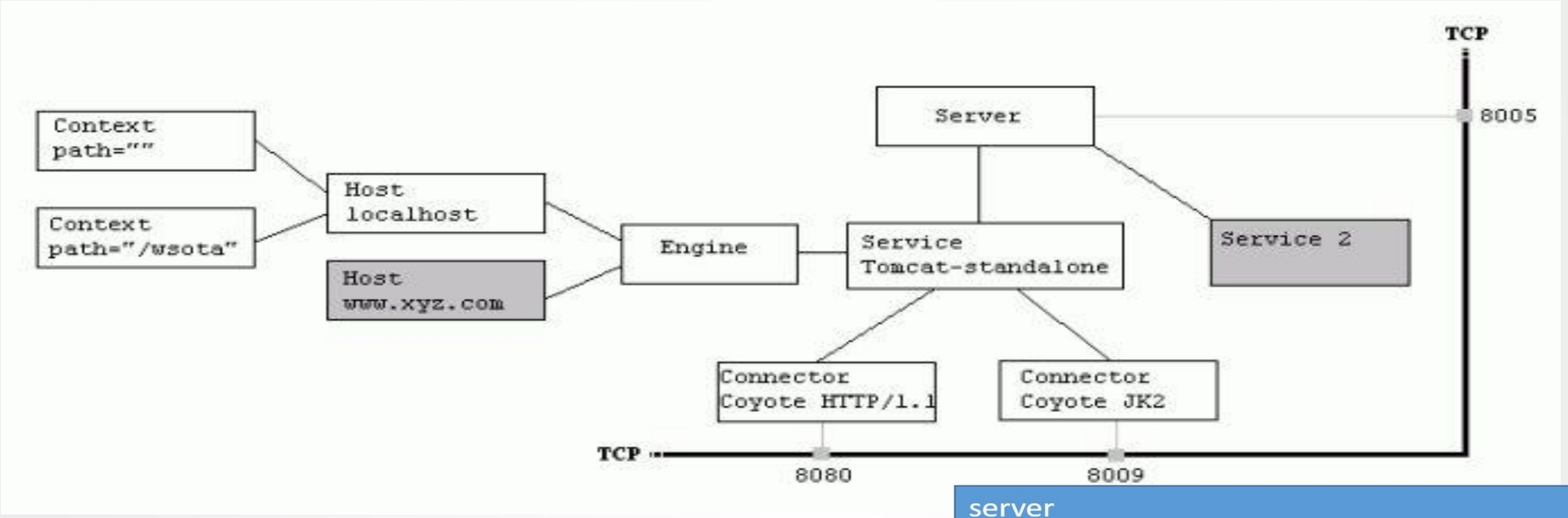
```
<Server>
        <Listener />
        <GlobaNamingResources>
        <Service>
                <Executor />
                <Connector />
                <Engine>
                        <Cluster />
                        <Realm />
                        <Realm />
                           <Host>
                                   <Valve />
                                   <Context />
                           </Host>
                </Engine>
        </Service>
</Server>
```

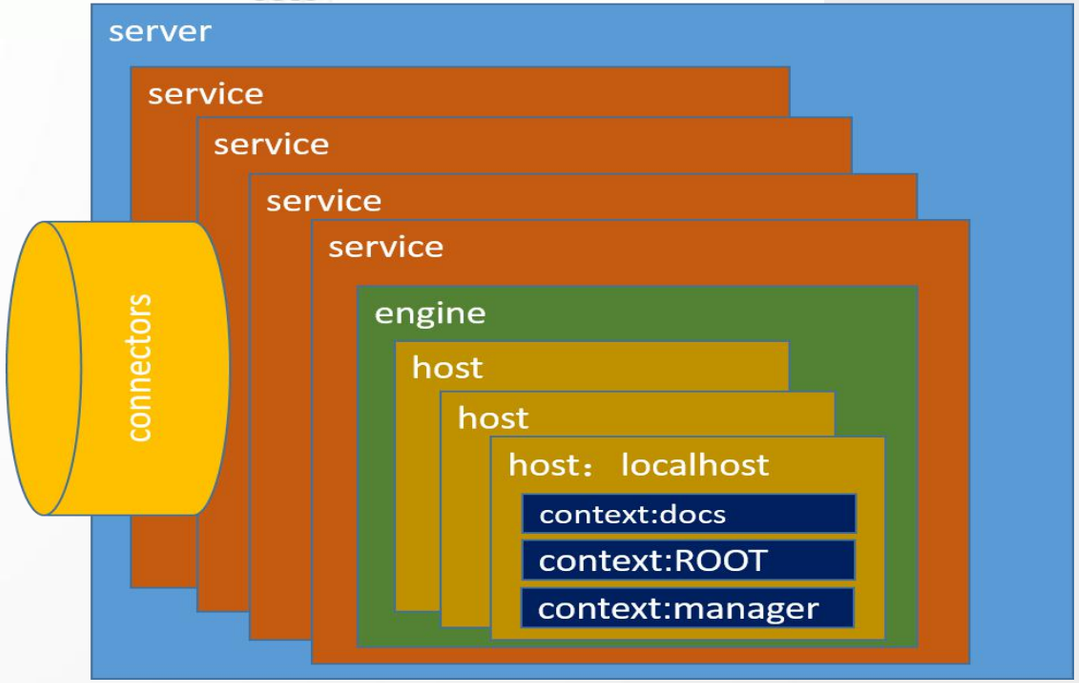
Tomcat内部结构



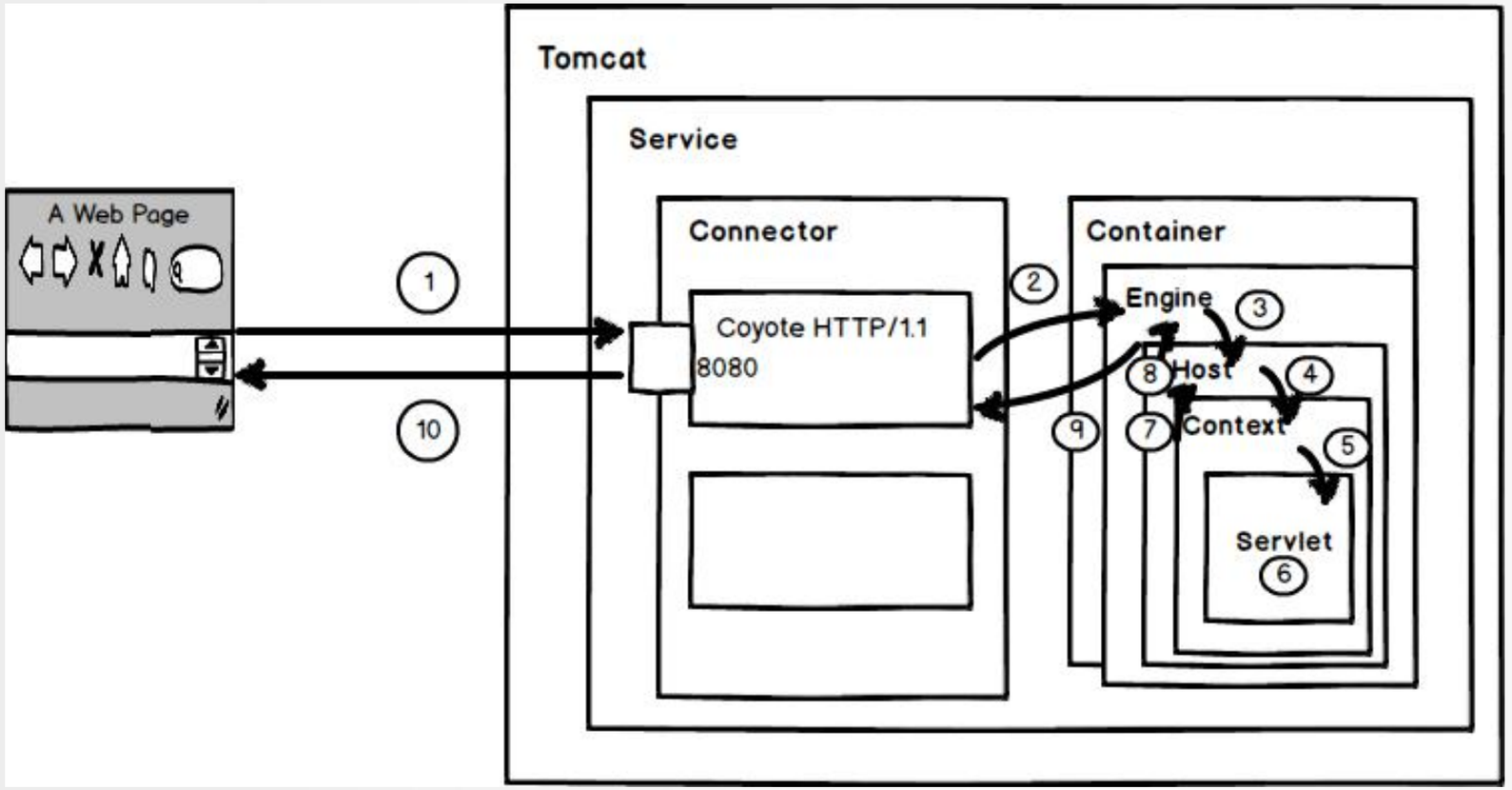
- server：指的是整个应用的上下文，也是最顶层的容器，tomcat中所有的东西都在这个server里边。
- service：指的是一个服务，主要的功能是把connector组件和engine组织起来，使得通过connector组件与整个容器通讯的应用可以使用engine提供的服务。
- engine：服务引擎，这个可以理解为一个真正的服务器，内部提供了多个虚拟主机对外服务。
- host：虚拟主机，每一个虚拟主机相当于一台服务器，并且内部可以部署多个应用，每个虚拟主机可以绑定一个域名，并指定多个别名。
- context：应用上下文，每一个webapp都有一个单独的context，也可以理解为每一个context代表一个webapp。
- connector：连接器组件，可以配置多个连接器支持多种协议，如http，APJ等

Tomcat内部结构


TCP
Server
8005
Context
path=""
Context
path="/wsota"
Host
localhost
Host
www.xyz.com
Engine
Service
Tomcat-standalone
Service 2
Connector
Coyote HTTP/1.1
Connector
Coyote JK2
TCP
8080
8009


server
service
service
service
service
engine
host
host
host： localhost
context:docs
context:ROOT
context:manager
connectors

性能的定义及表现



- **顶级组件**：位于配置层次的顶级，并且彼此间有着严格的对应关系。Server
- **服务类组件**：位于Server的下一级。Service
- **连接器组件**：连接客户端（可以是浏览器或Web服务器）请求至Servlet容器。http,https,ajp
- **容器类组件**：包含一组其它组件。Engine,Host,Context
- **被嵌套的组件**：位于一个容器当中，但不能包含其它组件。valve,loger,realm,loder,manager,…
- **集群类组件**： listen,cluster,…

性能的定义及表现

Tomcat常见组件：
•**服务器(server)**：Tomcat的一个实例，通常一个JVM只能包含一个Tomcat实例；因此，一台物理服务器上可以在启动多个JVM的情况下在每一个JVM中启动一个Tomcat实例，每个实例分属于一个独立的管理端口。这是一个顶级组件。
•**服务(service)**：一个服务组件通常包含一个引擎和与此引擎相关联的一个或多个连接器。给服务命名可以方便管理员在日志文件中识别不同服务产生的日志。一个server可以包含多个service组件，但通常情下只为一个service指派一个server。

连接器类组件：
•**连接器(connectors)**：负责连接客户端（可以是浏览器或Web服务器）请求至Servlet容器内的Web应用程序，通常指的是接收客户发来请求的位置及服务器端分配的端口。默认端口通常是HTTP协议的8080，管理员也可以根据自己的需要改变此端口。还可以支持HTTPS，默认HTTPS端口为8443。同时也支持AJP，即（A）一个引擎可以配置多个连接器，但这些连接器必须使用不同的端口。默认的连接器是基于HTTP/1.1的Coyote。同时，Tomcat也支持AJP、JServ和JK2连接器。
容器类组件：
•**引擎(Engine)**：引擎通是指处理请求的Servlet引擎组件，即**Catalina Servlet**引擎，它检查每一个请求的HTTP首部信息以辨别此请求应该发往哪个host或context，并将请求处理后的结果返回的相应的客户端。严格意义上来说，容器不必非得通过引擎来实现，它也可以是只是一个容器。如果Tomcat被配置成为独立服务器，默认引擎就是已经定义好的引擎。而如果Tomcat被配置为Apache Web服务器的提供Servlet功能的后端，默认引擎将被忽略，因为Web服务器自身就能确定将用户请求发往何处。一个引擎可以包含多个host组件。
•**主机(Host)**：主机组件类似于Apache中的虚拟主机，但在Tomcat中只支持基于FQDN的“虚拟主机”。一个引擎至少要包含一个主机组件。
•**上下文(Context)**：Context组件是最内层次的组件，它表示Web应用程序本身。配置一个Context最主要的是指定Web应用程序的根目录，以便Servlet容器能够将用户请求发往正确的位置。Context组件也可包含自定义的错误页，以实现在用户访问发生错误时提供友好的提示信息。
被嵌套类(nested)组件：
这类组件通常包含于容器类组件中以提供具有管理功能的服务，它们不能包含其它组件，但有些却可以由不同层次的容器各自配置。
•**阀门(Valve)**：用来拦截请求并在将其转至目标之前进行某种处理操作，类似于Servlet规范中定义的过滤器。Valve可以定义在任何容器类的组件中。Valve常被用来记录客户端请求、客户端IP地址和服务器等信息，这种处理技术通常被称作请求转储(request dumping)。请求转储valve记录请求客户端请求数据包中的HTTP首部信息和cookie信息文件中，响应转储valve则记录响应数据包首部信息和cookie信息至文件中。
•**日志记录器(Logger)**：用于记录组件内部的状态信息，可被用于除Context之外的任何容器中。日志记录的功能可被继承，因此，一个引擎级别的Logger将会记录引擎内部所有组件相关的信息，除非某内部组件定义了自己的Logger组件。
•**领域(Realm)**：用于用户的认证和授权；在配置一个应用程序时，管理员可以为每个资源或资源组定义角色及权限，而这些访问控制功能的生效需要通过Realm来实现。Realm的认证可以基于文本文件、数据库表、LDAP服务等来实现。Realm的效用会遍及整个引擎或顶级容器，因此，一个容器内的所有应用程序将共享用户资源。同时，Realm可以被其所在组件的子组件继承，也可以被子组件中定义的Realm所覆盖。

02

优化思路

OPTIMIZATION IDEAS

全局优化思考


优化
思路
网络优化
BIO、NIO、NIO2、APR，也就是阻塞与非阻塞
压缩gzip、超时配置，防止close_wait过多。
并发优化
最大线程数
最佳并发数。。。
底层优化
JVM优化
多实例（必须的）
操作系统优化

非阻塞、传输

网络优化

1、非阻塞，Tomcat8已经取消BIO
四种请求连接模型
HTTP/1.1
org.apache.coyote.http11.Http11Protocol 阻塞模式的连接协议
org.apache.coyote.http11.Http11NioProtocol 非阻塞模式的连接协议
org.apache.coyote.http11.Http11Nio2Protocol 非阻塞模式的连接协议
org.apache.coyote.http11.Http11AprProtocol – 本地连接协议

2、启用压缩，消耗CPU，减小网络传输大小

```
compression="on"
disableUploadTimeout="true"
compressionMinSize="2048"
compressableMimeType="text/html,text/xml,text/plain,text/css,text/javascript,application/javascript"
URIEncoding="utf-8"
```

非阻塞、传输

并发优化

连接数:maxConnections（排队数量）

处理线程：maxThreads（操作系统允许多少线程，线程多大会引起切换效能）

等候队列：acceptCount

非阻塞、传输

底层优化

JVM优化：固定堆内存，多线程并发收集，对象预留新生代，大对象进入老年代，启用内联

多实例：多个tomcat实例在一台机上

操作系统优化：网络参数，线程数，关闭IPV6，最大文件数
Linux服务器每进程不允许超过1000个线程，据说~~6、700线程服务器切换线程就慢下来~~
ps -eLf | grep java | wc –l
Linux线程栈大小是8M，ulimit –s设置

03

主体优化

SUBJECT OPTIMIZATION

示例

/etc/tomcat/tomcat.conf

```
JAVA_OPTS="-server –Xmx2048M –Xms2048G –Xmn768m -XX:TargetSurvivorRatio=90 -XX:PetenureSizeThreshold=1000000 -XX:MaxTenuringThreshold=30 –XX:+UseParallelGC –XX:+UseConcMarkSweepGC –XX:ParallelGCThreads=2”
```

/etc/tomcat/server.conf

```
<Executor name="tomcatThreadPool" namePrefix="catalina-exec-" maxThreads="500" //最大并发数，默认设置 200，一般建议在 500 ~ 800，根据硬件设施和业务来判断
minSpareThreads="100" //Tomcat 初始化时创建的线程数，默认设置 25
prestartminSpareThreads = "true"//在 Tomcat 初始化的时候就初始化 minSpareThreads 的参数值，如果不等于 true，minSpareThreads 的值就无效
maxQueueSize = "100"//最大的等待队列数，超过则拒绝请求 />
<Connector executor="tomcatThreadPool" port="8080"
protocol="org.apache.coyote.http11.Http11Nio2Protocol" //Tomcat 8 设置 nio2 更好，Tomcat 6、7设置nio更好：org.apache.coyote.http11.Http11NioProtocol
connectionTimeout="20000"
minSpareThreads="100"  maxSpareThreads="1000"最大处理连接数线程
minProcessors="100 “同时处理请求的最小数
maxProcessors= “1000” 同时处理请求的最大数
maxConnections="1000" redirectPort="8443"
enableLookups="false" //禁用DNS查询 acceptCount="100" //指定当所有可以使用的处理请求的线程数都被使用时，可以放到处理队列中的请求数，超过这个数的请求将不予处理，默认设置 100
maxPostSize="10485760" //以 FORM URL 参数方式的 POST 提交方式，限制提交最大的大小，默认是 2097152(2兆)，它使用的单位是字节。10485760 为 10M。如果要禁用限制，则可以设置为 -1。
compression="on" disableUploadTimeout="true" compressionMinSize="2048"
acceptorThreadCount="2" //用于接收连接的线程的数量，默认值是1。一般这个指需要改动的时候是因为该服务器是一个多核CPU，如果是多核 CPU 一般配置为 2.
compressableMimeType="text/html,text/xml,text/plain,text/css,text/javascript,application/javascript" URIEncoding="utf-8" keepAliveTimeout="0"
 />
```

示例

/etc/tomcat/server.conf

关闭shutdown端口：<Server port="-1" shutdown="SHUTDOWN">

关闭ajp连接：注释<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

取消访问日志Valve阀门

```
<Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log." suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
```

示例

ulimit优化

内核优化

linux 默认值 open files 和 max user processes 为 1024
#ulimit -n
1024
#ulimit –u
1024
问题描述： 说明 server 只允许同时打开 1024 个文件，处理 1024 个用户进程
使用ulimit -a 可以查看当前系统的所有限制值，使用ulimit -n 可以查看当前的最大打开文件数。
新装的linux 默认只有1024 ，当作负载较大的服务器时，很容易遇到error: too many open files 。
解决方法：
使用 ulimit –n 65535 可即时修改，但重启后就无效了。
有如下三种修改方式：
3. 在**/etc/security/limits.conf** 最后增加：
*** soft nofile 65535**
*** hard nofile 65535**
*** soft nproc 65535**
*** hard nproc 65535**

net.ipv4.tcp_syncookies = 1 开启SYN Cookies。当出现SYN等待队列溢出时，启用cookies来处理，可防范少量SYN攻击；
net.ipv4.tcp_tw_reuse = 1 开启重用。允许将TIME-WAIT sockets重新用于新的TCP连接，默认为0
net.ipv4.tcp_tw_recycle = 1 开启TCP连接中TIME-WAIT sockets的快速回收，默认为0，表示关闭。
net.ipv4.tcp_fin_timeout = 30 如果套接字由本端要求关闭，它决定了它保持在FIN-WAIT-2状态的时间。
net.ipv4.tcp_keepalive_time = 1200 当keepalive起用的时候，TCP发送keepalive消息的频度。缺省是2小时
net.ipv4.tcp_keepalive_intvl = 30
net.ipv4.tcp_keepalive_probes = 3 probe 3次(每次30秒)不成功,内核才彻底放弃。
tcp_keepalive_time = 7200 seconds (2 hours)
tcp_keepalive_probes = 9
tcp_keepalive_intvl = 75 seconds
net.ipv4.ip_local_port_range = 1024 65000 用于向外连接的端口范围。缺省情况下很小：32768到61000，改为1024到65000。
net.ipv4.tcp_max_syn_backlog = 8192 SYN队列的长度，默认为1024，加大队列长度为8192，可以容纳更多等待连接的网络连接数。
net.ipv4.netdev_max_backlog = 1000 表示进入包的最大设备队列，默认300，改大
net.core.tcp_max_tw_buckets = 5000 系统同时保持TIME_WAIT套接字的最大数量，如果超过这个数字，TIME_WAIT套接字将立刻被清除并打印警告信息。默认为180000，改为 5000。
另外可以参考优化内核配置：
/proc/sys/net/core/wmem_max 最大socket写buffer,可参考的优化值:873200
/proc/sys/net/core/rmem_max 最大socket读buffer,可参考的优化值:873200
/proc/sys/net/ipv4/tcp_wmem TCP写buffer,可参考的优化值: 8192 436600 873200
/proc/sys/net/ipv4/tcp_rmem TCP读buffer,可参考的优化值: 32768 436600 873200
/proc/sys/net/ipv4/tcp_mem
同样有3个值,意思是:配置单位为页，不是字节
net.ipv4.tcp_mem[0]:低于此值,TCP没有内存压力. 786432
net.ipv4.tcp_mem[1]:在此值下,进入内存压力阶段. 1048576
net.ipv4.tcp_mem[2]:高于此值,TCP拒绝分配socket. 1572864
/proc/sys/net/core/somaxconn 256
listen()的默认参数,挂起请求的最大数量.默认是128.对繁忙的服务器,增加该值有助于网络性能.
/proc/sys/net/core/optmem_max socket buffer的最大初始化值,默认10K.
/proc/sys/net/ipv4/tcp_retries2 TCP失败重传次数,默认值15.减少到5,以尽早释放内核资源.
net.core.somaxconn = 32768 socket监听（listen）的backlog上限，是socket的监听队列。比如nginx 定义NGX_LISTEN_BACKLOG默认到511

nginx优化

1. worker_processes 8;nginx 进程数，建议按照cpu 数目来指定，一般为它的倍数 (如,2个四核的cpu计为8)。
2. worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000 01000000 10000000;为每个进程分配cpu，上例中将8 个进程分配到8 个cpu，当然可以写多个，或者将一
个进程分配到多个cpu。
3. worker_rlimit_nofile 65535;这个指令是指当一个nginx 进程打开的最多文件描述符数目，理论值应该是最多打开文件数（ulimit -n）与nginx 进程数相除，但是nginx 分配请求并不是那么均匀，所以最好与ulimit -n 的值保持一致。
查看linux系统文件描述符的方法：
[root@web001 ~]# sysctl -a | grep fs.file
fs.file-max = 789972
fs.file-nr = 510 0 789972
4. use epoll; 使用epoll 的I/O 模型
5. worker_connections 65535;每个进程允许的最多连接数， 理论上每台nginx 服务器的最大连接数为worker_processes*worker_connections。
6. keepalive_timeout 60;keepalive 超时时间。
7. client_header_buffer_size 4k;客户端请求头部的缓冲区大小，这个可以根据你的系统分页大小来设置，一般一个请求头的大小不会超过1k，不过由于一般系统分页都要大于1k，所以这里设置为分页大小。
分页大小可以用命令**getconf PAGESIZE** 取得。
[root@web001 ~]# getconf PAGESIZE
4096
但也有client_header_buffer_size超过4k的情况，但是client_header_buffer_size该值必须设置为**“系统分页大小”的整倍数。**
8. open_file_cache max=65535 inactive=60s;这个将为打开文件指定缓存，默认是没有启用的，max 指定缓存数量，建议和打开文件数一致，inactive 是指经过多长时间文件没被请求后删除缓存。
9. open_file_cache_valid 80s;这个是指多长时间检查一次缓存的有效信息。
10. open_file_cache_min_uses 1;open_file_cache 指令中的inactive 参数时间内文件的最少使用次数，如果超过这个数字，文件描述符一直是在缓存中打开的，如上例，如果有一个文件在inactive 时间内一次没被使用，它将被移除。

示例

```
# This variable is used to figure out if config is loaded or not.
TOMCAT_CFG_LOADED="1"

# In new-style instances, if CATALINA_BASE isn't specified, it will
# be constructed by joining TOMCATS_BASE and NAME.
TOMCATS_BASE="/var/lib/tomcats/"

# Where your java installation lives
JAVA_HOME="/usr/lib/jvm/jre"

# Where your tomcat installation lives
CATALINA_HOME="/usr/share/tomcat"

# System-wide tmp
CATALINA_TMPDIR="/var/cache/tomcat/temp"

# You can pass some parameters to java here if you wish to
#JAVA_OPTS="-Xminf0.1 -Xmaxf0.3"
JAVA_OPTS="-server -Xms2048m -Xmx2048m -XX:PermSize=128m -XX:MaxNewSize=512m -XX:MaxPermSize=256m -Djava.awt.headless=true"

# Use JAVA_OPTS to set java.library.path for libtcnative.so
#JAVA_OPTS="-Djava.library.path=/usr/lib"

# You can change your tomcat locale here
#LANG="en_US"

# Run tomcat under the Java Security Manager
SECURITY_MANAGER="false"

# Time to wait in seconds, before killing process
# TODO(stingray): does nothing, fix.
# SHUTDOWN_WAIT="30"

# If you wish to further customize your tomcat environment,
# put your own definitions here
# (i.e. LD_LIBRARY_PATH for some jdbc drivers)
```

示例Server.xml

```
<Server port="-1" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.startup.VersionLoggerListener" />
  <!-- Security listener. Documentation at /docs/config/listeners.html
  <Listener className="org.apache.catalina.security.SecurityListener" />
  -->
  <!--APR library loader. Documentation at /docs/apr.html -->
  <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
  <!--Initialize Jasper prior to webapps are loaded. Documentation at /docs/jasper-howto.html -->
  <Listener className="org.apache.catalina.core.JasperListener" />
  <!-- Prevent memory leaks due to use of particular java/javax APIs-->
  <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
  <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

  <!-- Global JNDI resources
       Documentation at /docs/jndi-resources-howto.html
  -->
  <GlobalNamingResources>
    <!-- Editable user database that can also be used by
         UserDatabaseRealm to authenticate users
    -->
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>

  <!-- A "Service" is a collection of one or more "Connectors" that share
       a single "Container" Note:  A "Service" is not itself a "Container",
       so you may not define subcomponents such as "Valves" at this level.
       Documentation at /docs/config/service.html
   -->
  <Service name="Catalina">

    <!--The connectors can use a shared executor, you can define one or more named thread pools-->
    <!--
    <Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
        maxThreads="150" minSpareThreads="4"/>
    -->


    <!-- A "Connector" represents an endpoint by which requests are received
         and responses are returned. Documentation at :
         Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
         Java AJP  Connector: /docs/config/ajp.html
         APR (HTTP/AJP) Connector: /docs/apr.html
         Define a non-SSL HTTP/1.1 Connector on port 8080
    -->

    <Connector port="8080" protocol="org.apache.coyote.http11.Http11AprProtocol"
        connectionTimeout="2000" maxKeepAliveRequests="8000"
        useBodyEncodingForURI="true" maxConnections="1000"
        maxHttpHeaderSize ="8192"
        maxThreads ="500" minSpareThreads ="25"
        enableLookups ="false" acceptCount ="700"
        disableUploadTimeout ="true"  URIEncoding ="UTF-8"
        redirectPort="443" keepAliveTimeout="0"
    />
```

示例nginx.conf

```
user www www;
worker_processes 8;
worker_cpu_affinity 00000001 00000010 00000100 00001000 00010000 00100000
01000000;
error_log /www/log/nginx_error.log crit;
pid /usr/local/nginx/nginx.pid;
worker_rlimit_nofile 204800;
events {
                use epoll;
                worker_connections 204800;
}
http {
include mime.types;
default_type application/octet-stream;
charset utf-8;
server_names_hash_bucket_size 128;
client_header_buffer_size 2k;
large_client_header_buffers 4 4k;
client_max_body_size 8m;
sendfile on;
tcp_nopush on;
keepalive_timeout 60;
open_file_cache max=204800 inactive=20s;
open_file_cache_min_uses 1;
open_file_cache_valid 30s;
tcp_nodelay on;
gzip on;
gzip_min_length 1k;
gzip_buffers 4 16k;
gzip_http_version 1.0;
gzip_comp_level 2;
gzip_types text/plain application/x-javascript text/css application/xml;
gzip_vary on;
```

```
server
{
listen 8080;
server_name test.com;
index index.php index.htm;
root /www/html/;
location /status
{
stub_status on;
}
location ~ .*\.(php|php5)?$
{
fastcgi_pass 127.0.0.1:9000;
fastcgi_index index.php;
include fcgi.conf;
}
location ~ .*\.(gif|jpg|jpeg|png|bmp|swf|js|css)$
{
expires 30d;
}
log_format access '$remote_addr -- $remote_user [$time_local] "$request" '
'$status $body_bytes_sent "$http_referer" '
'"$http_user_agent" $http_x_forwarded_for';
access_log /www/log/access.log access;
}
}
```

04

集群优化

CLUSTER OPTIMIZATION

集群优化

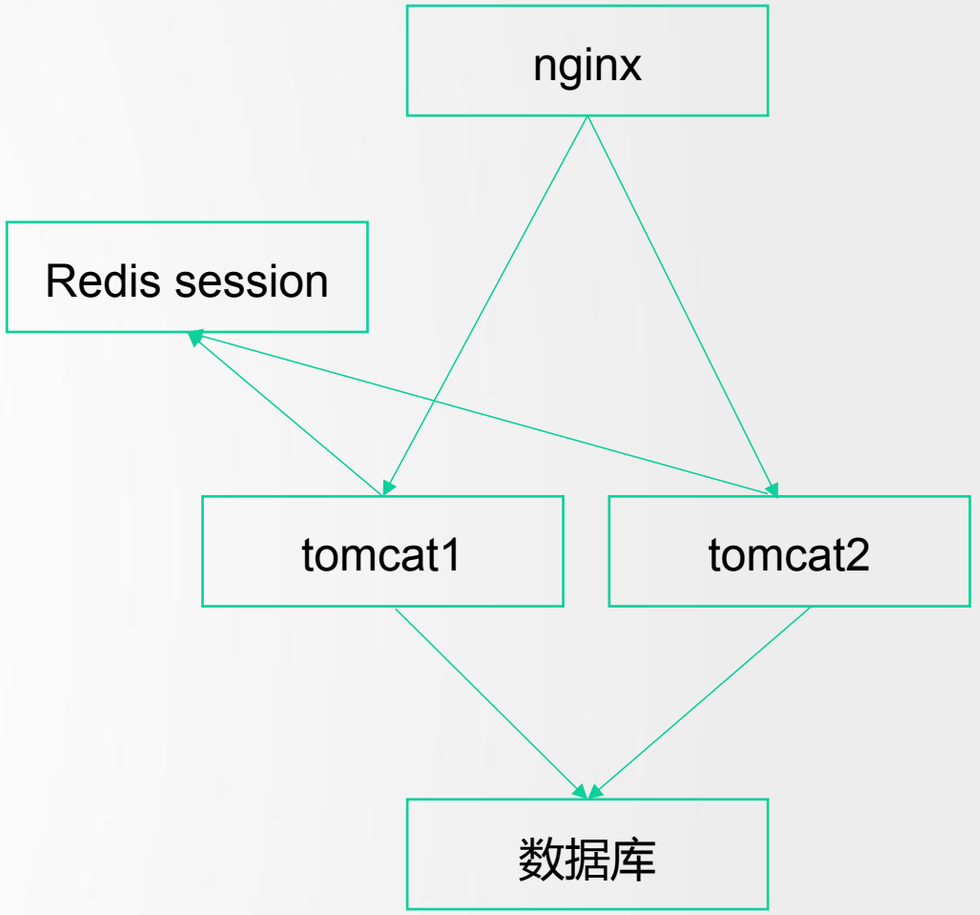
当线程数达到250以上，考虑群集部署

集群部署需要考虑的两个问题

Tomcat部署，session共享

Tomcat<4时，可用tomcat内部的集群session共享
否则采用redis方式集群

redis集群




```
<Valve className="tomcat.request.session.redis.RequestSessionHandlerValve"/>
<Manager className="tomcat.request.session.redis.RequestSessionManager"/>
```

TomcatRedisSessionManager-1.1 .jar
Jredis-2.8.0.jar
Commons-logging-1.2.jar
Commons-pool2-2.4.1.jar

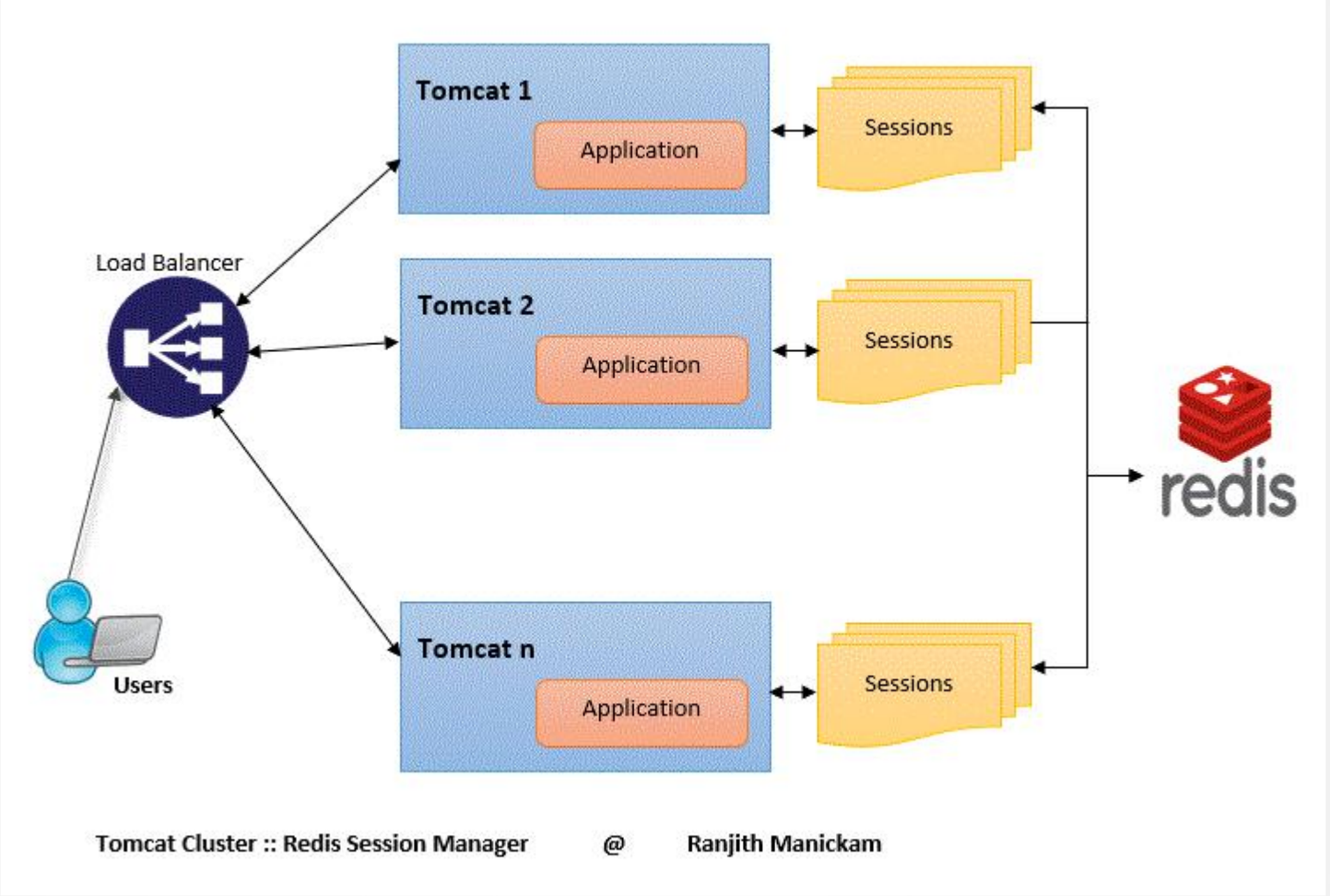
redis-data-cache.properties，放在conf.d目录

```
redis.hosts=127.0.0.1:6379
redis.cluster.enabled=false

#- redis database (default 0)
#redis.database=0

#- redis connection timeout (default 2000)
#redis.timeout=2000
```

Redis session共享



Ab测试

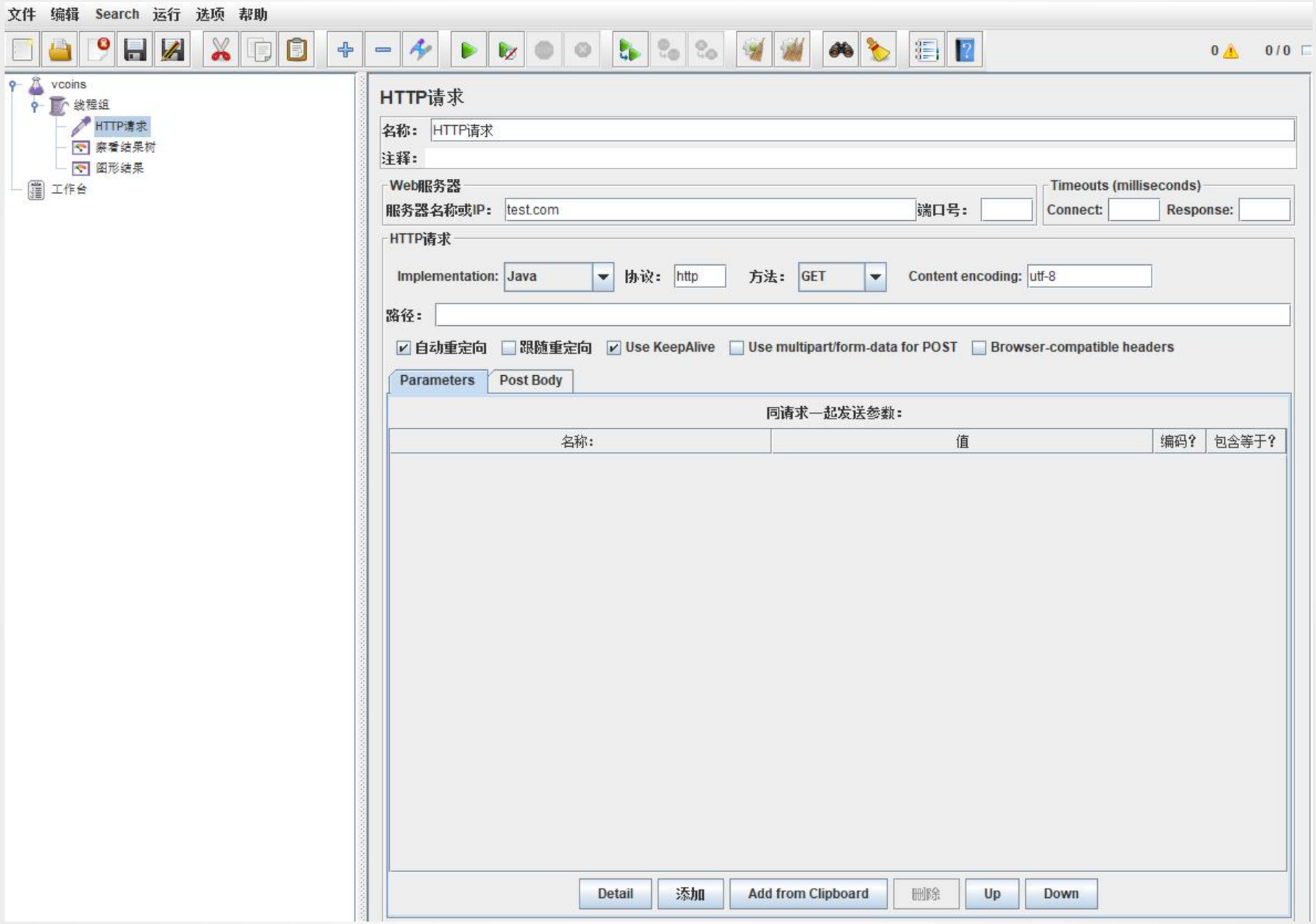
吞吐率（Requests per second）：总请求数 / 处理完成这些请求数所花费的时间

并发连接数（The number of concurrent users，Concurrency Level）：
一个用户可能同时会产生多个会话，也即连接数
用户平均请求等待时间（Time per request）：
处理完成所有请求数所花费的时间/（总请求数 / 并发用户数）

服务器平均请求等待时间（Time per request: across all concurrent requests）
计算公式：处理完成所有请求数所花费的时间 / 总请求数

ab –n 1000 –c 100 url/
如果只用到一个Cookie，那么只需键入命令：
ab －n 100 －C key＝value http://test.com/
如果需要多个Cookie，就直接设Header：
ab -n 100 -H “Cookie: Key1=Value1; Key2=Value2” http://test.com/

Jmeter测试


文件 编辑 Search 运行 选项 帮助
0 0 / 0
vcoins
线程组
HTTP请求
察看结果树
图形结果
工作台
HTTP请求
名称: HTTP请求
注释:
Web服务器
服务器名称或IP: test.com
端口号:
Timeouts (milliseconds)
Connect:
Response:
HTTP请求
Implementation: Java
协议: http
方法: GET
Content encoding: utf-8
路径:
自动重定向
跟随重定向
Use KeepAlive
Use multipart/form-data for POST
Browser-compatible headers
Parameters
Post Body
同请求一起发送参数:
名称:
值
编码?
包含等于?
Detail
添加
Add from Clipboard
删除
Up
Down