The Challenges Tomcat Faces in High Throughput Production Systems
Biography

• Huxing Zhang (张乎兴)
• Software Engineer @ Alibaba Middleware Group since 2014
• Previously worked for DeNA, Tokyo, Japan
• Apache Tomcat Committer since 2016
• Core tomcat maintainer @ Alibaba Inc.
Outline

• How we use Tomcat at Alibaba
• High throughput web applications
• The challenges
• Case studies
• Summary
• Q & A
Outline

• How we use Tomcat at Alibaba
  • High throughput web applications
  • The challenges
  • Case studies
• Summary
• Q & A
Tomcat in Alibaba

• Small trial since 2009
• Migration From JBoss to Tomcat
  • Jboss uses Tomcat as its servlet container
  • Officially started in 2013
• Becomes the dominant servlet container in 2015
  • Thousands of web applications
  • Hundreds of thousands of instances
• Status Quo
  • 95% tomcat, 5% other
  • Mainly 7.0.x (support JDK 6+)
  • Few 8.0.x
  • Emerging 8.5.x because of micro-service (e.g. spring-boot)
Deployment Standard

One web application per Tomcat instance

- **Startup script features**
  - Per startup log rotation for catalina.out
  - Smart auto-fix for incorrect JVM args
  - CATALINA_BASE preparation
  - Tomcat startup status detection

**Runtime Configuration (CATALINA_BASE)**
- AppName. war
- Tomcat conf
- Nginx conf

**Application Specific Configuration**
- Nginx conf
- Tomcat conf
- Startup Hooks
- JVM Args
- Health Check

**Standardized Configuration (CATALINA_HOME)**
- Nginx Conf
- Tomcat Conf
- Startup script

Deployment Platform

Deployments Platform

Rollback

Publish

Symbolic link
Monitoring, diagnostics and development tools

• Tomcat Monitor (RESTful service)
  • Metrics for OS, JVM, Tomcat, Middleware, and application
  • Classloading: locate which jar file a class is loaded from
  • Thread: per thread CPU usage, thread state, etc.
  • Connector stats: tomcat thread pool statistics

• Diagnostics on production environment
  • https://github.com/oldmanpushcart/greys-anatomy
  • Much easier to use than BTrace

• Development tools
  • Package free tomcat Eclipse plugin
Other Usages that worth mentioning

• Cluster Session Management: Taobao Session
  • Cookie + distributed key-value storage(Tair)

• Database Connection Pool: Druid
  • https://github.com/alibaba/druid
  • Performance, Monitoring, Extensibility

• Multi-tenancy
  • High density deployment for legacy web applications
  • Providing CPU/Memory isolation
  • Based on multi-tenant JVM
Outline

• How we use Tomcat at Alibaba

• High throughput web applications

• The challenges

• Case studies

• Summary

• Q & A
High throughput web application

- Taobao API Gateway for mobile devices
- Interacting with hundreds of backend services
- Handling millions of request per second
• Nginx + Tomcat on the same machine
  • Security
  • Rate Limiting
Outline

• How we use Tomcat at Alibaba
• High throughput web applications
• The challenges
• Case studies
• Summary
• Q & A
The challenges

- Connector Selection
- Configuration Tweaking
- NIO Connection & Concurrency Issues
  - Case study 1
- Security Issues
  - Case study 2
The challenges

• Connector Selection
• Configuration Tweaking
• NIO Connection & Concurrency Issues
  • Case study 1
• Security Issues
  • Case study 2
Architecture: BIO + Sync RPC

• One working thread per request
• Can not scale up to thousands of requests per second
• Low cpu utilization
Architecture: APR + Async RPC

- Blocking on reading request headers
- Non-blocking on waiting for the next request
- Cons:
  - Unexpected JVM crash
  - Maintenance cost: requires separate tcnative library
Architecture: NIO + Async RPC

- Non-blocking on reading request headers
- Non-blocking on waiting for next request
- Fully asynchronized
The challenges

• Connector Selection
• Configuration Tweaking
• NIO Connection & Concurrency Issues
  • Case study 1
• Security Issues
  • Case study 2
Tweaking Tomcat

• Connector configuration
  • Do not change if you really encounter some issue
  • Read the documentation carefully

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Default</th>
<th>Recommended</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConnectionTimeout</td>
<td>20s</td>
<td>Decrease</td>
<td></td>
</tr>
<tr>
<td>maxThreads</td>
<td>200</td>
<td>Increase</td>
<td></td>
</tr>
<tr>
<td>acceptCount (backlog)</td>
<td>100</td>
<td>Increase</td>
<td>min(acceptCount, /proc/sys/net/core/somaxconn)</td>
</tr>
<tr>
<td>processorCache</td>
<td>200</td>
<td>Increase</td>
<td>max(maxThreads, concurrent connection)</td>
</tr>
</tbody>
</table>

• access log disabled
  • Available on Nginx side
The challenges

- Connector Selection
- Configuration Tweaking
- NIO Connection & Concurrency Issues
  - Case study 1
- Security Issues
  - Case study 2
1. Acceptor accepts a socket
2. Acceptor retrieves a nioChannel object from cache
3. Poller thread registers nioChannel to its selector and selects for IO events
4. Poller assigns the nioChannel to one of the worker threads to process the request
5. SocketProcessor finishes processing the request and returns the nioChannel
Common NIO Issues

• Acceptor stops accepting new request
  • Connection leakage
  • Connection count leakage
  • nioChannel Cache pollution due to concurrency issues

• NPE during request processing
  • Caused by sendfile, default is on
Case study 1

- Tomcat 7.0.x
- NIO connector
- Throughput: 1000+ request/s
- Tomcat Poller thread throws ConcurrentModificationException

```
Exception in thread "http-nio-7001-ClientPoller-1" java.util.ConcurrentModificationException
at java.util.HashMap$HashIterator.nextEntry(HashMap.java:793)
   at java.util.HashMap$KeyIterator.next(HashMap.java:828)
   at java.util.Collections$UnmodifiableCollection$1.next(Collections.java:1010)
   at org.apache.tomcat.util.net.NioEndpoint$Poller.timeout(NioEndpoint.java:1421)
   at org.apache.tomcat.util.net.NioEndpoint$Poller.run(NioEndpoint.java:1215)
   at java.lang.Thread.run(Thread.java:662)
```  

- After a while Tomcat stops accepting new request
Troubleshooting

- Poller thread has its own selector
- Should not have concurrency issues
- Who is modifying the SelectionKeys?

```java
protected void timeout(int keyCount, boolean hasEvents) {
    //timeout
    Set<SelectionKey> keys = selector.keys();
    ... 
    for (Iterator<SelectionKey> iter = keys.iterator();
        iter.hasNext());
        SelectionKey key = iter.next();
    ... 
    }
    ...
    }
```
Digging into the JVM

• The only entrance: sun.nio.ch.EpollSelectorImpl

```java
protected void implRegister(SelectionKeyImpl ski) {
    ...
    keys.add(ski);
    + System.out.println("> " + Thread.currentThread().getName() + ", " + this);
}

protected void implDereg(SelectionKeyImpl ski) throws IOException {
    ...
    keys.remove(ski);
    + System.out.println("> " + Thread.currentThread().getName() + ", " + this);
    ...
}
```

Issue NOT reproducible!
Minimize the overhead

```java
protected void implRegister(SelectionKeyImpl ski) {
    ...
    keys.add(ski);
    + System.out.println(Thread.currentThread().getName() + " > " + this);
}

protected void implDereg(SelectionKeyImpl ski) throws IOException {
    ...
    keys.remove(ski);
    + System.out.println(Thread.currentThread().getName() + " > " + this);
    ...
}
```

• 2 poller thread are modifying the same selector instance!
JVM bug?

• Update to JDK 1.7 and the issue is remaining
• Setting poller thread number to 1 prevents the issue from being reproduced
• NOT a JVM bug!
• Where is the selection key being modified?

```java
@Override
public void run() {
    if (interestOps == OP_REGISTER) {
        try {
            socket.getIOChannel().register(
                socket.getPoller().getSelector(),
                SelectionKey.OP_READ, key);
        } catch (Exception x) {
            log.error("", x);
        }
    } else {
        // …
    }
} //end if

nioChannel registered to the wrong selector?
```

java.lang.Exception: Stack trace
  at sun.nio.ch.SelectorImpl.register(SelectorImpl.java:133)
  at java.nio.channels.spi.AbstractSelectableChannel.register(AbstractSelectableChannel.java:180)
  at org.apache.tomcat.util.net.NioEndpoint$PollerEvent.run(NioEndpoint.java:897)
  at org.apache.tomcat.util.net.NioEndpoint$Poller.events(NioEndpoint.java:1059)
  at org.apache.tomcat.util.net.NioEndpoint$Poller.run(NioEndpoint.java:1155)
  at java.lang.Thread.run(Thread.java:662)
socket.getPoller() != this?

```java
if (socket.getPoller() != this) {
    System.out.println("t: " + Thread.currentThread().getName() + ", s: " +
                      socket.getPoller().getName());
}
socket.getIOChannel().register(
    socket.getPoller().getSelector(), SelectionKey.OP_READ, key);
// ...
```

- It is confirmed that socket’s poller is modified after being pushed in event queue
nioChannel’s poller has been modified by another poller thread

Duplicate nioChannel Object?
Further suspicion

- Duplicate nioChannel objects in nioChannels cache

```java
protected boolean setSocketOptions(SocketChannel socket) {
    try {
        // ...
        NioChannel channel = nioChannels.poll();
        if (channel == null) {
            // ...
            channel = new NioChannel(socket, bufhandler);
        }
        // ...
        getPoller0().register(channel);
    } catch (Throwable t) {
        // ...
        return false;
    }
    return true;
}
```
Why duplicate entries?

- Two threads are concurrently offering the same nioChannel to cache
  - Poller#timeout due to Async Timeout
  - Async Complete by Async RPC callback thread

```java
public boolean processSocket(NioChannel socket, SocketStatus status,
   boolean dispatch, int where) {
    ...
    SocketProcessor sc = processorCache.poll();
    if (sc == null) sc = new SocketProcessor(socket, status, where);
    else sc.reset(socket, status, where);
    if (dispatch && getExecutor() != null) getExecutor().execute(sc);
    else sc.run();
    ...
}
```

thread: Thread[http-nio-7001-ClientPoller-1,5,main], where: Poller#processKey
thread: Thread[http-nio-7001-ClientPoller-1,5,main], where: Poller#timeout
thread: Thread[HSF-CallBack-11-thread-2,10,main], where: Http11NioProcessor#actionInternal
Recap

• Async Servlet timeout
  • 4s~15s
• Async RPC timeout
  • 3s~15s
• 2 SocketProcessor threads are trying to offer nioChannel object to cache concurrently
  • Async timeout
  • Async complete
Recap (cont.)

- The same nioChannel object has been offered twice
- And it is highly probable that the object has been offered twice consecutively!
Recap (cont.)

- The same.nioChannel has been assigned to two different poller event queue
  - Poller thread 0 is processing timeout on that object
  - Poller thread 1 is registering keys to that.nioChannel
Recap (cont.)

- Poller thread 0 throws Concurrent Modification Exception and dies
Recap (cont.)

- Acceptor does not know Poller thread 0 has already died
- Acceptor keeps assigning to Poller thread 0 until it reaches maxConnection(10000)
Recap (cont.)

- Acceptor stops accepting new request
Solution

• Tomcat has fixed a similar issue in 7.0.58
  • But not a complete fix

• It may also happen if handshake == -1
  • It means the socket has been processed by another thread

• Final fix
  • Allowing the nioChannel to be offered exactly once
  • Using CAS to avoid lock overhead

• Available from 7.0.64/8.0.25 onwards
The challenges

- Connector Selection
- Configuration Tweaking
- NIO Connection & Concurrency Issues
  - Case study 1
- Security Issues
  - Case study 2
Security Issues

• Disable manager and host-manager
• Disable jmx remote access
• Reverse Proxy can defend Tomcat against some attack
  • e.g. Slow HTTP body attack
  • Nginx will collect entire http body before handing off to tomcat
• Special Case to high throughput web applications
  • Cookie-based memory exhaustion
Case Study 2

- Symptom: MessageBytes object array has 13m instances, and takes up ~2G memory which cannot be garbage collected

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Objects</th>
<th>Shallow Heap</th>
<th>Retained Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.apache.tomcat.util.buf.ByteChunk</td>
<td>0/0/0/0/13220789</td>
<td>0/0/0/0/634507872</td>
<td>&gt;=643312600</td>
</tr>
<tr>
<td>org.apache.tomcat.util.buf.MessageBytes</td>
<td>0/3/0/0/13194614</td>
<td>0/3/0/0/581958872</td>
<td>&gt;=1801383536</td>
</tr>
<tr>
<td>org.apache.tomcat.util.buf.CharChunk</td>
<td>0/0/0/0/1320484</td>
<td>0/0/0/0/528194560</td>
<td>&gt;=569456590</td>
</tr>
<tr>
<td>char[]</td>
<td>0/0/0/0/2601380</td>
<td>0/0/0/0/283700656</td>
<td>&gt;=283700656</td>
</tr>
<tr>
<td>int[]</td>
<td>0/0/0/0/445144</td>
<td>0/0/0/0/165063736</td>
<td>&lt;=165063736</td>
</tr>
</tbody>
</table>

- Who is holding the reference?
Who is holding the reference?

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Shallow Heap</th>
<th>Retained Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>org.apache.tomcat.util.buf.MessageBytes @ 0x78ee5cde8</td>
<td>48</td>
<td>13752</td>
</tr>
<tr>
<td>decodedQuery org.apache.tomcat.util.http.Parameters @ 0x773899b8</td>
<td>72</td>
<td>16376</td>
</tr>
<tr>
<td>parameters org.apache.coyote.Request @ 0x78ed35ef0</td>
<td>160</td>
<td>1090416</td>
</tr>
<tr>
<td>req org.apache.coyote.RequestInfo @ 0x78ede1c60</td>
<td>99</td>
<td>280</td>
</tr>
<tr>
<td>[1780] java.lang.Object[2776] @ 0x79199ae98</td>
<td>11120</td>
<td>11120</td>
</tr>
<tr>
<td>elementData java.util.ArrayList @ 0x751028320</td>
<td>24</td>
<td>11144</td>
</tr>
<tr>
<td>processors org.apache.coyote.RequestGroupInfo @ 0x750df61c0</td>
<td>64</td>
<td>11256</td>
</tr>
<tr>
<td>global org.apache.coyote.http11.Http11NioProtocol$Http11Connection-Handler @ 0x7510</td>
<td>32</td>
<td>6312</td>
</tr>
<tr>
<td>handler org.apache.tomcat.util.net.NioEndpoint @ 0x7508a2d18</td>
<td>280</td>
<td>972992</td>
</tr>
<tr>
<td>this$0 org.apache.tomcat.util.net.NioEndpoint$Poller @ 0x762aeed08</td>
<td>48</td>
<td>120</td>
</tr>
<tr>
<td>&lt;Java Local&gt;, target java.lang.Thread @ 0x762ac4f08 http-nio-7001-ClientPoll</td>
<td>104</td>
<td>504</td>
</tr>
<tr>
<td>this$0 org.apache.tomcat.util.net.NioEndpoint$Poller @ 0x762b519a0</td>
<td>48</td>
<td>144</td>
</tr>
<tr>
<td>&lt;Java Local&gt;, target java.lang.Thread @ 0x762aad408 http-nio-7001-ClientPo</td>
<td>104</td>
<td>296</td>
</tr>
<tr>
<td>this$0 org.apache.tomcat.util.net.NioEndpoint$Acceptor @ 0x762cacf8d</td>
<td>24</td>
<td>112</td>
</tr>
<tr>
<td>&lt;Java Local&gt;, target java.lang.Thread @ 0x762aad3a0 http-nio-7001-Acceptor</td>
<td>104</td>
<td>592</td>
</tr>
</tbody>
</table>

- Normally 1 request has only a few MessageBytes
- Why 2k requests has 13m MessageBytes?
ServerCookies

- One request has 2k server cookies
- What’s inside the cookies?
ServerCookies inside

- Dynamically double the cookie array if current size is exceeded

private ServerCookie addCookie() {
    if (cookieCount >= scookies.length) {
        ServerCookie scookiesTmp[] = new ServerCookie[2*cookieCount];
        System.arraycopy(scookes, 0, scookiesTmp, 0, cookieCount);
        scookes = scookiesTmp;
    }
    ServerCookie c = scookes[cookieCount];
    if (c == null) {
        c = new ServerCookie();
        scookes[cookieCount] = c;
    }
    cookieCount++;
    return c;
}

- Never gets shrunk even after recycling
- If the cookie string is small enough, it can produce a large number of server cookie objects
The malicious attack

- Cookies are considered as one of the header
  - maxHeaderCount cannot defend
- Only maxHttpHeader effectively limits the number of cookies.
- RFC6265 does not require cookie name and cookie value to be non-empty

<table>
<thead>
<tr>
<th>Cookie string</th>
<th>maxHttpHeader</th>
<th># of bytes a cookie takes</th>
<th># of server cookie object</th>
</tr>
</thead>
<tbody>
<tr>
<td>“a=b;a=b;a=b;...”</td>
<td>8k</td>
<td>4</td>
<td>2k</td>
</tr>
<tr>
<td>“a=;a=;a=;...”</td>
<td>8k</td>
<td>3</td>
<td>4k</td>
</tr>
<tr>
<td>“=;=;=;=;...”</td>
<td>8k</td>
<td>2</td>
<td>4k</td>
</tr>
</tbody>
</table>

4k cookies * 2k request = 8m cookies
Solution

• Add a maxCookieCount parameter to limit the max number of cookie allowed
• Shrink the server cookie object array if limit is exceed
• Fixed in
  • 9.0.x for 9.0.0.M10 onwards
  • 8.5.x for 8.5.5 onwards
  • 8.0.x for 8.0.37 onwards
  • 7.0.x for 7.0.71 onwards
  • 6.0.x for 6.0.46 onwards
Summary

• Asynchronize your web application to improve throughput
  • NIO is relatively STABLE for high throughput production use
  • Check your version for known issues!

• Troubleshooting concurrency issues can be very tricky
  • A good tool is really helpful

• Upgrade your tomcat version to get latest updates!
Q & A

• Contact:
  • huxing.zhx@alibaba-inc.com
  • huxing@apache.org
  • Facebook: https://www.facebook.com/huxing.zhang
  • LinkedIn: https://www.linkedin.com/in/huxing-zhang

• Thanks!