Connectors

Permalink to this page: https://cwiki.apache.org/confluence/x/yiklBg

Preface

Please see the Other Resources Link for other pages describing how they were able to link Tomcat with a connector. With luck, someone documented their experience in an environment which is similar to yours.

Here is a link to the Apache Tomcat Connectors (aka JK Connectors) project page. It contains more configuration and installation information.

Please note, jk2 is no longer supported. Please use mod_jk instead.

Questions

Answers

What is JK (or AJP)?

AJP is a wire protocol. It an optimized version of the HTTP protocol to allow a standalone web server such as Apache to talk to Tomcat. Historically, Apache has been much faster than Tomcat at serving static content. The idea is to let Apache serve the static content when possible, but proxy the request to Tomcat for Tomcat related content.

Which connector: mod_jk or mod_proxy?

- mod_jk is mature, stable and extremely flexible. It is under active development by members of the Tomcat community.
- mod_proxy_ajp is distributed with Apache httpd 2.2 and later. Note that the communication protocol used is AJP.
- mod_proxy_http is a cheap way to proxy without the hassles of configuring JK. If you don't need some of the features of mod_jk, this is a very simple alternative. Note that the communication protocol used is HTTP/1.1.
- mod_proxy_http2 uses HTTP/2 as communication protocol. I has support for both secure (h2) and cleartext (h2c) variants of HTTP/2. Both are understood by Tomcat 8.5 and later.

Here are some anecdotal comments from members of the Tomcat community:

> _ I have been using mod_jk for a very long time and I saw (at the time) only one reason to make the switch to mod_proxy_ajp: it is bundled with Apache and so you (likely) don't have to build the module yourself.
>
> That said, simple configurations are *way* more simple in mod_proxy_ajp than with mod_jk, although the (somewhat) recent addition of `JkWorkerProperty` and `JkMount` "extensions" do help quite a bit.
>
> mod_proxy_ajp can also be trivially swapped-out with mod_proxy_http just by changing the URLs in your `ProxyPass` and `ProxyPassReverse` directives to say http:// (or https://) instead of ajp://. This might help you if you need to switch protocols for debugging purposes or if you suddenly need switch to HTTPS to secure the traffic without any external configuration (e.g. stunnel or VPN). (See AJP with stunnel.)
>
> mod_proxy also supports `ProxyPassMatch` directive which lets you use regular expressions in your URL mappings, which mod_jk's `JkMount` does not (though you *can* use `<LocationMatch>` along with `SetHandler` in order to achieve the same result, it's a cleaner configuration with mod_proxy).
>
> That said, I have found that mod_jk supports more complicated configurations where I have struggled to get mod_proxy_ajp to do the same. Specifically, overlapping URL spaces that must be mapped to separate workers. Technically speaking, I suppose you could use lots of `ProxyPassMatch` directives and/or have a complex regular expression to direct the various URLs, but again you end up with a rather messy configuration that way. Messy configurations are a maintenance risk as well as at risk of becoming "arcane knowledge" that nobody actually understands and so they are afraid to modify it for any reason.
>
> Generally, mod_jk will get fixed faster than mod_proxy_ajp due to its independent release cycle: the httpd folks might have a fix for a problem but it doesn't get released for a while due to testing of other components, etc. At this point, mod_proxy_ajp has (IMHO) reached a point of stability that this is less of an issue than it used to be.
>
> At this stage, there is no reason for me to move any of my projects from mod_jk to mod_proxy_ajp but if I were starting from scratch, I might choose mod_proxy_ajp solely due to its binary availability and simple configuration. If the configuration became complicated to the extent that switching to mod_jk were a good option, then I'd move.

> As for performance, I have no data on that one way or another. I would suspect that mod_jk has a slight performance advantage because it has been especially designed for the purpose rather than mod_proxy_ajp which must support the mod_proxy API and might have a bit more plumbing code to accomplish that. I would be surprised if you could detect any performance difference between the two if you were to test them both faithfully and with compatible configurations. If anyone has relative performance data between mod_jk and mod_proxy_ajp, I'd be happy to read it._
> (Source: https://tomcat.markmail.org/message/u5v4aiejluzy7tde)

What about mod_jserv, mod_jk2, mod_webapp (aka warp)?

All of these connectors have been abandoned long ago. Do not use any of them.

mod_jk2 sounds like it could be an updated version of mod_jk, it is not: it was an aborted effort whose features have been re-incorporated into mod_jk.

For historical purposes, and emphasis:

- mod_jserv is unsupported and will not be supported in Tomcat 5 onward. mod_jserv was the original connector which supported the ajp protocol. **Do not use mod_jserv.**
- Stay away from mod_webapp, aka warp. It is deprecated and unsupported due to lack of developer interest and there are better options such as jk and mod_proxy. It WILL NOT run on windows. **Do not use mod_webapp or warp.**
- jk2 is a refactoring of mod_jk and uses the Apache Portable Runtime (apr). But due to lack of developer interest, it is unsupported. **Do not use mod_jk2.**

Why should I integrate Apache HTTP Server with Apache Tomcat? (or not)

There are many reasons to integrate Tomcat with Apache. And there are reasons why it should not be done too. Needless to say, everyone will disagree with the opinions here. With the performance of Tomcat 5 and 6, performance reasons become harder to justify. So here are the issues to discuss in integrating vs not.

- Clustering. By using Apache as a front end you can let Apache act as a front door to your content to multiple Tomcat instances. If one of your Tomcats fails, Apache ignores it and your Sysadmin can sleep through the night. This point could be ignored if you use a hardware loadbalancer and Tomcat's clustering capabilities.
- Clustering/Security. You can also use Apache as a front door to different Tomcats for different URL namespaces (/app1/, /app2/, /app3/, or virtual hosts). The Tomcats can then be each in a protected area and from a security point of view, you only need to worry about the Apache server. Essentially, Apache becomes a smart proxy server.
- Security. This topic can sway one either way. Java has the security manager while Apache has a larger mindshare and more tricks with respect to security. I won't go into this in more detail, but let Google be your friend. Depending on your scenario, one might be better than the other. But also keep in mind, if you run Apache with Tomcat - you have two systems to defend, not one.
- Add-ons. Adding on CGI, perl, PHP is very natural to Apache. Its slower and more of a kludge for Tomcat. Apache also has hundreds of modules that can be plugged in at will. Tomcat can have this ability, but the code hasn't been written yet.
- Decorators! With Apache in front of Tomcat, you can perform any number of decorators that Tomcat doesn't support or doesn't have the immediate code support. For example, mod_headers, mod_rewrite, and mod_alias could be written for Tomcat, but why reinvent the wheel when Apache has done it so well?
- Speed. Apache is faster at serving static content than Tomcat. But unless you have a high traffic site, this point is useless. But in some scenarios, tomcat can be faster than Apache httpd. So benchmark YOUR site. **Tomcat can perform at httpd speeds when using the proper connector (APR with sendFile enabled). Speed should not be considered a factor when choosing between Apache httpd and Tomcat**
- Socket handling/system stability. Apache has better socket handling with respect to error conditions than Tomcat. The main reason is Tomcat must perform all its socket handling via the JVM which needs to be cross platform. The problem is socket optimization is a platform specific ordeal. Most of the time the java code is fine, but when you are also bombarded with dropped connections, invalid packets, invalid requests from invalid IP's, Apache does a better job at dropping these error conditions than JVM based program. (YMMV)

At boot, is order of start up (Apache HTTP Server vs Apache Tomcat) important?

No. This way either Apache HTTPd or Apache Tomcat can be restarted at any time independently of one another.

Is there any way to control the content of automatically generated mod_jk.conf-auto? I need my own specific commands added to it.

There really is no need to. Just copy the automatically generated mod_jk.conf-auto and edit it manually to your preference. None of production tomcat installations really use mod_jk.conf-auto as it is.

How do I bind to a specific IP address?

Each Connector element allows an `address` property. See the HTTP Connector docs or the AJP Connector docs.

Where can I download a binary distribution of my connector?

You cannot: you need to download the source and compile it for your platform. The source distributions are available from the standard location. Note that JPackage.org has RPM distributions for the connectors as well as tomcat itself: JPackage.org

I'm having strange UTF-8 issues with my request parameters.

See Character Encoding

How do I configure Apache Tomcat connectors for a heavy load site?

See Performance and Monitoring