Runs a Grails application as a JAR file with an embedded Tomcat or Jetty server

Grails Standalone Plugin - Reference Documentation

Authors: Burt Beckwith

Version: 9.0.0.M4

Table of Contents

1 Introduction to the Standalone Plugin

2 Running the application

1 Introduction to the Standalone Plugin

The Standalone plugin builds a runnable JAR file with an embedded war built from your application and an embedded Tomcat 9 or Jetty 7 instance. This allows you to build a single archive that can be run on any computer with Java 8 or higher by running `java -jar standalone.jar`.

Release History

- April 20, 2016
  - 9.0.0.M4 release
- April 20, 2016
  - 8.0.33 release
- December 3, 2014
  - 1.3 release
- October 30, 2013
  - 1.2.3 release
- October 29, 2013
  - 1.2.2 release
- May 10, 2013
  - 1.2.1 release
- May 10, 2013
  - 1.2 release
- August 16, 2012
  - 1.1.1 release
- August 6, 2012
  - 1.1 release
- August 4, 2012
  - 1.0.1 release
- July 8, 2011
  - initial 1.0 release

2 Running the application

Building the jar

The first step is to run the build-standalone script, e.g.

```
grails prod build-standalone
```

or

```
grails -Dgrails.env=demo build-standalone our_cool_demo.jar
```

If you pass the --jetty flag the embedded server will be Jetty instead of the default Tomcat, for example

```
grails prod build-standalone --jetty
```

or

```
grails -Dgrails.env=demo build-standalone our_cool_demo.jar
--jetty
```

You can change the default embedded server to Jetty by adding this to BuildConfig.groovy:

```
grails.plugin.standalone.useJetty = true
```

If you've set Jetty to the default, you can force a Tomcat build later with the --tomcat flag:

```
grails build-standalone --tomcat
```

Build Configuration

There are several configuration settings that you can use to change how the jar is built; add any of these to BuildConfig.groovy to override the defaults:

Property	Default Value	Meaning
grails.plugin.standalone.ecjDependency	org.eclipse.jdt.core.compiler:ecj:4.5.1	the dependency config for the ECJ jar
grails.plugin.standalone.extraDependencies	*none*	extra dependency jars to include in the standalone jar
grails.plugin.standalone.ivyLogLevel	`warn`	the Ivy log level
grails.plugin.standalone.jettyServletApiDependency	javax.servlet:servlet-api:2.5 or javax.servlet:javax.servlet-api:3.1.0	the Jetty servlet API jar dependency
grails.plugin.standalone.jettyVersion	`7.6.0.v20120127`	the version of Jetty to use
grails.plugin.standalone.tomcatVersion	`9.0.0.M4`	the version of Tomcat to use
grails.plugin.standalone.tomcatDependencies	`['tomcat-annotations-api', 'tomcat-api', 'tomcat-catalina-ant', 'tomcat-catalina', 'tomcat-coyote', 'tomcat-juli', 'tomcat-servlet-api', 'tomcat-util']`	the Tomcat jars to use
grails.plugin.standalone.tomcatEmbedDependencies	`['tomcat-embed-core', 'tomcat-embed-el', 'tomcat-embed-jasper', 'tomcat-embed-logging-juli', 'tomcat-embed-logging-log4j', 'tomcat-embed-websocket']`	the Tomcat embed jars to use
grails.plugin.standalone.mainClass	`grails.plugin.standalone.JettyLauncher` or `grails.plugin.standalone.Launcher`	Optionally specify a custom main class to include in the MANIFEST.MF. Note that you will then be required to call either grails.plugin.standalone.JettyLauncher or grails.plugin.standalone.Launcher yourself

Running the server

As long as the target machine has Java 8 or higher available, all you need to do next is run

```
java -jar /path/to/jar_name.jar
```

There are several arguments you can pass to customize how the application runs, using `name=value` syntax:

1. `context`, the context name; if not specified it defaults to "" (the "root" context)
2. `host`, the host name; if not specified it defaults to "localhost"
3. `port`, the HTTP port; if not specified it defaults to 8080
4. `httpsPort`, the HTTPS port; there is no default for this, but if specified you can also specify the keystore path and password
5. `keystorePath` or `javax.net.ssl.keyStore`, the SSL keystore path; if not specified a temporary keystore will be generated
6. `keystorePassword` or `javax.net.ssl.keyStorePassword`, the SSL keystore password; required if an existing keystore path is specified
7. `truststorePath` or `javax.net.ssl.trustStore`, the SSL truststore path
8. `trustStorePassword` or `javax.net.ssl.trustStorePassword`, the SSL truststore password; required if an existing truststore path is specified
9. `enableClientAuth`, whether to enable client auth, defaults to "want"; values can be one of (case-insensitive) 'true', 'yes', 'require', or 'required' for REQUIRED, 'optional' or 'want' for OPTIONAL, 'optionalNoCA' or 'optional_no_ca' for OPTIONAL_NO_CA, or 'false', 'no', or 'none' for NONE
10. `workDir`, the working directory where the war file is extracted, defaults to the system temp directory
11. `enableCompression`, whether to enable compression (Tomcat only)
12. `compressableMimeTypes`, a comma separated list of MIME types for which HTTP compression may be used; defaults to the Tomcat defaults, `"text/html,text/xml,text/plain"`
13. `sessionTimeout`, the session timeout in minutes; defaults to 30
14. `nio` or `tomcat.nio`, whether to use NIO; defaults to true
15. `serverName`, a specific value to use as HTTP Server Header, by default tomcat will use Apache-Coyote/1.1 if none set at application level (Tomcat only)
16. `enableProxySupport`, enables support for X-Forwarded headers by adding a pre-configured RemoteIpValve, defaults to false (Tomcat only)
17. `certificateFile`, the path to the OpenSSL certificate file, no default
18. `certificateKeyFile`, the path to the OpenSSL certificate private key file, no default
19. `certificateKeyPassword`, the password for the OpenSSL certificate private key file, no default

In addition, if you specify a value that is the name of a system property (e.g. 'home.dir'), the system property value will be used.

For example running

```
java -jar /path/to/jar_name.jar
```

will start a server at `http://localhost:8080/` and

```
java -jar /path/to/jar_name.jar context=cool_demo port=9000
```

will start a server at `http://localhost:9000/cool_demo`

```
java -jar /path/to/jar_name.jar context=cool_demo port=8080
httpsPort=8443
```

will start a server at `http://localhost:8080/cool_demo` and will also support SSL at `https://localhost:8443/cool_demo`

HTTP/2

As of the 9.0.0.M4 release of the plugin it is possible to run your application with HTTP/2 enabled (only with Tomcat currently). You must use Java 8 or higher (this is a general requirement of Tomcat 9) and SSL, and because Java 8 does not support ALPN you must also use the Tomcat Native Library and APR. Much of this is configured in the plugin, but you will have to install the Tomcat Native Library binaries and add them to your LD_LIBRARY_PATH (or use a `-Djava.library.path` commandline arg) - see the documentation for more info.

In addition to configuring the Tomcat Native Library, you must create an OpenSSL certificate and private key file. The keystore that is auto-generated for you when you specify `-https` and do not specify the location of your own cannot be used for HTTP/2. See the OpenSSL docs for detailed information about the process. For local testing you should be able to create a self-signed certificate with

```
openssl req -x509 -newkey rsa:2048 -keyout private-key.pem
-out cert.pem -days 365
```

Then specify the paths to the certificate file and private key file, along with the password you used when generating them, as commandline args, e.g.

bc. java -jar /path/to/jar_name.jar port=8080 httpsPort=8443 certificateKeyFile=/path/to/private-key.pem certificateFile=/path/to/cert.pem certificateKeyPassword=…