jsimone / webapp-runner

No description, website, or topics provided.

382 commits | 8 branches | 68 releases | 19 contributors

Branch: master | New pull request | Find file | Clone or download

Fetching latest commit…

.mvn/wrapper	Updated maven version	Nov 26, 2017
assembly	[maven-release-plugin] prepare for next development iteration	May 12, 2018
etc	Configure travis to run integration tests only with heroku api key	Jan 15, 2018
main	[maven-release-plugin] prepare for next development iteration	May 12, 2018
memcached	[maven-release-plugin] prepare for next development iteration	May 12, 2018
redis	[maven-release-plugin] prepare for next development iteration	May 12, 2018
src/main/resources/license	Refactored into separate modules to enable a no-dep artifact.	Feb 10, 2017
.gitignore	Upgrade Tomcat to 8.5 and all other dependencies.	Sep 25, 2016
.travis.yml	Configure travis to run integration tests only with heroku api key	Jan 15, 2018
README.md	Replace tomcat-redis-session with Redisson	May 12, 2018
mvnw	Refactored into separate modules to enable a no-dep artifact.	Feb 10, 2017
mvnw.cmd	Refactored into separate modules to enable a no-dep artifact.	Feb 10, 2017
pom.xml	[maven-release-plugin] prepare for next development iteration	May 12, 2018
release.sh	Update release script	Jan 15, 2018

README.md

Webapp Runner build passing maven central 9.0.8.0

Webapp runner is designed to allow you to launch an exploded or compressed war that is on your filesystem into a tomcat container with a simple `java -jar` command. It supports the following version of Tomcat:

- Tomcat 7.x: tomcat7 branch
- Tomcat 8.0: tomcat8.0 branch
- Tomcat 8.5: master branch
- Tomcat 9.x: tomcat9 branch

Usage

Clone and Build

```
git clone https://github.com/jsimone/webapp-runner.git
mvn package
```

Execute

```
java -jar target/webapp-runner.jar path/to/my/project
```

or

```
java -jar target/webapp-runner.jar myProject.war
```

Help

```
java -jar target/webapp-runner.jar --help
```

Prints out all arguments accepted

Using with Maven in your project

You can use the Maven dependency plugin to download webapp-runner as part of your build. This will eliminate the need for any external dependencies other than those specified in your build to run your application.

pom.xml

Add the following to your pom.xml:

```
<build>
...
  <plugins>
      <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-dependency-plugin</artifactId>
          <version>2.3</version>
          <executions>
              <execution>
                  <phase>package</phase>
                  <goals><goal>copy</goal></goals>
                  <configuration>
                      <artifactItems>
                          <artifactItem>
                              <groupId>com.github.jsimone</groupId>
                              <artifactId>webapp-runner</artifactId>
                              <version>${webapp-runner.version</version>
                              <destFileName>webapp-runner.jar</destFileName>
                          </artifactItem>
                      </artifactItems>
                  </configuration>
              </execution>
          </executions>
      </plugin>
  </plugins>
...
</build>
```

Excluding Memcached and Redis libraries

Webapp-runner bundles Memcached and Redis client libraries into it's fat-jar package. These libraries can cause conflicts with similar libraries in your application. This frequently manifests itself as a `java.lang.NoSuchMethodError` .

If you do not require these client libraries (because you are storing session data in some other way), You can exclude them by using `webapp-runner-main` :

```
<dependency>
  <groupId>com.github.jsimone</groupId>
  <artifactId>webapp-runner-main</artifactId>
  <version>${webapp-runner.version</version>
```

```
</dependency>
```

In most cases, this `groupId` and `artifactId` can be substituted for `com.github.jsimone:webapp-runner`.

Launching

Now when you run `maven package` webapp runner will be downloaded for you. You can then launch your application with:

```
 $ java -jar target/dependency/webapp-runner.jar target/<appname>.war
```

Store your sessions in memcache

In versions 7.0.29.1 and newer support for a session manager that stores sessions in memcache is built in.

To use it add `--session-store memcache` to your startup command:

```
$ java -jar target/dependency/webapp-runner.jar --session-store memcache target/<appname>.war
```

Then make sure that three environment variables are available for configuration: MEMCACHE_SERVERS, MEMCACHE_USERNAME, MEMCACHE_PASSWORD

Store your sessions in redis

In versions 7.0.29.1 and newer support for a session manager that stores sessions in redis is built in.

To use it add `--session-store redis` to your startup command:

```
$ java -jar target/dependency/webapp-runner.jar --session-store redis target/<appname>.war
```

Then make sure that Redis environment variable is available for configuration: REDIS_URL

Using Tomcat behind a reverse proxy server

If you are using webapp-runner behind a proxy server, you can set the proxy base url within tomcat:

```
$ java -jar target/dependency/webapp-runner.jar --proxy-base-url http://example.com
target/<appname>.war
```

If you pass an HTTPS base url, e.g. https://example.com, secure flag will be automatically added to session cookies. This indicates to the browser that cookies should only be sent over a secure protocol.

Running your application in Eclipse

Since your application will just be a standard webapp you can still use WTP and the traditional Tomcat integration points to run your application within Eclipse. However the containerless nature of webapp runner allows you to run from within Eclipse in a simpler way.

Start by importing your project into Eclipse. It is best to import it as an existing Maven project using the m2eclipse plugin.

Make your application dependent on webapp runner

Add the following dependency to your pom.xml:

```
<dependency>
  <groupId>com.github.jsimone</groupId>
  <artifactId>webapp-runner</artifactId>
  <version>8.5.11.3</version>
  <scope>provided</scope>
</dependency>
```

This will cause Eclipse to include webapp-runner on the classpath of your project so that it can be used for launching. It won't affect the final artifact built for your application.

Create a launch configuration

1. Right-click on your project and choose 'Debug As -> Debug Configurations...'
2. From the 'Debug Configuration' window create a new 'Java Application' launch configuration by double-clicking on 'Java Application' in the left hand list or right-clicking on it and selecting 'New'
3. Give your launch configuration a sensible name. Then enter the name of your project in the 'Project' box
4. Enter 'webapp.runner.launch.Main' in the 'Main Class' box
5. Click on the 'Arguments' tab and enter './src/main/webapp' in the 'Program Arguments' box
6. Click 'Apply' and then 'Run'

Your application should start and you should see the log output in the Eclipse console. Since you have a debugger attached to your application you'll now see changes to your code get automatically loaded without restarting and can set breakpoints.

You can stop the application from the red square in the console pane or from the debug perspective. It can be restarted by right-clicking on the project and choosing your new launch configuration from the 'Debug As' menu or from the debug menu in the Eclipse toolbar (the icon with the little bug).

Maven Central

Note: webapp runner is now available in Maven Central. The version scheme has also changed to match the version of Tomcat that it relies on. The format is `<tomcat version>.<minor webapp runner version>` . Versions 0.0.1 to 0.0.7 are still available at http://jsimone.github.com/webapp-runner/repository.

Options

```
$ java -jar webapp-runner.jar --help
The specified path "src/main/webapp" does not exist.
Usage: <main class> [options]
  Options:
    --basic-auth-pw
       Password to be used with basic auth. Defaults to BASIC_AUTH_PW env
       variable.
    --basic-auth-user
       Username to be used with basic auth. Defaults to BASIC_AUTH_USER env
       variable.
    --bind-on-init
       Controls when the socket used by the connector is bound. By default it is
       bound when the connector is initiated and unbound when the connector is
       destroyed., default value: true
       Default: true
    --compressable-mime-types
       Comma delimited list of mime types that will be compressed when using
       GZIP compression.
       Default:
text/html,text/xml,text/plain,text/css,application/json,application/xml,text/javascript,application/javascr

    --context-xml
       The path to the context xml to use.
    --enable-basic-auth
       Secure the app with basic auth. Use with --basic-auth-user and
       --basic-auth-pw or --tomcat-users-location
       Default: false
    --enable-client-auth
       Specify -Djavax.net.ssl.keyStore and -Djavax.net.ssl.keyStorePassword in
       JAVA_OPTS
       Default: false
    --enable-compression
       Enable GZIP compression on responses
       Default: false
    --enable-naming
       Enables JNDI naming
       Default: false
    --enable-ssl
```

```
       Specify -Djavax.net.ssl.keyStore, -Djavax.net.ssl.keyStorePassword,
       -Djavax.net.ssl.trustStore and -Djavax.net.ssl.trustStorePassword in JAVA_OPTS. Note: should not
be
       used if a reverse proxy is terminating SSL for you (such as on Heroku)
       Default: false
    --expand-war-file
       Expand the war file and set it as source
       Default: true
    --expanded-dir-name
       The name of the directory the WAR file will be expanded into.
       Default: expanded
    --help

       Default: false
    --max-threads
       Set the maximum number of worker threads
       Default: 0
    --path
       The context path
       Default: <empty string>
    --port
       The port that the server will accept http requests on.
       Default: 8080
    --proxy-base-url
       Set proxy URL if tomcat is running behind reverse proxy
       Default: <empty string>
    --scanBootstrapClassPath
       Set jar scanner scan bootstrap classpath.
       Default: false
    --session-store
      Session store to use (valid options are 'memcache' or 'redis')
    --session-store-ignore-pattern
      Request pattern to not track sessions for. Valid only with memcache
      session store. (default is '.*\.(png|gif|jpg|css|js)$'. Has no effect
      for 'redis')
      Default: .*\.(png|gif|jpg|css|js)$
    --session-store-locking-mode
      Session locking mode for use with memcache session store. (default is
      all. Has no effect for 'redis')
      Default: all
    --session-store-operation-timeout
      Operation timeout for the memcache session store. (default is 5000ms)
      Default: 5000
    --session-store-pool-size
      Pool size of the session store connections (default is 10. Has no effect
      for 'memcache')
      Default: 10
    --session-timeout
      The number of minutes of inactivity before a user's session is timed
      out.
    --shutdown-override
      Overrides the default behavior and casues Tomcat to ignore lifecycle
      failure events rather than shutting down when they occur.
      Default: false
    --temp-directory
      Define the temp directory, default value: ./target/tomcat.PORT
    --tomcat-users-location
      Location of the tomcat-users.xml file. (relative to the location of the
      webapp-runner jar file)
    --uri-encoding
      Set the URI encoding to be used for the Connector.
    --use-body-encoding-for-uri
      Set if the entity body encoding should be used for the URI.
      Default: false
    -A
      Allows setting HTTP connector attributes. For example: -Acompression=on
      Syntax: -Akey=value
      Default: {}
```

See the Tomcat documentation for a complete list of HTTP connector attributes.

Using without Memcached or Redis

The default packaging of Webapp Runner (`com.github.jsimone:webapp-runner`) includes client libraries for Memcached and Redis so they can easily be used for session storage. If you do not require these dependencies, you can alternative use the `webapp-runner-main` packaging thusly:

```
<dependency>
  <groupId>com.github.jsimone</groupId>
  <artifactId>webapp-runner-main</artifactId>
  <version>${webapp-runner.version</version>
</dependency>
```

If you are encountering `NoClassDefFoundError` or other conflicts in dependency versions, this may resolve your problem.

Development

To run the entire suite of integration tests, use the following command:

```
$ mvn clean install -Pintegration-test
```

To run an individual integration test, use a command like this:

```
$ mvn clean install -Pintegration-test -Dinvoker.test=memcache-test
```

License