P6Spy UI Plugin - Reference Documentation

Authors: Burt Beckwith

Version: 1.0.0

Table of Contents

1 Introduction to the P6Spy UI Plugin

The P6Spy UI plugin uses the P6Spy library to intercept JDBC calls and display them in a web page. One benefit of P6Spy is that it will display the SQL that was actually run with SQL ? placeholders but also the SQL with the actual values.

{info} Note that the plugin does not require the use of the Grails p6spy plugin and is actually incompatible with it. If you are currently using it, migrate your settings from spy.properties to Config.groovy as described in the configuration section and uninstall it. {info}

The plugin includes with a controller and a GSP that will display the executed SQL in a DataTable, and also graphs in a second tab. Once the plugin is installed navigate to http://localhost:8080/appname/p6spy to view the queries and graphs, or http://localhost:8080/appname/p6spy/admin for a basic page displaying configuration settings.

Release history

- 1/04/2016 1.0.0 release
- 11/11/2012 Initial 0.1 release

2 Configuration

P6Spy works by intercepting JDBC calls to log the executed SQL and parameters. There are multiple ways to configure this; by default the plugin will update the JDBC URL and driver class name for you, and the P6Spy driver will intercept calls and re-route them to the real driver. This is not possible if you use a JNDI data source, so in that case or if the configuration is more complex you can manually configure the DataSource.

If you cannot or don't want to use the auto-configuration feature, the first thing you need to do is change the `driverClassName` property in `DataSource.groovy` (or in the JNDI configuration) for any environments that you want to use the plugin in. Change it to `com.p6spy.engine.spy.P6SpyDriver`, e.g.

```
dataSource {
   …
   driverClassName = 'com.p6spy.engine.spy.P6SpyDriver'
   …
}
```

Having done this, you need to tell P6Spy what the real driver is. You do not create a `spy.properties` file like you usually do when working with P6Spy - instead you store settings in Config.groovy. This is more convenient since the plugin has many default values already set, so you only need to set the values that are required (currently just "realdriver" and only if you're not using auto-configuration) and any overrides or other values that don't have defaults set. In addition you can also externalize properties and take advantage of other features of setting properties in Config.groovy. If you're using MySQL you would specify the real driver class as

```
grails.plugin.p6spy.config.realdriver = 'com.mysql.jdbc.Driver'
```

and if you're using a different database change the value to the appropriate driver class.

The following table summarizes the various configuration options. Options that start with "config" are passed through to P6Spy, and those starting with "gsp" and "updateDataSource" are plugin-specific.

All must be set if Config.groovy or an external config file, and must include the `grails.plugin.p6spy.` prefix, e.g.

```
grails.plugin.p6spy.config.jmx = false
```

See the P6Spy documentation for more information about the available P6Spy options.

Property	Default	Meaning
config.appender	'grails.plugin.p6spy.ui.MemoryLogger'	class name of the appender to use
config.autoflush	true	whether to flush per statement

config.databaseDialectDateFormat	'dd-MMM-yy'	SimpleDateFormat format used for logging of PreparedStatement date/time values
config.dateformat	*none*	SimpleDateFormat format used for logging of the query time (if not set, millseconds since epoch is logged)
config.driverlist	*none*	comma-separated list of JDBC drivers to load and register
config.exclude	*none*	if filter is true, a comma-separated list of strings used to define the include/exclude filter rule
config.excludecategories	'batch, debug, info, result, resultset'	comma-separated list of category names to exclude
config.executionthreshold	'0'	if set, only statements that have taken longer than the time specified (in milliseconds) will be logged
config.filter	false	if true, filter what is logged
config.include	*none*	if filter is true, a comma-separated list of strings used to define include/exclude filter rule
config.jmx	true	whether to expose options via JMX
config.jmxPrefix	*none*	if jmx is true, the prefix used for the naming pattern
config.jndicontextcustom	*none*	JNDI config setting if using external JNDI
config.jndicontextfactory	*none*	JNDI config setting if using external JNDI
config.jndicontextproviderurl	*none*	JNDI config setting if using external JNDI
config.modulelist	'com.p6spy.engine.spy.P6SpyFactory, com.p6spy.engine.logging.P6LogFactory'	comma-separated list of names of module classes which should be active

config.outagedetection	false	if true, detects long-running statements that may indicate a database outage and logs statements that exceed outagedetectioninterval seconds; when enabled, only long-running statements are logged
config.outagedetectioninterval	'30'	threshold (in seconds) for outage detection
config.realdatasource	*none*	if not using auto-update, the real JNDI name
config.realdatasourceclass	*none*	if not using auto-update, the name of the real DataSource class
config.realdatasourceproperties	*none*	if not using auto-update, optional DataSource url properties
config.sqlexpression	*none*	if filter is true, a regex defining filter rule for SQL statements
config.stacktrace	false	if true, log a stack trace for every statement logged
config.stacktraceclass	*none*	if stacktrace is true, a class name which must be present in the stacktrace for the stacktrace to be logged
gsp.layoutAdmin	'main'	the layout to use for admin.gsp
gsp.layoutIndex	'p6spy-ui'	the layout to use for index.gsp
updateDataSource.autoUpdate	true	if true, update the DataSource URL to include "p6spy:" to trigger P6Spy's auto-update feature
updateDataSource.driverClassNameProperty	'driverClassName'	if autoUpdate is true, the name of the DataSource driver class name property
updateDataSource.urlProperty	'url'	if autoUpdate is true, the name of the DataSource 'url' property