p6spy Documentation

Release 3.7.1-SNAPSHOT

p6spy team

Jun 01, 2018

Contents

P6Spy is a framework that enables database activity to be seamlessly intercepted and logged with no code changes to existing applications.

Contents:

CHAPTER 1

P6Spy Installation

This section will document the steps to install P6Spy on various application servers. In additional, it contains *Generic Instructions* for applications servers not listed as well as applications that do not use an application server. If you create instructions for other application servers, send us a copy for possible publication in the documentation.

The instructions for all application servers make the following assumptions.

1. The operating system is *nix. For Windows installations, the steps are the same but the syntax will be a little different (environment variables, path separators, etc).
2. MySQL is the database being used. If you are using a different database, just substitute the JDBC connection URL and driver class appropriate for your database.
3. Database connections are being obtained from a JNDI DataSource configured on the application server. If the application does not use a JNDI DataSource, the instructions for modifying the data source configuration will be incorrect. You will need to use the instructions as guidelines for modifying the application specific database configuration. The instructions for adding the p6spy jar file and spy.properties will still be correct.
4. You have already downloaded the P6Spy distribution and extracted the contents to a temporary directory. Throughout the rest of the instructions, the files included in this temporary directory will be referenced by name only.
5. Your application is running on Java 1.6 or later. For earlier versions of Java, you will need to use P6Spy 1.3

After you have completed the installation, a log file called **spy.log** will be created in the current working directory when the application runs. This log file will contain a list of the various database statements executed. You can alter the location of this log file as well as what gets logged by editing **spy.properties**. See Common Property File Settings for the various configuration options available.

Application Servers:

1.1 JBoss/WildFly

The following sections contain specific information on installing P6Spy on *JBoss 4.2.x, 5.1.x, 6.1.x and JBoss 5.x EAP* and *JBoss 7.1.x, WildFly 8.x*

Please note **XA Datasource proxying IS NOT supported** for these.

1.1.1 JBoss 4.2.x, 5.1.x, 6.1.x and JBoss 5.x EAP

The following instructions were tested with JBoss 4.2.3.GA, 5.1.0.GA, 6.1.0.Final and JBoss 5.2.0 EAP. For these instructions, P6Spy assumes that you are using the default server residing in `$JBOSS_DIST\server\default`, where $JBOSS_DIST is the directory in which JBoss is installed.

1. Move the **p6spy.jar** file to the `$JBOSS_DIST\server\default\lib` directory.
2. Move the **spy.properties** file to the `$JBOSS_DIST\server\default\conf` directory.
3. Update the connection URL and driver class for your data source in `$JBOSS_DIST\server\default\deploy`. This file is normally called `?????-ds.xml`. An example of the pertinent portions (not the complete XML file) follows:

```
<jndi-name>MySqlDS</jndi-name>
<connection-url>jdbc:p6spy:mysql://<hostname>:<port>/<database></connection-url>
<driver-class>com.p6spy.engine.spy.P6SpyDriver</driver-class>
```

1.1.2 JBoss 7.1.x, WildFly 8.x, WildFly 10.x

The following instructions were tested with JBoss 7.1.0, WildFly 8.1.Final and WildFly 10.0.0.Final (works with p6spy version 2.1.0 or higher). For these instructions, P6Spy assumes that you are using the standalone and $JBOSS_DIST is the directory in which JBoss/WildFly is installed.

1. Deploy **p6spy.jar** as a module:

   - via moving it to to the `$JBOSS_DIST\modules\system\layers\base\com\p6spy\main` (for Wildfly) or to `$JBOSS_DIST\modules\com\p6spy\main` (for JBoss 7.1) directory
   - and via providing `module.xml` in the same directory with the contents:

     <resource-root path="p6spy-2.0.3.jar"/>

   please note, that p6spy-2.0.3 version jar is used in the sample configuration. Moreover the reference to module holding the real (proxied) jdbc driver has to be provided (in the sample case is h2 one used).
2. Move the **spy.properties** file to the `$JBOSS_DIST\bin` directory.
3. Update the connection URL and driver section in your `<datasources>` in `$JBOSS_DIST\standalone\configuration\standalone.xml`. An example of the pertinent portions (not the complete XML file) follows:

```
<datasources>
    <datasource jndi-name="java:/jdbc/p6spy" enabled="true" use-java-context=
↪"true" pool-name="p6spyPool">
        <connection-url>jdbc:p6spy:h2:tcp://<hostname>:<port>/<database></
↪connection-url>
        <driver>p6spy</driver>
        ...
    </datasource>
```

(continues on next page)

(continued from previous page)

```
    <drivers>
        <driver name="p6spy" module="com.p6spy">
            <driver-class>com.p6spy.engine.spy.P6SpyDriver</driver-class>
        </driver>
        <!-- make sure that you also include the real driver -->
        <driver name="h2" module="com.h2database.h2">
        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
    </driver>
    </drivers>
</datasources>
```

Please note that you have to include both drivers in the corresponding section, the real driver (here the H2 driver) and the p6spy driver, although only the p6spy driver is referenced after the connection URL.

1.2 Apache Tomcat and Apache TomEE

The following sections contain specific information on installing P6Spy on *Tomcat 6.x, 7.x, 8.x and TomEE 1.6.x*.

1.2.1 Apache Tomcat 6.x, 7.x, 8.x and TomEE 1.6.x

The following instructions were tested with Apache Tomcat versions: 6.0.34, 7.0.54 and 8.0.15 as well as Apache TomEE 1.6.0.2 Webprofile and Apache TomEE 1.6.0.2 Plus. For these instructions, it is assumed that $CATALINA_HOME refers to the tomcat/tomee installation directory. Please be aware that there are many ways to configure JNDI data sources on tomcat/tomee.

1. Move the **p6spy.jar** file to the lib directory. An example of the path to your lib directory is `$CATALINA_HOME\lib\`.
2. Move the **spy.properties** file to the lib directory. An example of the path to your lib directory is `$CATALINA_HOME\lib\`.
3. Configure the class name of the real JDBC driver in **spy.properties**

```
driverlist=com.mysql.jdbc.Driver
```

4. Modify the JDBC connection URL and driver class for the data source. Please be aware that there are several places where a JNDI data source may be defined. It is normally defined in a `<Resource/>` element in `$CATALINA_BASE/conf/server.xml` or in the application specific `$CATALINA_BASE/conf/catalina/localhost/????.xml`.See Tomcat 6 JNDI Resources/Tomcat 7 JNDI Resources/Tomcat 8 JNDI Resources for specifics of where a data source is configured. An example of the pertinent portions of the resource definition are shown below.

```
<Resource name="jdbc/mydb"
          type="javax.sql.DataSource"
          ...
          driverClassName="com.p6spy.engine.spy.P6SpyDriver"
          url="jdbc:p6spy:mysql://<hostname>:<port>/<database>"
          ...
          />
```

1.3 Glassfish and Payara

The following section contains specific information on installing P6Spy on *Glassfish 3.1.2.2, 4.0 and Payara 4.1.144* (works with p6spy version 2.1.0 or higher).

Please note **XA Datasource proxying IS supported** for these.

1.3.1 Glassfish 3.1.2.2, 4.0 and Payara 4.1.144

The provided instructions were tested with Glassfish OSE 3.1.2.2, Glassfish OSE 4.0 and Payara 4.1.144. In later section is `$GLASSFISH_HOME` the directory where Glassfish/Payara is installed and `$DOMAIN_X` is the domain name used for deployment (for example, can be: `domain1`).

1. Move the **p6spy.jar** file to the `$GLASSFISH_HOME/domains/$DOMAIN_X/lib/ext` directory.
2. Move the **spy.properties** file to the `$GLASSFISH_HOME/domains/$DOMAIN_X/config` directory.
3. Configure new datasource. Please note there are 3 configuration options available:

   - updating JDBC Url if using `java.sql.Driver`

   using command line:

```
    # create jdbc connection pool
    asadmin create-jdbc-connection-pool --driverclassname=com.p6spy.engine.spy.
↪P6SpyDriver --restype=java.sql.Driver --property=URL='jdbc:p6spy:h2:tcp://
↪<hostname>:<port>/<database>':User='<username>':Password='<password>' p6spyPool

    # ping the pool to prove it works (optionally)
    asadmin ping-connection-pool p6spyPool

    # create jdbc resource
    asadmin --user=<asadmin_user> --passwordfile=<sample_passworfile.properties>␣
↪create-jdbc-resource --connectionpoolid=p6spyPool jdbc/p6Spy
```

   or directly by editing `$GLASSFISH_HOME/domains/$DOMAIN_X/config/domain.xml` (please note the previous commands would lead to similar added to your config file):

```
    <jdbc-connection-pool driver-classname="com.p6spy.engine.spy.P6SpyDriver"␣
↪res-type="java.sql.Driver" name="p6spyPool">
        <property name="URL" value="jdbc:p6spy:h2:tcp://<hostname>:<port>/
↪<database>"></property>
        <property name="Password" value=""></property>
        <property name="User" value="sa"></property>
    </jdbc-connection-pool>
    <jdbc-resource pool-name="p6spyPool" jndi-name="jdbc/p6spy"></jdbc-resource>
```

   - `javax.sql.ConnectionPoolDataSource` proxying (via additional datasource)

   using command line:

```
    # create jdbc connection pool
    asadmin create-jdbc-connection-pool --datasourceclassname=com.p6spy.engine.
↪spy.P6DataSource --restype=javax.sql.ConnectionPoolDataSource --
↪property=realDataSource='<realDSJndi>':User='<username>':Password='<password>'␣
↪p6spyPool

    # ping the pool to prove it works (optionally)
```

(continues on next page)

(continued from previous page)

```
    asadmin ping-connection-pool p6spyPool

    # create jdbc resource
    asadmin --user=<asadmin_user> --passwordfile=<sample_passworfile.properties>␣
↪create-jdbc-resource --connectionpoolid=p6spyPool jdbc/p6Spy
```

or directly by editing `$GLASSFISH_HOME/domains/$DOMAIN_X/config/domain.xml` (please note the previous commands would lead to similar added to your config file):

```
    <jdbc-connection-pool datasource-classname="com.p6spy.engine.spy.P6DataSource
↪" res-type="javax.sql.ConnectionPoolDataSource" name="p6spyPool">
      <property name="realDataSource" value="jdbc/<realDSJndi>"></property>
      <property name="Password" value=""></property>
      <property name="User" value="sa"></property>
    </jdbc-connection-pool>
    <jdbc-resource pool-name="p6spyPool" jndi-name="jdbc/p6spy"></jdbc-resource>
```

- `javax.sql.XADataSource` proxying (via additional datasource)

using command line:

```
    # create jdbc connection pool
    asadmin create-jdbc-connection-pool --datasourceclassname=com.p6spy.engine.
↪spy.P6DataSource --restype=javax.sql.XADataSource --property=realDataSource='
↪<realDSJndi>':User='<username>':Password='<password>' p6spyPool

    # ping the pool to prove it works (optionally)
    asadmin ping-connection-pool p6spyPool

    # create jdbc resource
    asadmin --user=<asadmin_user> --passwordfile=<sample_passworfile.properties>␣
↪create-jdbc-resource --connectionpoolid=p6spyPool jdbc/p6Spy
```

or directly by editing `$GLASSFISH_HOME/domains/$DOMAIN_X/config/domain.xml` (please note the previous commands would lead to similar added to your config file):

```
    <jdbc-connection-pool datasource-classname="com.p6spy.engine.spy.P6DataSource
↪" res-type="javax.sql.XADataSource" name="p6spyPool">
      <property name="realDataSource" value="jdbc/<realDSJndi>"></property>
      <property name="Password" value=""></property>
      <property name="User" value="sa"></property>
    </jdbc-connection-pool>
    <jdbc-resource pool-name="p6spyPool" jndi-name="jdbc/p6spy"></jdbc-resource>
```

Please note, you need to replace following:

- `<asadmin_user>` - asadmin user name (by default `asadmin`)
- `<sample_passworfile.properties>` - is a properties file, that should hold your asadmin password (by default should hold: `AS_ADMIN_ADMINPASSWOD=adminadmin`)
- `<username>` - username to be used for the DB access
- `<password>` - password to be used for DB access
- `<hostname>` - DB server hostname
- `<port>` - DB server port
- `<database>` - DB server database

- `<realDSJndi>` - jndi-name of the real datasource to be proxied

And the jndi name of the created jndi resource in the sample configurations is: `jdbc/p6spy`

1.4 Weblogic

The following section contains specific information on installing P6Spy on *Weblogic 12.1.3* (works with p6spy version 2.1.0 or higher).

1.4.1 Weblogic 12.1.3

The provided instructions were tested with Weblogic 12.1.3 (for developers). In later section is `$WLS_HOME` the directory where Weblogic is installed and `$DOMAIN_X` is the domain name used for deployment (for example, can be: `mydomain`).

1. Move the **p6spy.jar** file to the `$WLS_HOME/user_projects/domains/$DOMAIN_X/lib` directory.
2. Move the **spy.properties** file to the `$WLS_HOME/user_projects/domains/$DOMAIN_X` directory.
3. Update JDBC URL in the datasource to something like:

```
jdbc:p6spy:mysql://<hostname>:<port>/<database>
```

4. Change driver in the datasource to:

```
com.p6spy.engine.spy.P6SpyDriver
```

1.5 Generic Instructions

The following installation instructions are intended for use with other application servers and applications that do not use application servers. To install P6Spy, complete the following steps:

1. Put the **p6spy.jar** file in your classpath.
2. Put **spy.properties** into a directory which is on the classpath. Many application servers have a directory for configuration files which are accessible via the classpath. Most applications which do not run an application server will have one as well.
3. Configure the class name of the real JDBC driver in **spy.properties**

```
driverlist=com.mysql.jdbc.Driver
```

4. Configure the data source.

If the JNDI DataSource is configured using a driver class (implements javax.sql.Driver), then you should modify the JDBC connection URL to include 'p6spy:' and update the driver class to `com.p6spy.engine.spy.P6SpyDriver`. Example URL including p6spy: `jdbc:p6spy:mysql://<hostname>:<port>/<database>`.

If the JNDI DataSource is configured using a data source class (implements javax.sql.DataSource) then you will need to create a 'proxy' data source using the following instructions.

1. Rename the JNDI name of the current data source to something else. For example, if the current name is 'jdbc/myds', then change it to 'jdbc/myds-real'.

2. Create a new JNDI DataSource with the original name of the real data source. Continuing with the example from above, the JNDI name should be 'jdbc/myds'.
3. Create one property for the data source called 'RealDataSource'. The value of this property should be 'jdbc/myds-real'
4. Set the class or implementation to `com.p6spy.engine.spy.P6DataSource`.
5. If the application server requires a classpath for the datasource, it should include p6spy.jar and spy.properties.

CHAPTER 2

Integrating P6Spy

A very typical use case for P6Spy is to enabled SQL logging to troubleshoot various database related issues during development. Assuming that making code changes is acceptable, then the following instructions can be used. If making code changes is not a viable option, then following the instructions for Installing P6Spy.

1. Add **p6spy.jar** to the classpath. If your application uses Maven, Ivy, Gradle, etc just add a dependency on p6spy:p6spy.
2. Wrap your DataSource with P6DataSource or modify your connection URL to add 'p6spy:'.

If your application uses a DataSource, simply wrap your current DataSource object with P6DataSource. P6DataSource has a constructor method that accepts the DataSource to wrap. This is by far the simplest method especially if you use a dependency injection framework such as Spring or Guice.

If your application obtains connections from DriverManager, simply modify your JDBC connection URL to include 'p6spy:'. For example, if your URL is `jdbc:mysql://host/db` then just change it to `jdbc:p6spy:mysql://host/db`. P6Spy implements the JDBC 4.0 API allowing automatic registration of our JDBC driver with DriverManager. As long as your application obtains connections through DriverManager, you only need to modify your database connection URL to activate P6Spy.

By default, a file called spy.log will be created in the current working directory. To customize the logging (including using your application's logging framework) you can provide alternate configuration in a file called spy.properties. This file just needs to be at the root of the classpath. See Configuration and Usage for details.

CHAPTER 3

Configuration and Usage

Configuration follows **layered approach**, where **each layer overrides the values set by the lower ones** (leaving those not provided unchanged):

- JMX set properties (please note, that these are reset on next reload)
- System properties
- Environment variables
- spy.properties
- defaults

For the full list of available options, see the section *Common Property File Settings*. Please note that providing any of these via System properties/Environment variables is possible, using the particular property name following naming rule: `p6spy.config.<property name><property value>`;

Please be aware of the restriction. In fact this also means you need to be aware of values set by the lower configuration layers (including defaults) to properly override/modify those.

There are **two cases one needs to distinguish when overriding**:

- don't override the property on the current level (can be achieved by specifying neither key nor value) and
- clear the property value (can be achieved by specifying the key empty string value, could be for specified in `spy.properties` like this: `excludecategories=`)

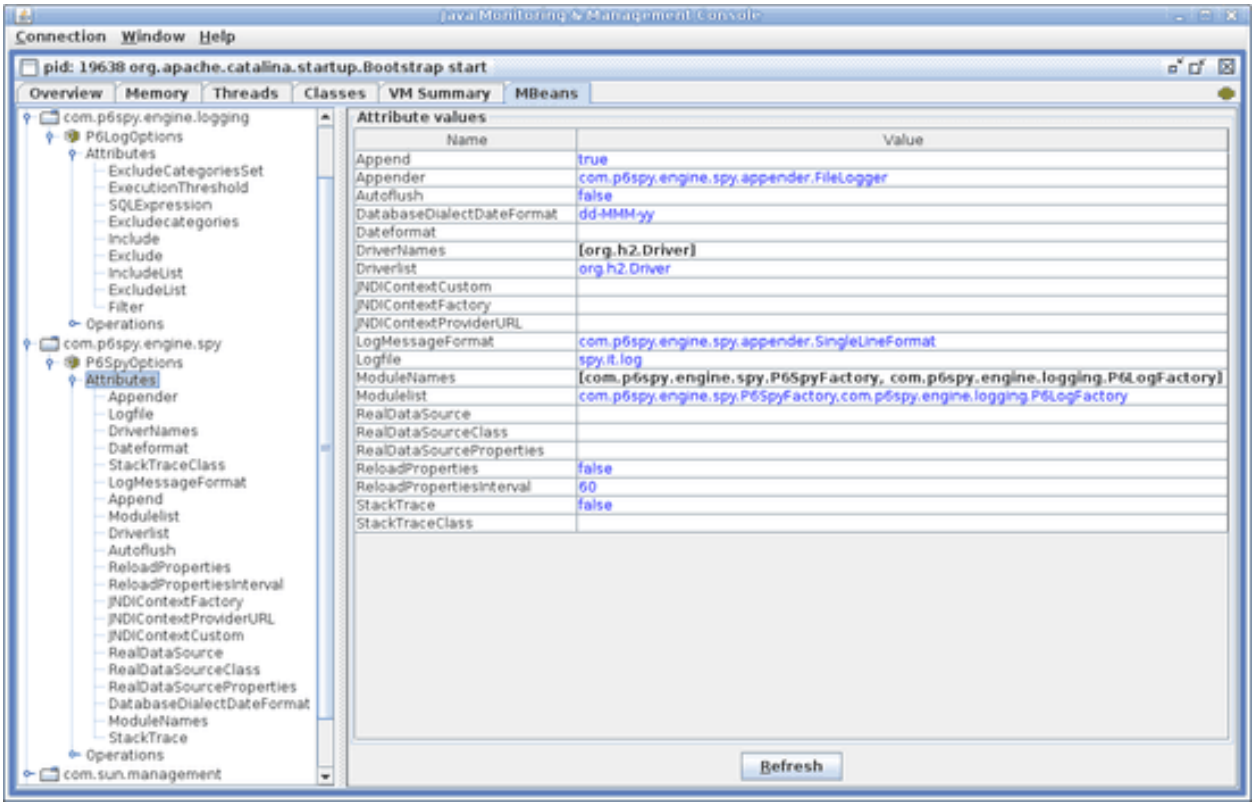
The `spy.properties` configuration file can be located in various places. The following locations are searched to locate the file.

1. The file name configured in the system property "spy.properties" (can include path)
2. The current working directory (for relative path) or any directory (for absolute path)
3. The classpath

3.1 Properties exposal via JMX

Please note that all the properties are exposed via JMX. So you can use your tool of choice (e.g.,JConsole) to view/change them. Moreover reload operation is exposed as well. To provide on-demand reload option.

In the JConsole p6spy related JMX attributes might look like this:



3.2 Command Line Options

Every parameter specified in the property file can be set and overriden at the command line using the Java -D flag (system property), adding the the prefix:

```
p6spy.config.
```

An example follows:

```
java -Dp6spy.config.logfile=my.log -Dp6spy.config.append=true
```

Moreover to set different file to be used as the properties file (as an example: another_spy.properties), it should be specified using system property "spy.properties" as:

```
java -Dspy.properties=c:\jboss\lib\another_spy.properties
```

3.3 Common Property File Settings

An example `spy.properties` file follows (please note default values mentioned as these refer to defaults mentioned in section: *Configuration and Usage*):

```
#*************************************************************
# P6Spy Options File                                         #
# See documentation for detailed instructions                #
# http://p6spy.github.io/p6spy/2.0/configandusage.html       #
#*************************************************************

#*************************************************************
# MODULES                                                    #
#                                                            #
# Module list adapts the modular functionality of P6Spy.     #
# Only modules listed are active.                            #
# (default is com.p6spy.engine.logging.P6LogFactory and      #
# com.p6spy.engine.spy.P6SpyFactory)                         #
# Please note that the core module (P6SpyFactory) can't be   #
# deactivated.                                               #
# Unlike the other properties, activation of the changes on  #
# this one requires reload.                                  #
#*************************************************************
#modulelist=com.p6spy.engine.spy.P6SpyFactory,com.p6spy.engine.logging.P6LogFactory,
↪com.p6spy.engine.outage.P6OutageFactory

#*************************************************************
# CORE (P6SPY) PROPERTIES                                    #
#*************************************************************

# A comma separated list of JDBC drivers to load and register.
# (default is empty)
#
# Note: This is normally only needed when using P6Spy in an
# application server environment with a JNDI data source or when
# using a JDBC driver that does not implement the JDBC 4.0 API
# (specifically automatic registration).
#driverlist=

# for flushing per statement
# (default is false)
#autoflush=false

# sets the date format using Java's SimpleDateFormat routine.
# In case property is not set, milliseconds since 1.1.1970 (unix time) is used␣
↪(default is empty)
#dateformat=

# prints a stack trace for every statement logged
#stacktrace=false
# if stacktrace=true, specifies the stack trace to print
#stacktraceclass=

# determines if property file should be reloaded
# Please note: reload means forgetting all the previously set
# settings (even those set during runtime - via JMX)
# and starting with the clean table
```

(continues on next page)

(continued from previous page)

```
# (default is false)
#reloadproperties=false

# determines how often should be reloaded in seconds
# (default is 60)
#reloadpropertiesinterval=60

# specifies the appender to use for logging
# Please note: reload means forgetting all the previously set
# settings (even those set during runtime - via JMX)
# and starting with the clean table
# (only the properties read from the configuration file)
# (default is com.p6spy.engine.spy.appender.FileLogger)
#appender=com.p6spy.engine.spy.appender.Slf4JLogger
#appender=com.p6spy.engine.spy.appender.StdoutLogger
#appender=com.p6spy.engine.spy.appender.FileLogger

# name of logfile to use, note Windows users should make sure to use forward slashes␣
↪in their pathname (e:/test/spy.log)
# (used for com.p6spy.engine.spy.appender.FileLogger only)
# (default is spy.log)
#logfile=spy.log

# append to the p6spy log file. if this is set to false the
# log file is truncated every time. (file logger only)
# (default is true)
#append=true

# class to use for formatting log messages (default is: com.p6spy.engine.spy.appender.
↪SingleLineFormat)
#logMessageFormat=com.p6spy.engine.spy.appender.SingleLineFormat

# Custom log message format used ONLY IF logMessageFormat is set to com.p6spy.engine.
↪spy.appender.CustomLineFormat
# default is %(currentTime)|%(executionTime)|%(category)|connection%(connectionId)|
↪%(sqlSingleLine)
# Available placeholders are:
#   %(connectionId)            the id of the connection
#   %(currentTime)             the current time expressing in milliseconds
#   %(executionTime)           the time in milliseconds that the operation took to␣
↪complete
#   %(category)                the category of the operation
#   %(effectiveSql)            the SQL statement as submitted to the driver
#   %(effectiveSqlSingleLine)  the SQL statement as submitted to the driver, with all␣
↪new lines removed
#   %(sql)                     the SQL statement with all bind variables replaced␣
↪with actual values
#   %(sqlSingleLine)           the SQL statement with all bind variables replaced␣
↪with actual values, with all new lines removed
#customLogMessageFormat=%(currentTime)|%(executionTime)|%(category)|connection
↪%(connectionId)|%(sqlSingleLine)

# format that is used for logging of the date/time/... (has to be compatible with␣
↪java.text.SimpleDateFormat)
# (default is dd-MMM-yy)
#databaseDialectDateFormat=dd-MMM-yy
```

(continues on next page)

(continued from previous page)

```
# format that is used for logging booleans, possible values: boolean, numeric
# (default is boolean)
#databaseDialectBooleanFormat=boolean

# whether to expose options via JMX or not
# (default is true)
#jmx=true

# if exposing options via jmx (see option: jmx), what should be the prefix used?
# jmx naming pattern constructed is: com.p6spy(.<jmxPrefix>)?:name=<optionsClassName>
# please note, if there is already such a name in use it would be unregistered first␣
↪(the last registered wins)
# (default is none)
#jmxPrefix=

# if set to true, the execution time will be measured in nanoseconds as opposed to␣
↪milliseconds
# (default is false)
#useNanoTime=false

#################################################################
# DataSource replacement                                        #
#                                                               #
# Replace the real DataSource class in your application server  #
# configuration with the name com.p6spy.engine.spy.P6DataSource #
# (that provides also connection pooling and xa support).       #
# then add the JNDI name and class name of the real             #
# DataSource here                                               #
#                                                               #
# Values set in this item cannot be reloaded using the          #
# reloadproperties variable. Once it is loaded, it remains      #
# in memory until the application is restarted.                 #
#                                                               #
#################################################################
#realdatasource=/RealMySqlDS
#realdatasourceclass=com.mysql.jdbc.jdbc2.optional.MysqlDataSource

#################################################################
# DataSource properties                                         #
#                                                               #
# If you are using the DataSource support to intercept calls    #
# to a DataSource that requires properties for proper setup,    #
# define those properties here. Use name value pairs, separate  #
# the name and value with a semicolon, and separate the         #
# pairs with commas.                                            #
#                                                               #
# The example shown here is for mysql                           #
#                                                               #
#################################################################
#realdatasourceproperties=port;3306,serverName;myhost,databaseName;jbossdb,foo;bar

#################################################################
# JNDI DataSource lookup                                        #
#                                                               #
# If you are using the DataSource support outside of an app     #
# server, you will probably need to define the JNDI Context     #
# environment.                                                  #
```

(continues on next page)

(continued from previous page)

```
#                                                                   #
# If the P6Spy code will be executing inside an app server then #
# do not use these properties, and the DataSource lookup will   #
# use the naming context defined by the app server.             #
#                                                                   #
# The two standard elements of the naming environment are       #
# jndicontextfactory and jndicontextproviderurl. If you need    #
# additional elements, use the jndicontextcustom property.      #
# You can define multiple properties in jndicontextcustom,      #
# in name value pairs. Separate the name and value with a       #
# semicolon, and separate the pairs with commas.                #
#                                                                   #
# The example shown here is for a standalone program running on #
# a machine that is also running JBoss, so the JNDI context     #
# is configured for JBoss (3.0.4).                              #
#                                                                   #
# (by default all these are empty)                              #
#####################################################################
#jndicontextfactory=org.jnp.interfaces.NamingContextFactory
#jndicontextproviderurl=localhost:1099
#jndicontextcustom=java.naming.factory.url.pkgs;org.jboss.naming:org.jnp.interfaces

#jndicontextfactory=com.ibm.websphere.naming.WsnInitialContextFactory
#jndicontextproviderurl=iiop://localhost:900

#####################################################################
# P6 LOGGING SPECIFIC PROPERTIES                                    #
#####################################################################

# filter what is logged
# please note this is a precondition for usage of: include/exclude/sqlexpression
# (default is false)
#filter=false

# comma separated list of strings to include
# please note that special characters escaping (used in java) has to be done for the␣
↪provided regular expression
# (default is empty)
#include=
# comma separated list of strings to exclude
# (default is empty)
#exclude=

# sql expression to evaluate if using regex
# please note that special characters escaping (used in java) has to be done for the␣
↪provided regular expression
# (default is empty)
#sqlexpression=

#list of categories to exclude: error, info, batch, debug, statement,
#commit, rollback, result and resultset are valid values
# (default is info,debug,result,resultset,batch)
#excludecategories=info,debug,result,resultset,batch

#whether the binary values (passed to DB or retrieved ones) should be logged with␣
↪placeholder: [binary] or not.
# (default is false)
```

(continues on next page)

(continued from previous page)

```
#excludebinary=false

# Execution threshold applies to the standard logging of P6Spy.
# While the standard logging logs out every statement
# regardless of its execution time, this feature puts a time
# condition on that logging. Only statements that have taken
# longer than the time specified (in milliseconds) will be
# logged. This way it is possible to see only statements that
# have exceeded some high water mark.
# This time is reloadable.
#
# executionThreshold=integer time (milliseconds)
# (default is 0)
#executionThreshold=

#################################################################
# P6 OUTAGE SPECIFIC PROPERTIES                                 #
#################################################################
# Outage Detection
#
# This feature detects long-running statements that may be indicative of
# a database outage problem. If this feature is turned on, it will log any
# statement that surpasses the configurable time boundary during its execution.
# When this feature is enabled, no other statements are logged except the long
# running statements. The interval property is the boundary time set in seconds.
# For example, if this is set to 2, then any statement requiring at least 2
# seconds will be logged. Note that the same statement will continue to be logged
# for as long as it executes. So if the interval is set to 2, and the query takes
# 11 seconds, it will be logged 5 times (at the 2, 4, 6, 8, 10 second intervals).
#
# outagedetection=true|false
# outagedetectioninterval=integer time (seconds)
#
# (default is false)
#outagedetection=false
# (default is 60)
#outagedetectioninterval=30
```

3.3.1 modulelist

modulelist holds the list of p6spy modules activated. A module contains a group of functionality. If none are specified only core p6spy framework will be activated (no logging,. . .). Still once reload of the properties happen, or these are set by JMX, modules would be dynamically loaded/unloaded.

The following modules come with the p6spy by default:

```
modulelist=com.p6spy.engine.logging.P6LogFactory,com.p6spy.engine.outage.
↪P6OutageFactory
```

Where these are required:

- com.p6spy.engine.logging.P6LogFactory - for the logging functionality and
- com.p6spy.engine.outage.P6OutageFactory - for outage functionality.

Please note to implement custom module have a look at the implementation of the any of the existing ones.

3.3.2 driverlist

This is a comma separated list of JDBC driver classes to load and register with DriverManager. You should list the classname(s) of the JDBC driver(s) that you want to proxy with P6Spy if any of the following conditions are met.

1. The JDBC driver does not implement the JDBC 4.0 API
2. You are using a JNDI Data Source - Some application servers will prevent the automatic registration feature from working.

3.3.3 autoflush

For standard development, set the autoflush value to true. When set to true, every time a statement is intercepted, it is immediately written to the log file. In some cases, however, instant feedback on every statement is not a requirement. In those cases, the system performs slightly faster with autoflush set to false.

An example follows:

```
autoflush=true
```

3.3.4 dateformat

Setting a value for dateformat changes the date format value printed in the log file. No value prints the current time in milliseconds (unix time), a useful feature for parsing the log. The date format engine is Java's SimpleDateFormat class. Refer to the SimpleDateFormat class in the JavaDocs for information on setting this value. An example follows:

```
dateformat=MM-dd-yy HH:mm:ss:SS
```

3.3.5 stacktrace

If stacktrace is set, the log prints out the stack trace for each SQL statement logged.

3.3.6 stacktraceclass

Limits the stack traces printed to those that contain the value set in stacktraceclass. For example, specifying stacktraceclass=com.mycompany.myclass limits the printing of stack traces to the specified class value. The stack trace is converted to a String and string.indexOf(stacktraceclass) is performed.

3.3.7 reloadproperties and reloadpropertiesinterval

If reloadproperties is set to true, the property file is reloaded every n seconds, where n is defined by the value set by reloadpropertiesinterval. For example, if reloadproperties=true and reloadpropertiesinterval=10, the system checks the File.lastModified() property of the property file every 10 seconds, and if the file has been modified, it will be reloaded.

If you set append=true, the log will be suddenly truncated when you change your properties. This is because using reloadproperties is intended to be the equivalent of restarting your application server. Restarting your application server truncates your log file.

reloadproperties will not reload any driver information (such as realdriver, realdriver2, and realdriver3) and will not change the modules that are in memory.

3.3.8 appender

Appenders allow you to specify where and how log information is output. Appenders are a flexible architecture allowing anyone to write their own output class for P6Spy. To use an appender, specify the classname of the appender to use. The current release comes with three options which are slf4j, stdout, and logging to a file (default). Please note, that all of these output in the CSV format (where separator is: "|").

- **Using the File output**: Uncomment the `FileLogger` appender and specify a `logfile` and whether or not to `append` to the file or to clear the file each time:

```
#appender=com.p6spy.engine.spy.appender.Slf4JLogger
#appender=com.p6spy.engine.spy.appender.StdoutLogger
appender=com.p6spy.engine.spy.appender.FileLogger

# name of logfile to use, note Windows users should make sure to use forward␣
↪slashes in their pathname (e:/test/spy.log)
# (used for com.p6spy.engine.spy.appender.FileLogger only)
# (default is spy.log)
#logfile=spy.log

# append to the p6spy log file. if this is set to false the
# log file is truncated every time. (file logger only)
append=true
```

- **Using StdOut**: Uncomment the `StdoutLogger` as follows:

```
#appender=com.p6spy.engine.spy.appender.Slf4JLogger
appender=com.p6spy.engine.spy.appender.StdoutLogger
#appender=com.p6spy.engine.spy.appender.FileLogger
```

- **Using SLF4J**: Uncomment the `Slf4JLogger` as follows:

```
appender=com.p6spy.engine.spy.appender.Slf4JLogger
#appender=com.p6spy.engine.spy.appender.StdoutLogger
#appender=com.p6spy.engine.spy.appender.FileLogger
```

  In general you need to slf4j-api and the appropriate bridge to the actual logging implementation as well as the logging implementation itself on your classpath. To simplify setup for those not having any of the additional dependencies already on classpath following `*-nodep.jar` bundles are provided as part of p6spy distribution:

  - `p6spy-<version>-log4j-nodep.jar` - having log4j included,
  - `p6spy-<version>-log4j2-nodep.jar` - having log4j2 included and
  - `p6spy-<version>-logback-nodep.jar` - having logback included.

  Mapping to SLF4J levels is provided in the following way:

  Internally is Slf4j Logger is retrieved for the: `p6spy`, keep this in mind when configuring your logging implementation. So for example for the `log4j` following could be used to restrict the p6spy logging (if using xml-based configuration) to `INFO` level only:

```
<category name="p6spy">
  <priority value="INFO" />
</category>
```

  For further instructions on configuring SLF4J, see the SLF4J documentation.

3.3.9 logMessageFormat

The log message format is selected by specifying the class to use to format the log messages. The following classes are available with P6Spy.

- `com.p6spy.engine.spy.appender.SingleLineFormat` which results in log messages in format:

```
current time|execution time|category|connection id|statement SQL String|effective␣
↪SQL string
```

- `com.p6spy.engine.spy.appender.CustomLineFormat`, which allows log messages to be full customized, in a separate property called `customLogMessageFormat`. See below for details.
- `com.p6spy.engine.spy.appender.MultiLineFormat`, which results in log messages in format:

```
current time|execution time|category|connection id|statement SQL String
effective SQL string
```

Where:

- `current time` - the current time is obtained through System.getCurrentTimeMillis() and represents the number of milliseconds that have passed since January 1, 1970 00:00:00.000 GMT. (Refer to the J2SE documentation for further details on System.getCurrentTimeMillis().) To change the format, use the dateformat property described in *Common Property File Settings*.
- `execution time` - the time it takes in milliseconds for a particular method to execute. (This is not the total cost for the SQL statement.) For example, a statement `SELECT * FROM MYTABLE WHERE THISCOL = ?` might be executed as a prepared statement, in which the .execute() function will be measured. This is recorded as the statement category. Further, as you call .next() on the ResultSet, each .next() call is recorded in the result category.
- `category` - You can manage your log by including and excluding categories, which is described in *Common Property File Settings*.
- `connection id` - Indicates the connection on which the activity was logged. The connection id is a sequentially generated identifier.
- `statement SQL string` - This is the SQL string passed to the statement object. If it is a prepared statement, it is the prepared statement that existed prior to the parameters being set. To see the complete statement, refer to effective SQL string.
- `effective SQL string` - If you are not using a prepared statement, this contains no value. Otherwise, it fills in the values of the Prepared Statement so you can see the effective SQL statement that is passed to the database. Of course, the database still sees the prepared statement, but this string is a convenient way to see the actual values being sent to the database.

The `com.p6spy.engine.spy.appender.MultiLineFormat` might be better from a readability perspective. Because it will place the effective SQL statement on a separate line. However, the SingleLineFormat might be better if you have a need to parse the log messages. The default is `com.p6spy.engine.spy.appender.SingleLineFormat` for backward compatibility.

You can also supply your own log message formatter to customize the format. Simply create a class which implements the `com.p6spy.engine.spy.appender.MessageFormattingStrategy` interface and place it on the classpath.

3.3.10 customLogMessageFormat

The custom log message format to use when 'logMessageFormat' is set to `com.p6spy.engine.spy.appender.CustomLineFormat`

The message is build out of the format string, with the all the Java special characters supported (`\n`, `\t` etc) and the following placeholders being resolved to the appropriate values:

- `%(connectionId)` the id of the connection
- `%(currentTime)` the current time expressing in milliseconds
- `%(executionTime)` the time in milliseconds that the operation took to complete
- `%(category)` the category of the operation
- `%(effectiveSql)` the SQL statement as submitted to the driver
- `%(effectiveSqlSingleLine)` the SQL statement as submitted to the driver, with all new lines removed
- `%(sql)` the SQL statement with all bind variables replaced with actual values
- `%(sqlSingleLine)` the SQL statement with all bind variables replaced with actual values, with all new lines removed

3.3.11 filter, include, exclude

P6Spy allows you to filter SQL queries by specific strings to be present (`includes` property value) or not present (`excludes` property value). As a precondition, setting `filter=true` has to be provided. P6Spy will perform string matching on each statement to determine if it should be written to the log file. `include` accepts a comma-delimited list of expressions which is required to appear in a statement before it can appear in the log. `exclude` accepts a comma-delimited list to exclude. Exclusion overrides inclusion, so that a statement matching both an include string and an exclude string is excluded.

Please note that matching mode used in the underlying regex is (achieved via prefix `(?mis)`):

- multiline,
- dotall and
- case insensitive.

An example showing capture of all statements having select, except those having order follow:

```
filter=true
# comma separated list of strings to include
include=select
# comma separated list of strings to exclude
exclude=order
```

Please note, that internally following regex would be used for particular expression matching: `(?mis)^(?!.*(order).*)(.*(select).*)$`

An example showing only capture statements having any of: order_details, price, and price_history follows:

```
filter=true
# comma separated list of strings to include
include=order,order_details,price,price_history
# comma separated list of strings to exclude
exclude=
```

Please note, that internally following regex would be used for particular expression matching: `(?mis)^(.*(order|order_details|price|price_history).*)$`

An example showing the capture of all statements, except statements order string in them follows:

```
filter=false
# comma separated list of strings to include
include=
# comma separated list of strings to exclude
exclude=order
```

Please note, that internally following regex would be used for particular expression matching: `(?mis)^(?!.*(order).*)(.*)$`

As you can use full regex syntax, capture of statements having: pri[cz]e follows:

```
filter=true
# comma separated list of strings to include
include=pri[cz]e
# comma separated list of strings to exclude
exclude=
```

Please note, that internally following regex would be used for particular expression matching: `(?mis)^(.*(pri[cz]e).*)$`

Moreover, please note, that special characters escaping (used in java) has to be done for the provided regular expression. As an example, matching for:

```
from\scustomers
```

would mean, that following should be specified (please note doubled backslash):

```
filter=true
include=from\\scustomers
```

3.3.12 filter, sqlexpression

If you need more control over regular expression for matching, SQL string property `sqlexpression` is to be used as an alternative to `exclude` and `include`. An example follows:

```
filter=true
sqlexpression=your expression
```

If your expression matches the SQL string, it is logged. If the expression does not match, it is not logged. Please note you can use `sqlexpression` together with `include`/`exclude`, where both would be evaluated.

Moreover, please note, that special characters escaping (used in java) has to be done for the provided regular expression. As an example, matching for:

```
^(.*(from\scustomers).*)$
```

would mean, that following should be specified (please note doubled backslash)::

```
filter=true
sqlexpression=^(.*(from\\scustomers).*)$
```

3.3.13 excludecategories

The log includes category information that describes the type of statement. This property excludes the listed categories. Valid options include the following:

- `error` includes P6Spy errors. (It is recommended that you include this category.)
- `info` includes driver startup information and property file information.
- `debug` is only intended for use when you cannot get your driver to work properly, because it writes everything.
- `statement` includes Statements, PreparedStatements, and CallableStatements.
- `batch` includes calls made to the addBatch() JDBC API.
- `commit` includes calls made to the commit() JDBC API.
- `rollback` includes calls made to the rollback() JDBC API.
- `outage` includes outage related information.
- `result` includes statements generated by ResultSet.
- `resultset` includes values retrieve from the ResultSet.

Enter a comma-delimited list of categories to exclude from your log file. See filter, include, exclude for more details on how this process works.

3.3.14 excludebinary

whether the binary values (passed to DB or retrieved ones) should be logged with placeholder: [binary] or not.

3.3.15 outagedetection

This feature detects long-running statements that may be indicative of a database outage problem. When enabled, it logs any statement that surpasses the configurable time boundary during its execution. No other statements are logged except the long-running statements.

3.3.16 outagedetectioninterval

The interval property is the boundary time set in seconds. For example, if set to 2, any statement requiring at least 2 seconds is logged. The same statement will continue to be logged for as long as it executes. So, if the interval is set to 2 and a query takes 11 seconds, it is logged 5 times (at the 2, 4, 6, 8, 10-second intervals).

3.3.17 jmxPrefix

If set to true, the execution time will be measured in nanoseconds as opposed to milliseconds.

CHAPTER 4

Release Notes

4.1 3.7.1 (Unreleased)

Improvements:

Defects resolved:

4.2 3.7.0 (2018-04-03)

Improvements:

- issue #437 Add ability to register custom `JdbcEventListenerFactory`

4.3 3.6.0 (2017-11-12)

Improvements:

- issue #426 The format for boolean parameters made configurable

4.4 3.5.1 (2017-11-02)

Defects resolved:

- issue #423: 3.5.0 version never made it to bintray/maven central (due broken `.travis.yml`)

4.5 3.5.0 (2017-11-02)

Defects resolved:

- issue #417: Fixed `IllegalArgumentException` for `CustomLineFormat` with specific chars in statement
- issue #363: Fixed `NullPointerException` for `OutageJdbcEventListener.onAfterCommit` via refactoring the `P6OutageDetector` to `enum`

Known issues:

- issue #423: 3.5.0 version never made it to bintray/maven central (due broken `.travis.yml`)

4.6 3.4.0 (2017-10-13)

Improvements:

- issue #405: Reintroduced p6spy compatibility with java 6
- issue #412: Introduced event `onBeforeGetConnection`

Defects resolved:

- issue #406: Fixed API breaking changes comming with 3.3.0 via reintroduced `ConnectionWrapper.wrap()`

4.7 3.3.0 (2017-09-09)

Improvements:

- issue #384: Introduced event `onAfterConnectionGet`
- issue #400: Introduced `JdbcEventListenerFactory` (enabling programatic `JdbcEventListener` implementation)
- issue #400: Deprecated `P6Core` favoring `ConnectionWrapper` and `JdbcEventListenerFactory`

Defects resolved:

- issue #404: Reintroduced zip + tar distribution packaging (was missing since 3.1.0 version)
- issue #401: Fixed AOP Error: No visible constructors in class `com.p6spy.engine.wrapper.ConnectionWrapper`

4.8 3.2.0 (2017-09-01)

Improvements:

- issue #391: Introduced support for specifying log message format from configuration file

Defects resolved:

- issue #397: Fixed `%(connectionId)` not replaced in custom format

4.9 3.1.0 (2017-08-22)

Improvements:

- issue #369: Introduced `excludebinary=true|false` flag (causing `[binary]` instead of binary data logged)
- issue #373: Introduced `Loggable.getConnectionInformation()` while removed `Loggable.getConnectionId()`
- issue #367: `StatementInformation.getSqlWithValues()` returns `getSql()` rather than “”

Defects resolved:

- issue #369: Fixed `excludecategories` docs
- issue #387: Fixed `StatementInformation.getSql()` null for connection-pool validation queries

Other:

- issue #292: migrated from maven to gradle for build
- issue #372: integrated release process with travis-ci
- issue #377: release notes synced to github releases
- issue #377: local dev env moved from vagrant to docker

Known issues:

- issue #372: 3.1.0 version never made it to maven central (due to incomplete `pom.xml`), only available in bintray

4.10 3.0.0 (2016-10-26)

Identical to 3.0.0-rc3

4.11 3.0.0-rc3 (2016-10-06)

Improvements:

- issue #359: Add getConnectionInformation to StatementInformation
- issue #358: Add JdbcEventListener.onConnectionWrapped method
- issue #356: Store the creator of a connection and the connection itself in ConnectionInformation

Defects resolved:

- issue #348: tomcat 6x-8x integration

4.12 3.0.0-rc2 (2016-09-08)

Defects resolved:

- issue #347: The real exception is never thrown
- issue #346: Fixes NPE on static initializer order

4.13 3.0.0-rc1 (2016-09-02)

Improvements:

- issue #332: Add event listeners via service loader mechanism
- issue #323: log row count and SQLException
- issue #297: Allow for lazy initialization of P6Spy

4.14 3.0.0-alpha-1 (2016-07-26)

Improvements:

- issue #319: Add support for events
- issue #298: Provide access to ResultSetInformation
- issue #299: Support of reacting differently on SQL-Errors within datasource-proxy
- issue #327: Remove CGLIB

Other:

- issue #333: Remove org.objectweb.util.monolog.wrapper.p6spy.P6SpyLogger

4.15 2.3.1 (2016-06-23)

Defects resolved:

- issue #325: CGLIB lock contention/deadlock

Other:

- issue #326: Upgraded to CGLIB 3.2.3
- issue #329: Use default naming policy for CGLIB generated proxies

4.16 2.3.0 (2016-05-11)

Improvements:

- issue #295: Add option to report execution time in nanoseconds or milliseconds
- issue #313: Remove throws SQLException declaration on P6Core.wrapConnection

Defects resolved:

- issue #320: Calls to getResultSet() not instrumented on Statement and PreparedStatement
- issue #318: Remove unused Cache abstraction and make cache thread safe
- issue #317: Make connectionId thread safe
- issue #314: ClassNotFoundException loading MySQL Statement interface using OSGi

Other:

- issue #304: Build failures using openjdk7

- issue #300: Maven warning about missing dependency

4.17 2.2.0 (2016-03-23)

Improvements:

- issue #290: Lazy initialization for FileLogger

Defects resolved:

- issue #330: Unsafe iteration over System.getProperties()

4.18 2.1.4 (2015-05-09)

Defects resolved:

- issue #286: P6Spy proxy creation fails when JDBC object is wrapped by JBoss 7+
- issue #282: No resultset logged when executing a stored procedure

4.19 2.1.3 (2015-04-02)

Defects resolved:

- issue #275: ArrayIndexOutOfBoundsException when calling PreparedStatement.setMaxRows(int) with outage module enabled

4.20 2.1.2 (2014-10-14)

Defects resolved:

- [issue #268] (https://github.com/p6spy/p6spy/issues/268): SingleLineFormat updated to remove CR and LF characters from the log file
- [issue #267] (https://github.com/p6spy/p6spy/issues/267): The equals(Object) method on all proxied objects now unwraps the argument passed in (if it is a p6spy proxy) before invoking the method on the proxied object. This fixes a problem with c3p0 and statement caching.
- [issue #264] (https://github.com/p6spy/p6spy/issues/264): Fixed a defect causing the last row read of a result set to not be logged unless all rows were read.

4.21 2.1.1 (2014-09-03)

Defects resolved:

- [issue #256] (https://github.com/p6spy/p6spy/issues/256): jmx exposing becomes optional (enabled/disabled via flag) + jmx prefix introduced (see)
- [issue #254] (https://github.com/p6spy/p6spy/issues/254): resultset logging filtering fixed

4.22 2.1.0 (2014-06-15)

Improvements:

- P6ConnectionPoolDataSource merged to P6DataSource (to simplify datasource config)
- `excludecategories` using class `Category` rather than just plain strings (affects `P6Logger` API)
- issue #131: providing additional distribution artifacts - wrapping (slf4j bridged) logging implementations for log4j, log4j2 and logback `p6spy-<version>-*-nodep.jar`
- issue #231: `include`/`exclude` behavior enabling any substring in SQL string matching
- considering Wrapper for DataSource proxies (bringing support for Glassfish XADataSources)
- `unSet*` API provided for properties (in `com.p6spy.engine.spy.P6SpyOptions` and `com.p6spy.engine.logging.P6LogOptions`) to enable reverting to `null` (default value)
- issue #247: `-` prefixed syntax for list-like properties deprecated, in favor of full overriding

Defects resolved:

- [issue #221] (https://github.com/p6spy/p6spy/issues/221): Bind variables set by name on a CallableStatement are now logged
- issue #226: `setAppender()` considered in logging properly
- [issue #227] (https://github.com/p6spy/p6spy/issues/227): fixed disabling modules on reload
- issue #242: character `'` escaping in the logged SQL query fixed
- issue #246: `NullPointerException` fixed for empty batch execution

4.23 2.0.2 (2014-04-04)

Improvements:

- [issue #84] (https://github.com/p6spy/p6spy/issues/84): significant performance improvements for huge data selects

Defects resolved:

- [issue #214] (https://github.com/p6spy/p6spy/issues/214): fixed PostgreSQL issue: `operator is not unique: date + unknown`
- issue #219: fixed defect causing ClassCastException when setting bind variables by name on CallableStatement
- issue #217: fixed defect in P6Leak module causing closed connections not to be recorded properly

4.24 2.0.1 (2014-03-15)

Defects resolved:

- [issue #200] (https://github.com/p6spy/p6spy/issues/200): fixed usage with signed jdbc jars
- [issue #201] (https://github.com/p6spy/p6spy/issues/201): internal logs not printed out any more

4.25 2.0.0 (2014-03-04)

Improvements:

- project hosting was moved from sourceforge to github
- major part of the legacy code was refactored
- Java 6/7 JDBC API support introduced,
- proxying via modified `JDBC URL`s only was implemented, so for for MySQL original url would be (without a need for any further configuration):

```
jdbc:mysql://<hostname>:<port>/<database>
```

  the one proxied via p6spy would one:

```
jdbc:p6spy:mysql://<hostname>:<port>/<database>
```

- XA Datasource support has been introduced
- configuration via:
  - system/environment properties and
  - JMX properties
  - as an alternative to file configuration only
  - or even zero config use case supported
- slf4j support (more flexible as previously used log4j)
- junit tests were migrated to junit 4
- Continuous integration using Travis was setup providing testing on popular:
  - DB systems (namely: Oracle, DB2, PostgreSQL, MySQL, H2, HSQLDB, SQLite, Firebird, and Derby), see build status on: travis-ci as well as
  - application servers (namely: Wildfly 8, JBoss 4.2, 5.1, 6.1, 7.1, Glassfish 3.1, 4.0, Jetty 7.6, 8.1, 9.1, Tomcat 6, 7, 8, Resin 4, Jonas 5.3 and Geronimo 2.1, 2.2), see build status on: travis-ci.

4.26 1.3 (2005-12-27)

- release notes not provided

4.27 1.2

- Driver initialization bug fix and package import cleanup contributed by Joe Fisher (Joe Fisher)
- Further changes to better support JDK 1.2
- Changed a DataSource class name to avoid conflict with an Oracle class of the same name (Alan Arvesen of IronGrid)
- Allow unlimited SQL parameters (Bradley Johnson of IronGrid)

4.28 1.1

- Added a highly requested feature contributed by Jeff Wolfe that only logs queries taking longer than a specified threshold. (Jeff Wolfe)
- Added a bug fix that prevented modified property files from being persisted in Java environments prior to 1.4. (Jeff Wolfe)
- Added JBoss 2.x JMX support, submitted by Ralph Harnden (Ralph Harnden)
- Alan Arvesen of IronGrid added a driver patch that deregisters realdrivers using the same name as the P6 driver to avoid driver order registration problems (Alan Arvesen)

4.29 1.0.1

- Added a bug fix to prevent a NullPointerException from being thrown when there is a space before or after the name of the realdriver. (Paolo De Carlo)

4.30 1.0 Production Release

- In beta for over 6 months, P6Spy version 1.0 is a major rewrite of the P6Spy code. This release includes numerous new features, such as support for JDK 1.4, Datasources, log4j, JBoss 3.x, and WebSphere, as well as an overhauled, optimized architecture.

4.31 1.0 beta 9

- Added full support for DataSources and created installation instructions for WebSphere 4.0. (Dennis Parker, IronGrid)

4.32 1.0 beta 8

- Refactored options to support easier creation of new option files and clearer separation of file management from management of the actual properties.
- Reduced deployment JAR file size by 25%.
- P6Spy Documentation was overhauled. (Suzanne Patton)

4.33 1.0 beta 7

- Created code to enable more robust property file loading, including the ability to use the ClassLoader to load the property file. (Scott Howlett)
- Fixed a problem dealing with the log file being ignored.

4.34 1.0 beta 6

- Debugged and fixed a problem in which applications calling newInstance(), instead of using DriverManager, were causing the driver to load improperly.

4.35 1.0 beta 5

- Refactored the driver loading to first attempt the new classloader mechanism (implemented in 1 beta 2) and, upon failure, attempt the previously-used Name loading call. (Alan Arvesen, IronGrid)
- P6SpyDriver now throws an exception immediately when realdriver fails to load, making diagnostics easier.

4.36 1.0 beta 4

- Added error category and pushed error logging through the standard logging process instead of stderr. This should make it easier to diagnose problems when installation does not proceed as expected.

4.37 1.0 beta 3

- Added a call to each P6 class called getJDBC() that returns the native driver. This enables a workaround for using non-standard JDBC calls included with some JDBC drivers. See Known Issues for more information. Thanks to Ralph Harnden for the suggestion. (Alan Arvesen, IronGrid)
- Added an appender architecture which supports customizable logging. (Alan Arvesen, IronGrid)
- Changed the reloading code to work as a separate thread, making it more efficient. (Alan Arvesen, IronGrid)
- Enhanced module support, making it easier to create a new module. Now, the only required files for a new module are the factory, the driver, and the classes that are changing. If you want to intercept the Statement class only, that is the only class you need to create. In the past, you had to create an instance of every class, even if you did not want to override that class. The code to support driver loading has also been simplified.
- Added JDK 1.4 support. (Matthew Wakeling)

4.38 1.0 beta 1 & 2

- Added log4j support, which is one of the most requested feature enhancements. (Rafael Alvarez)
- Due to problems reported with driver loading in the alpha version, driver loading has been rewritten. It is now more efficient.

4.39 1.0 alpha

- Restructured code to inherit from a single core wrapper driver.
- Introduced concepts of modules and stackable drivers that dynamically load the necessary code into memory at runtime.
- Broke code into logical modules: P6Log and P6Outage.

- Refactored large amount of code.
- Rewrote JUnit tests and added more rigorous tests.

4.40 0.8

- Created an Outage Detection module that reports database outages when database statements do not respond within a given period of time. (Peter Laird)
- Created a JSP application that gives a visual control to P6Spy. The first version can be used to view configuration information about P6Spy and to create a demarcation in the log file. (Peter Laird)
- Added support for multiple simultaneous databases, a highly requested feature. Currently, support is limited to three databases, but can easily be expanded. (Viktor Szathmary)
- Added the logging of a connection ID, and enabled URLs to be prefixed with p6spy: to aid in debugging. A common mistake people make when installing is to have the real driver registered elsewhere; this feature avoids that problem. The default does not require this value, but in the future it may be mandatory to ease the install process.
- Rewrote P6SpyOptions to allow new options to be added quickly and easily.
- Additional JUnit tests added.

4.41 0.72

- Added the ability to dynamically reload the property file after a specified period of time. (Philip Ogren)
- Added logging to commit and rollback statements.

4.42 0.71

- Added installation instructions for BEA WebLogic Portal and Server. (Philip Ogren)
- Added Jakarta RegExp support. (Philip Ogren)
- Ability to print stack trace of logged statements. This is useful in understanding where a logged query is being executed in the application. (Philip Ogren)
- Simplified table monitoring property file option. (Philip Ogren)
- Updated the RegExp documentation.

4.43 0.7

- Added timing information to the log files, in order to better visualize bottleneck queries. (Simon Sadedin)
- Added RegExp support for table filtering, allowing sophisticated custom filtering. (Simon Sadedin)
- Added installation instructions for Sun iPlanet. (Michael Sgroi)
- Added installation instructions for BEA WebLogic. (Richard Delbert)
- Added support for callable statements.

- Added support for batch statements.
- Added a debug category that provides detailed debug information. By default, this is disabled. Refer to the Troubleshooting section for more information.
- Changed the default log format to include more information.
- Added a test target to Ant that works with JUnit to perform some basic tests, using Oracle as the test database.
- Added ResultSet logging and timing information. By default, this is disabled. Refer to the Log File Format documentation for more information.
- Fixed a number of bugs, in particular a bug that was causing an empty spy.log file to be created and never populated.

4.44 0.6

- Fixed a bug in which null connections were not returning null, but rather empty connections. This was a problem for some applications that were expecting a null connection.
- Added an option to allow the truncation/non-truncation of the log file, which can be specified within spy.options.

CHAPTER 5

Known Issues

5.1 Non-standard (driver specific) JDBC methods are not directly accessible

Many drivers provide methods that expose driver-specific, non-standard functionality. Most developers do not use these features, but in the event that an application does use these features they are not natively supported by P6Spy. For example, the MySQL JDBC drivers allow you to call the auto-increment function as follows:

```
((com.mysql.jdbc.PreparedStatement)stmt).getLastInsertId();
```

This fails when P6Spy is active since the prepared statement is actually a proxy generated by P6Spy. Since the proxy is not a subclass of com.mysql.jdbc.PreparedStatement, the cast fails. The only workaround available requires code changes to the application.

All proxies generated by P6Spy implement the java.sql.Wrapper interface. This interface is part of the JDBC 4.0 API (Java 6 and later). This provides a standard way to unwrap the proxied object to obtain access to driver specific methods.

```
// assuming MySQL JDBC driver
PreparedStatement stmt = connection.prepareStatement("....");
if( stmt.isWrapperFor(com.mysql.jdbc.PreparedStatement.class) {
  com.mysql.jdbc.PreparedStatement mySqlStmt = stmt.unwrap(com.mysql.jdbc.
↪PreparedStatement.class);
  mySqlStmt.getLastInsertId();
}
```

Please be aware that any P6Spy will not be able to log any actions performed against the unwrapped object. This is perfectly fine as long as you are only using the non-standard functionality. However, if you use the unwrapped PreparedStatement in the example above to execute a SQL statement, it would not be logged.

CHAPTER 6

Frequently Asked Questions

- Can I use log4j?

  Yes. See the Log File Format documentation for more information.
- Can I use regular expressions to determine what is logged?

  Yes. See the Common Property File Settings documentation and refer to the stringmatcher section.
- Once the application is running, can I change the properties and enable the system to use the new properties?

  Yes. See the Common Property File Settings documentation and refer to the reloadproperties section.

CHAPTER 7

Development

7.1 Prerequisites

1. Make sure to have Java 1.7 or later installed.

7.2 Building the project

The following are useful Gradle commands:

to build binaries:

```
./gradlew assemble
```

to run the JUnit tests Refer to the *Running the tests* section

7.3 License headers

There is a license check done for the source file headers (as part of the CI build), invoked as part of the:

./gradlew check

Once new files is introduced, make sure to run (and push the updated files):

./gradlew licenseFormat

7.4 Releasing the version

The project follows semantic versioning concept. To release the version follow these steps:

- change: `version` in `gradle.properties` to desired one (non-snapshot, to be released one),

- update: `docs/releasenotes.md` to reflect next version, it's release date and release notes,
- change: `version` as well as `release` in `docs/conf.py` to desired version to be released,
- push the changes to master branch of the p6spy repo,
- wait for the green build (in Travis CI), fix problems if necessary,
- perform release (via github pages `releases` -> `Draft new release`), for tag version, use prefix `p6spy-` the version should be named without the prefix,
- after green build performed by Travis CI on tag update the `version` in `gradle.properties` to next snapshot one,
- change: `version` in `docs/conf.py` to next snapshot one and
- push the changes to master branch of the p6spy repo.

Released artifacts should be afterwards present in the bintray and with a delay of approximatelly 24 hours also in the: maven central.

7.5 Running the tests

To run the JUnit tests against specific database(s):

1. Please note, that PostgreSQL, MySQL, Firebird and Oracle specific tests require to have the detabase servers running with the specific databases, users and permissions setup (see: *Integration tests-like environment with Docker Compose* section).
2. Moreover as the Oracle jdbc drivers are not publicly available in maven repositories, however can be copied from running docker container used for Oracle DB testing.

By default, tests run against H2 database. To enable other databases, make sure to setup environment variable DB to one of the:

- PostgreSQL
- MySQL
- H2
- HSQLDB
- SQLite
- Firebird
- Derby
- Oracle
- or comma separated list of these

7.5.1 Running the tests in the command line

use the following maven command:

```
./gradlew test -DDB=<DB_NAMES>
```

where `<DB_NAMES>` would hold the value of `DB` environment variable described before.

7.5.2 Running the tests in Eclipse

1. Make sure to have buildship plugin installed
2. Import the p6spy project to eclipse (as Gradle project)
3. Right click the Class holding the test to run and choose: Run As -> JUnit Test

The `DB` environment variable can be set using Arguments tab -> VM Argument of the JUnit Run Configuration.

7.5.3 Integration tests-like environment with Docker compose

It might be tricky to run full batery of tests on developer machine (especially due to need of DB servers setup). To make things easier, Docker with Docker compose is used to create environment close to the one running on our integration test servers ([travis-ci] (https://travis-ci.org/)).

To have tests running please follow these steps:

1. Install [Docker] (https://docs.docker.com/engine/installation/)
2. Install [Docker compose] (https://docs.docker.com/compose/install/) (version proved to be working is: 1.13.0).
3. To run integration tests on your local machine run following:

```
# get p6spy sources
git clone https://github.com/p6spy/p6spy
cd p6spy
# start databases in dockerized environment, please note SQLite installation␣
↪would still have to be done on the machine manually
docker-compose up
# once oracle container is started, run:
mkdir -p ./build/repo && docker cp p6spy_oracle_1:/u01/app/oracle/product/11.2.0/
↪xe/jdbc/lib/ojdbc6.jar ./build/repo
# run tests
./gradlew test -P travis
```

To debug the tests remotely, use the following command:

```
./gradlew test -P travis --debug-jvm
```

1. Afterwards use your favorite java IDE to remotely debug (using port 5005) the tests run.

CHAPTER 8

Thanks

We'd like to thank everyone supporting the project in a various ways, namely:

- contributors, users, testers - without your contribution the project could not exist
- GitHub, Inc - for enabling the collaborative development
- Travis CI GmbH - for donated Continuous Integration resources via travis-ci.org
- SonarSource S.A, Switzerland for donated SonarQube hosting via nemo.sonarqube.org
- Yourkit LLC - for kindly supporting p6spy open source project with its full-featured Java Profiler. YourKit, LLC is the creator of innovative and intelligent tools for profilingJava and .NET applications. Take a look at YourKit's leading software products: YourKit Java Profiler and YourKit .NET Profiler.
- and many other companies and individuals which would be too many to list here
- moreover we'd like to thank you, for showing interest in our project!