High-Performance Java Persistence

LOGGING

VLAD MIHALCEA

Agenda

- Why do we need logging?
- How to log statements with Hibernate?
- Beyond Hibernate logging
  - P6Spy
  - Datasource-proxy

Why do we need statement logging?

- If statements are automatically generated by the data access framework, you definitely need to log them.
- When a business logic is implemented, the Definition of Done should include a review of all the associated data access layer operations.
- Following this rule can save a lot of hassle when the enterprise system is deployed into production.

Hibernate logging configuration properties

Configuration property	Description
`hibernate.show_sql`	Prints SQL statements to the console. Should be avoided.
`hibernate.format_sql`	Formats SQL statements before being logged or printed to the console
`hibernate.use_sql_comments`	Add comments to the automatically generated SQL statement

Using a logging framework

- Hibernate uses the JBoss Logging library which, just like SLF4J, acts as a logging bridge.
- Therefore, you can use any logging framework you want like:
  - Log4j or Log4j 2
  - Logback via SLF4J
  - Java Util Logging

```
<logger name="org.hibernate.SQL" level="debug"/>
```

Using a logging framework

```
Post post = new Post();
post.setId(1L);
post.setTitle("Post it!");

entityManager.persist(post);
```

```
insert into post (title, version, id) values (?, ?, ?)
```

- Because Hibernate uses `PreparedStatement`(s) exclusively, the bind parameter values are not available when the statement gets printed into the log.

Hibernate statement logging formatting

```
<property name="hibernate.format_sql" value="true" />
```

- The `hibernate.format_sql` property applies to logged statements only and it doesn't propagate to the underlying JDBC Driver (SQL statements are still sent as single lines of text).
- This way, the statement formatting doesn't have any effect when statements are logged through an external `DataSource` proxy.

Hibernate statement logging formatting

```
insert
   into
      post
      (title, version, id)
   values
      (?, ?, ?)
```

- Bind parameters are still logged separately.

Hibernate logging statement comments

```
<property name="hibernate.use_sql_comments" value="true" />
```

- Hibernate can explain the statement generation process by appending SQL-level comments into the statement body.
- This feature allows the application developer to get a better understanding of the following processes:
  - the entity state transition that triggered the current executing statement
  - the reason for choosing a join when fetching a given result set
  - the explicit locking mechanism employed by the current statement

Hibernate Logging Statement comments

```
/* insert com.vladmihalcea.book.hpjp.entities.Post */
insert into post (title, version, id) values (?, ?, ?)
```

- As opposed to SQL statement formatting, SQL comments are not only generated during logging, being propagates to the underlying JDBC `Driver` as well.
- Although it might be a useful technique for debugging purposes, in a production environment, it's better to leave it disabled, to reduce the database request networking overhead.

Logging statement parameters

```
<logger name="org.hibernate.type.descriptor.sql"
        level="trace"/>
```

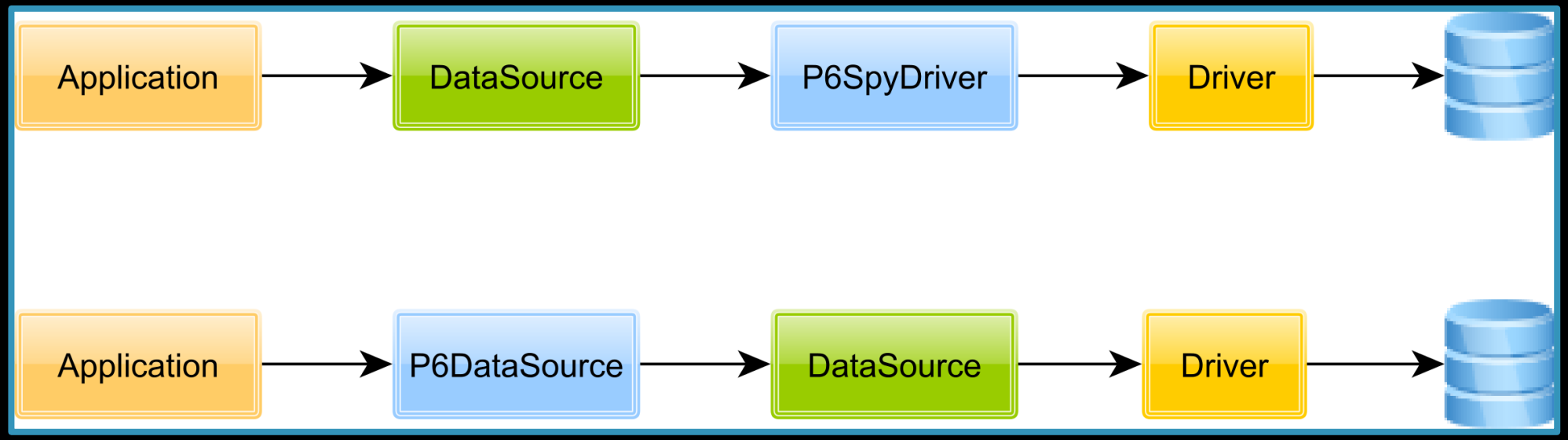
```
DEBUG: o.h.SQL - insert into post (title, version, id) values (?, ?, ?)

TRACE : o.h.t.d.s.BasicBinder - binding parameter [1] as [VARCHAR] - [Post it!]
TRACE : o.h.t.d.s.BasicBinder - binding parameter [2] as [INTEGER] - [0]
TRACE : o.h.t.d.s.BasicBinder - binding parameter [3] as [BIGINT] - [1]
```

Logging statement parameters

- The most straight-forward way of logging SQL statements along with their runtime bind parameter values is to use an external `JDBC statement` proxy.
- Either the JDBC `Driver` or the `DataSource` can be proxied to intercept statement executions and log them along with the actual parameter values.
- Besides statement logging, a JDBC proxy can provide other cross-cutting features like detecting long-running statements or statement count validators.

P6Spy


Application
DataSource
P6SpyDriver
Driver
Application
P6DataSource
DataSource
Driver

P6Spy

Column name	Description
Timestamp	The statement execution timestamp
Execution time	The statement execution duration (milliseconds)
Category	The current statement category (e.g. statement, batch)
Connection	The database connection identifier (as assigned by P6Spy)
Original statement	The original statement that was intercepted by P6Spy
Formatted statement	The statement with all parameter placeholders replaced with the actual bind values

P6Spy

```
Post post = new Post();
post.setId(1L);
post.setTitle("Post it!");

entityManager.persist(post);
```

```
p6spy - 1504269774595|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post it!', 0, 1)
p6spy - 1504269774597|0|commit|connection 7 ||
```

P6Spy

```
<property name="hibernate.jdbc.batch_size" value= " 5" />
```

```
for ( long i = 0; i < 3; i++ ) {
    Post post = new Post();
    post.setId(i);
    post.setTitle(String.format("Post no. %d", i));

    entityManager.persist(post);
}
```

P6Spy

```
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 0', 0, 0)
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 1', 0, 1)
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 2', 0, 2)
p6spy - 1448122491812|5|statement|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 2', 0, 2)
```

P6Spy

```
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 0', 0, 0)
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 1', 0, 1)
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 2', 0, 2)
p6spy - 1448122491812|5|statement|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 2', 0, 2)
```

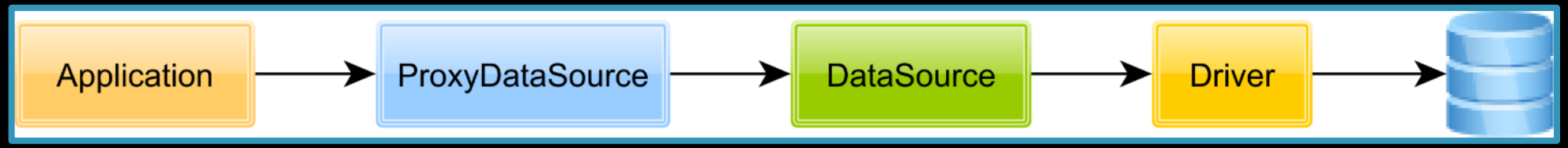
P6Spy

```
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 0', 0, 0)
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 1', 0, 1)
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 2', 0, 2)
p6spy - 1448122491812|5|statement|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 2', 0, 2)
```

P6Spy

```
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 0', 0, 0)
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 1', 0, 1)
p6spy - 1448122491807|0|batch|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 2', 0, 2)
p6spy - 1448122491812|5|statement|connection 7|
    insert into post (title, version, id) values (?, ?, ?)|
    insert into post (title, version, id) values ('Post no. 2', 0, 2)
```

DataSource-proxy




```
@Bean
public DataSource dataSource() {
    SLF4JQueryLoggingListener loggingListener =
        new SLF4JQueryLoggingListener();
    loggingListener.setQueryLogEntryCreator(
        new InlineQueryLogEntryCreator());
    return ProxyDataSourceBuilder.create(actualDataSource())
        .name(DATA_SOURCE_PROXY_NAME)
        .listener(loggingListener)
        .build();
}
```

DataSource-proxy

- In Java EE, not all application servers allow configuring an external `DataSource`, as they rely on their own custom implementations that bind the user supplied JDBC `Driver`.
- Because it can only decorate a `DataSource`, datasource-proxy might not be suitable in all Java EE environments.
- It provides support for custom JDBC statement execution listeners.

DataSource-proxy

```
Post post = new Post();
post.setId(1L);
post.setTitle("Post it!");

entityManager.persist(post);
```

```
Name:DATA_SOURCE_PROXY, Time:0, Success:True,
Type:Prepared, Batch:False, QuerySize:1, BatchSize:0,
Query:["insert into post (title, version, id) values (?, ?, ?)"],
Params:[(Post it!. 0, 1)]
```

DataSource-proxy

```
for ( long i = 0; i < 3; i++ ) {
    Post post = new Post();
    post.setId(i);
    post.setTitle(String.format("Post no. %d", i));

    entityManager.persist(post);
}
```

```
Name:DATA_SOURCE_PROXY, Time:6, Success:True,
Type:Prepared, Batch:True, QuerySize:1, BatchSize:3,
Query:["insert into post (title, version, id) values (?, ?, ?)"],
Params:[(Post no. 0, 0, 0), (Post no. 1, 0, 1), (Post no. 2, 0, 2)]
```

DataSource-proxy

```
for ( long i = 0; i < 3; i++ ) {
    Post post = new Post();
    post.setId(i);
    post.setTitle(String.format("Post no. %d", i));

    entityManager.persist(post);
}
```

```
Name:DATA_SOURCE_PROXY, Time:6, Success:True,
Type:Prepared, Batch:True, QuerySize:1, BatchSize:3,
Query:["insert into post (title, version, id) values (?, ?, ?)"],
Params:[(Post no. 0, 0, 0), (Post no. 1, 0, 1), (Post no. 2, 0, 2)]
```