PG для Java-разработчиков

Яндекс Маркет

Сергей Вальков
Lead Java-developer в Яндекс Маркете

whoami

Tg @SergioRussia
LinkedIn https://www.linkedin.com/in/sergeyvalkov
GitHub https://github.com/sergiorussia

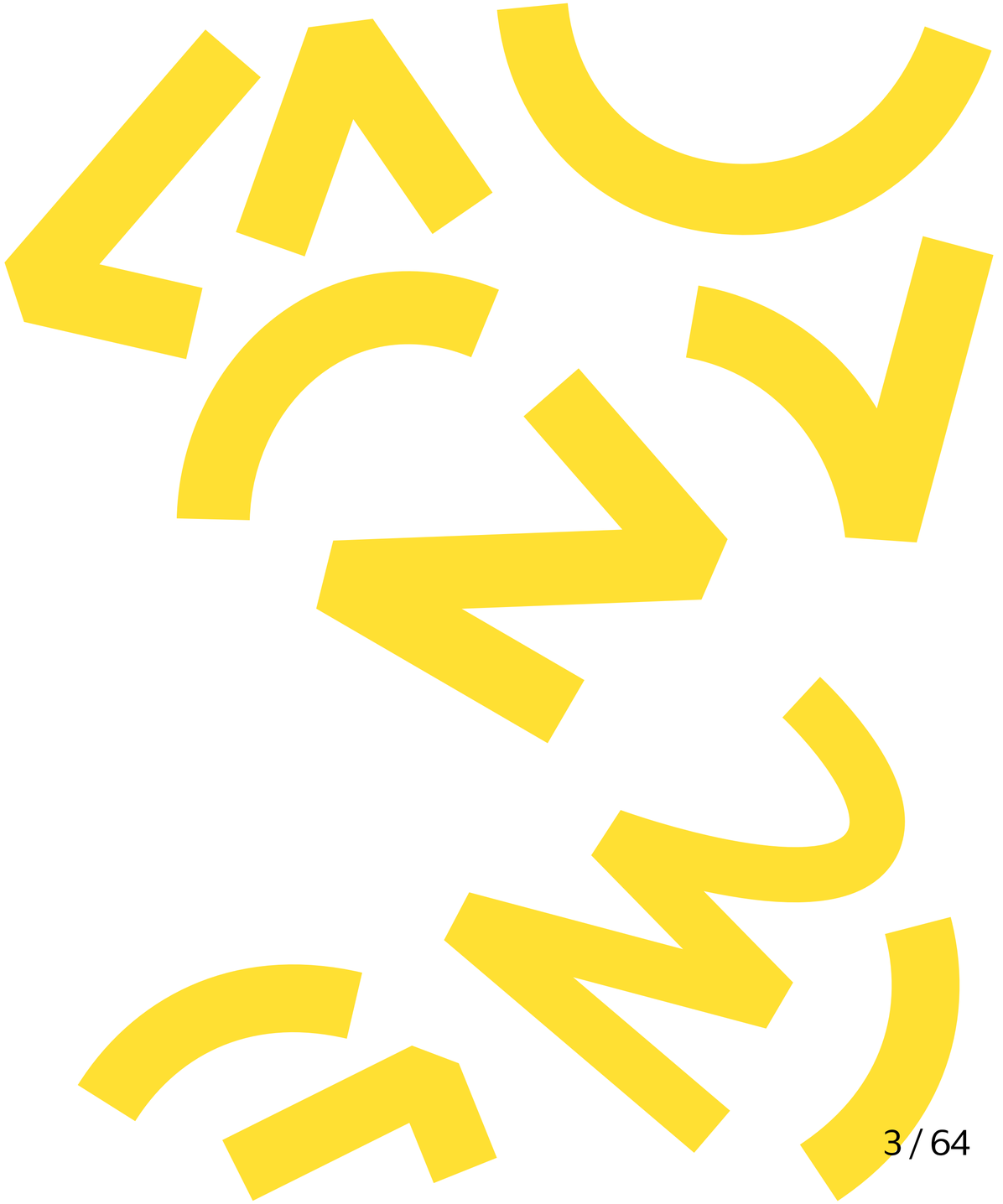
Сергей Вальков

1. Lead Java-developer в Яндекс Маркете
2. Пишу код с 2009 года
3. Писал desktop приложения, ERP и BI системы, последние лет 8 в backend-разработке
4. Писал на C++, C#, Go, Java, Kotlin, Objective-C, Python, Scala

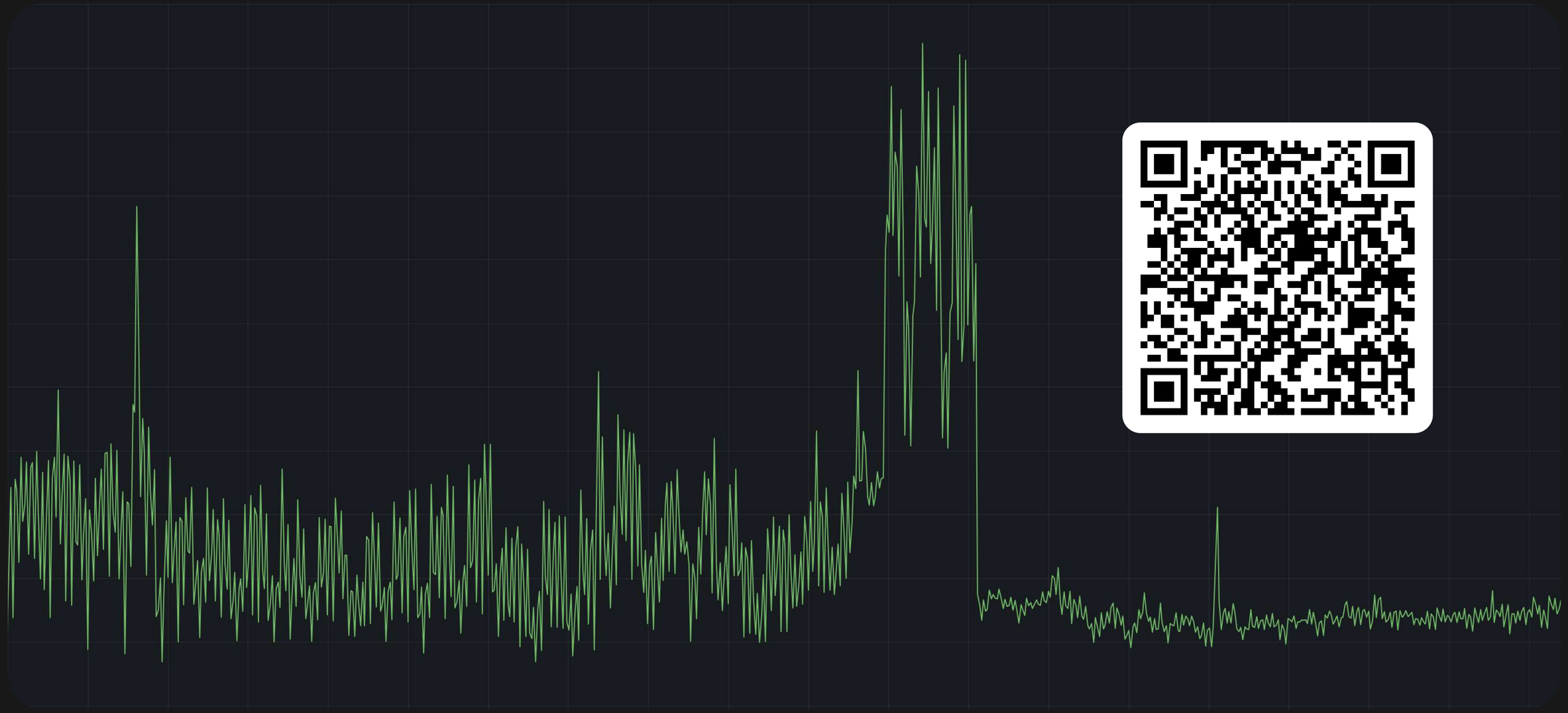
План

pgJDBC + HikariCP + Spring JDBC

1. Подключаемся
2. Мониторим
3. Ускоряем
4. Итоговый конфиг



Предыстория



https://habr.com/ru/companies/yandex/articles/801415

Disclaimer

1. Все сказанное далее проверялось и тестировалось в бою на следующих версиях:

   `org.postgresql:postgresql:42.2.24` `com.zaxxer:HikariCP:5.0.0`

   `org.springframework:spring-jdbc:5.1.6.RELEASE`

   Не расстраивайтесь раньше времени, good news ahead!

2. PGJDBC-NG не используем, в докладе не рассматриваю

3. JPA/Hibernate не используем, в докладе не рассматриваю

4. Используем HikariCP:
   - Пул коннектов для JDBC
   - Альтернатива c3p0, Tomcat, Apache DBCP и прочим
   - Выбран по-умолчанию в Spring Boot 2 и 3

01 Подключаемся

Timeouts?

Пусть у нас есть несколько равнозначных хостов:

```
jdbc:postgresql://~~host1:port1~~,host2:port2,host3:port3/db?params...
```

host1 умер - сервис не поднимается:

```
Exception in thread "main" org.postgresql.util.PSQLException:
Connection attempt timed out
```

Что не так с `HikariConfig.setConnectionTimeout(10000)` ?

Timeouts?

...следы ведут в `HikariPool.checkFailFast()`. Что делать?

Вариант 1:

`HikariConfig.setInitializationFailTimeout(-1)`

Отключит валидацию, но упадет первое же обращение к базе, поэтому не сто́ит

Вариант 2:

Поднять `HikariConfig.connectionTimeout`.

Но на сколько и почему?
Хватит ли 30 секунд по-умолчанию?

Timeouts?

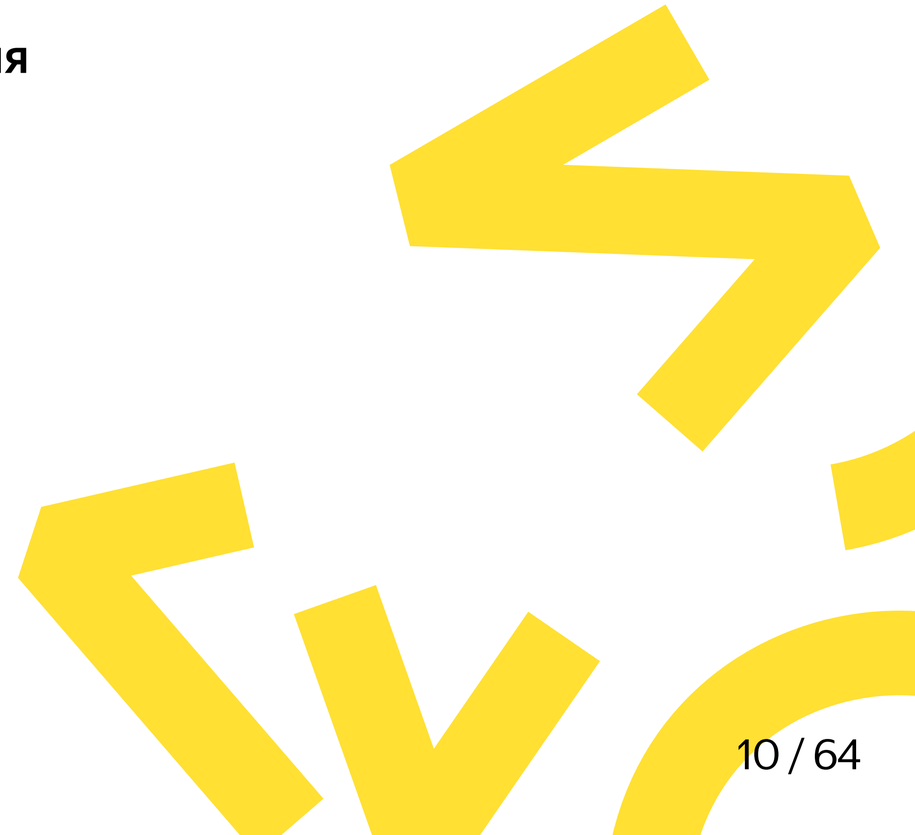
Приходите к нам в Java, у нас есть:

- `PGProperty.CONNECT_TIMEOUT`
- `PGProperty.LOGIN_TIMEOUT`
- `HikariConfig.connectionTimeout` (connect, connection... - почти все буквы те же)
- и еще пачка других таймаутов, см `PGProperty.***_TIMEOUT`
  и `HikariConfig.***Timeout`, но сегодня речь не про них

Timeouts?

Получение соединения из pgJDBC:

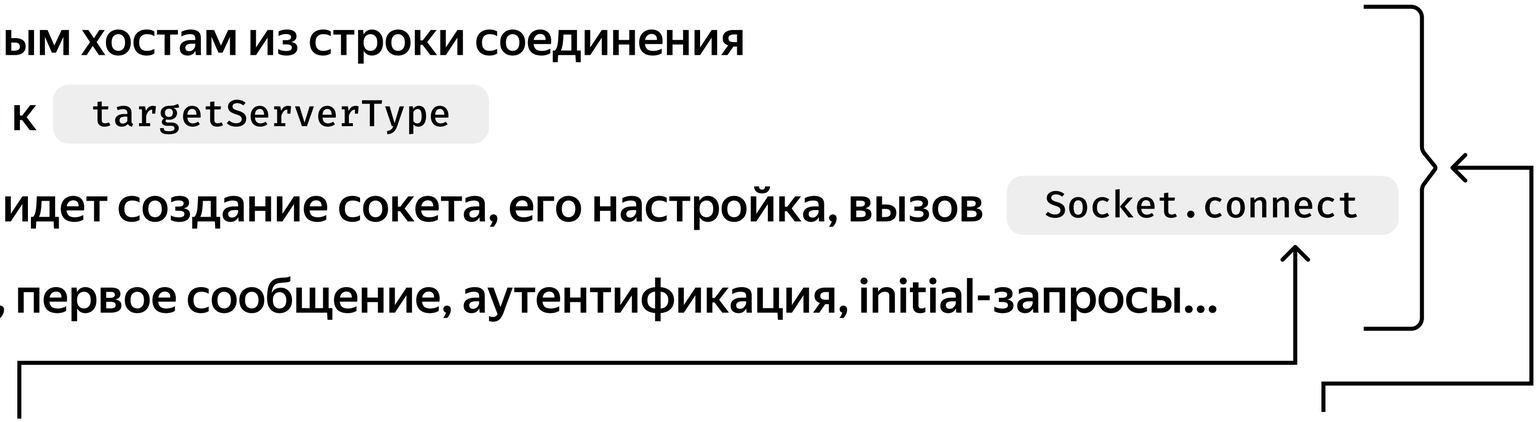
- Цикл по указанным хостам из строки соединения до подключения к `targetServerType`
- В цикле первым идет создание сокета, его настройка, вызов `Socket.connect`
- SSL handshake, если необходим
- Отправка первого сообщения с базовыми параметрами соединения
- Аутентификация пользователя
- Отправка initial `SET` — запросов



Timeouts?

Получение соединения из pgJDBC:

- Цикл по указанным хостам из строки соединения до подключения к `targetServerType`
- В цикле первым идет создание сокета, его настройка, вызов `Socket.connect`
- ...SSL handshake, первое сообщение, аутентификация, initial-запросы...



`PGProperty.CONNECT_TIMEOUT` :

- Таймаут на `Socket.connect` к **одному** хосту
- По-умолчанию **10** секунд, указан в секундах

`PGProperty.LOGIN_TIMEOUT` :

- **общий** таймаут на получение соединения из pgJDBC, на весь цикл
- по-умолчанию 0 секунд (бесконечный), указан в секундах

Timeouts?

Получение соединения из HikariCP:

- При создании пула для первого подключения `HikariConfig.connectionTimeout` **не участвует,** синхронно выполняется `DataSource.getConnection`
- В остальных случаях попытка дождаться соединения из connection bag за `HikariConfig.connectionTimeout`
- Если дождались — донастройка `Connection` , выполнение `HikariConfig.connectionInitSql` и т.д.

Connection bag заполняется асинхронно в фоне, вызывая `DataSource.getConnection` , `HikariConfig.connectionTimeout` участвует опосредованно для расчета backoff

Timeouts?

`HikariConfig.connectionTimeout` :

- По-умолчанию 30 секунд, указан в миллисекундах
- Перезатирает `DataSource.loginTimeout` !

```
// примерная логика PoolBase::setLoginTimeout
if (connectionTimeout != 0 && connectionTimeout != Integer.MAX_VALUE) {
    dataSource.setLoginTimeout(max(1, (connectionTimeout + 500) / 1000));
}
```

PS: если указывать `HikariConfig.jdbcUrl` (Spring Boot),
то `HikariConfig.connectionTimeout` не влияет на `PGProperty.LOGIN_TIMEOUT` ,
а pgJDBC сперва берет таймаут именно из `PGProperty.LOGIN_TIMEOUT`

Timeouts?

Вернемся к проблеме с багажом знаний:

`jdbc:postgresql://~~host1:port1~~,host2:port2,host3:port3/db?params...`

host1 умер — сервис не поднимается, connection timeout.

Причина теперь ясна:

- У нас не Spring Boot, указали `HikariConfig.driverClassName` (рекомендованный способ), а не `HikariConfig.jdbcUrl`
- Указали `HikariConfig.connectionTimeout` = 10 000
- Отсюда `PGProperty.LOGIN_TIMEOUT` = 10 = `PGProperty.CONNECT_TIMEOUT`
- Не можем добраться до host2 или host3, **застреваем на host1**

Timeouts

Решение:

- `PGProperty.CONNECT_TIMEOUT` > 0 , в зависимости от вашего окружения, по-умолчанию 10
- `PGProperty.LOGIN_TIMEOUT` ≥ (# hosts) * (`PGProperty.CONNECT_TIMEOUT` + epsilon), либо = 0, если готовы ждать до упора
- `HikariConfig.connectionTimeout` = `PGProperty.LOGIN_TIMEOUT` * 1000

Timeouts — проверяем

Имитируем проблему, но еще хуже. Два полуживых хоста localhost:5432 и 6543, и живой 7654:

```
docker run -d -p7654:5432 -e POSTGRES_HOST_AUTH_METHOD=trust postgres:14
zmodload zsh/net/tcp # модуль в zsh
ztcp -l 5432 && listenfd5432=$REPLY # просто слушаем порт
ztcp -l 6543 && listenfd6543=$REPLY # просто слушаем порт
```

```
var url = "jdbc:postgresql://localhost:5432,localhost:6543,localhost:7654/postgres";
var ds = new HikariDataSource();
ds.setUsername("postgres");
ds.setDataSourceClassName("org.postgresql.ds.PGSimpleDataSource");
ds.addDataSourceProperty("url", url);
ds.setConnectionTimeout(20000); // компенсируем ухудшение условий
var t = System.currentTimeMillis();
try (var con = ds.getConnection()) { /* noop */ } finally {
    System.out.println(System.currentTimeMillis() - t + " ms");
}
```

Примерный результат:

```
21062 ms
Exception in thread "main" org.postgresql.util.PSQLException: Connection attempt timed out.
```

Timeouts — проверяем

Тот же сетап, но корректно применим таймауты:

```
var url = "jdbc:postgresql://localhost:5432,localhost:6543,localhost:7654/postgres";
var ds = new HikariDataSource();
ds.setJdbcUrl(url); // вместо dataSourceClassName для имитации поведения Spring Boot
ds.setUsername("postgres");
ds.addDataSourceProperty("connectTimeout", 10); // выставляем явно
ds.addDataSourceProperty("loginTimeout", 33); // (3 хоста) * (connectTimeout + 1s запас)
ds.setConnectionTimeout(33000); // loginTimeout * 1000
var t = System.currentTimeMillis();
try (var con = ds.getConnection()) { /* noop */ } finally {
    System.out.println(System.currentTimeMillis() - t + " ms");
}
```

Примерный результат, успех:

```
20156 ms
```

Timeouts — нюансы

Тот же сетап с корректными таймаутами, возьмем 3 соединения сразу:

```
var t = System.currentTimeMillis();
try (var con1 = ds.getConnection(); var con2 = ds.get...; var con3 = ds.get...) {
    System.out.println(System.currentTimeMillis() - t + " ms");
}
```

Примерный результат:

```
... [HikariPool-1 connection adder] ... Added connection org.postgresql.jdbc.PgConnection...
... [HikariPool-1 connection adder] ... Added connection org.postgresql.jdbc.PgConnection...
20035 ms
```

Не великоват ли таймаут?

Timeouts — нюансы

В pgJDBC есть `GlobalHostStatusTracker`, который запоминает состояние хостов, можно ему немного помочь:

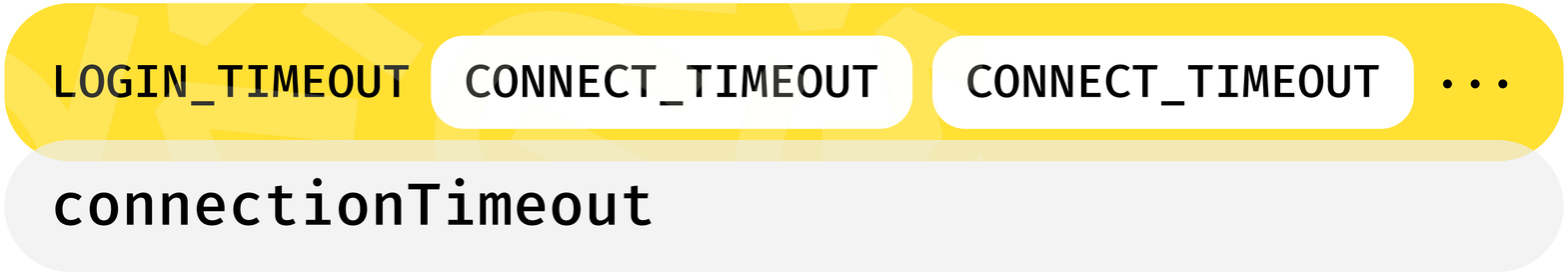
выставить `PGProperty.HOST_RECHECK_SECONDS` > `PGProperty.CONNECT_TIMEOUT`, т.к. по-умолчанию он тоже 10 секунд

```
ds.addDataSourceProperty("hostRecheckSeconds", 11); // > connectTimeout
```

Примерный результат после настройки:

```
8 ms
```

Timeouts — резюме



1. `PGProperty.CONNECT_TIMEOUT` — это таймаут на `Socket.connect` к **одному** хосту
2. `PGProperty.LOGIN_TIMEOUT` — это общий таймаут на получение соединения,
   делайте ≥ (# hosts) * (`PGProperty.CONNECT_TIMEOUT` + epsilon),
   либо = 0, если готовы ждать до упора
3. `HikariConfig.connectionTimeout` = `PGProperty.LOGIN_TIMEOUT` * 1000,
   иначе значение по-умолчанию может перезатереть `PGProperty.LOGIN_TIMEOUT`

PS: `PGProperty.HOST_RECHECK_SECONDS` > `PGProperty.CONNECT_TIMEOUT`

Config?

application.properties:

```
spring.datasource.hikari.connection-timeout=33000 # loginTimeout * 1000
spring.datasource.hikari.data-source-properties.connectTimeout=10 # указываем явно
spring.datasource.hikari.data-source-properties.hostRecheckSeconds=11 # > connectTimeout
spring.datasource.hikari.data-source-properties.loginTimeout=33 # 3 hosts * (connectTimeout + epsilon)
```

Load balance?

А если хотим читать из всех реплик, по возможности распределяя нагрузку?

```
docker run -d -p5432:5432 -e POSTGRES_HOST_AUTH_METHOD=trust postgres:14
docker run -d -p6543:5432 ...
docker run -d -p7654:5432 ...
```

```
var url = "jdbc:postgresql://localhost:5432,localhost:6543,localhost:7654/postgres
    ?user=postgres&targetServerType=preferSecondary";
try (var con = DriverManager.getConnection(url)) {
    System.out.println(con.unwrap(PgConnection.class).getQueryExecutor().getHostSpec());
} // и еще несколько раз...
```

Из коробки pgJDBC перебирает хосты в порядке, как указано в connection string.
Будем каждый раз получать один и тот же хост, вся нагрузка уйдет на него:

```
localhost:5432
localhost:5432
localhost:5432
...
```

Load balance

`PGProperty.LOAD_BALANCE_HOSTS`:

- `jdbc:postgresql://.../db?loadBalanceHosts=true`
- `pgSimpleDataSource.setLoadBalanceHosts(true)`
- `spring.datasource.hikari.data-source-properties.loadBalanceHosts=true`

Рандомизирует (`Collections.shuffle`) порядок обхода хостов-кандидатов в упомянутом ранее цикле подключения. Кандидаты выбираются в зависимости от `targetServerType` .

Используйте, если все хосты равнозначны, и нет требований к порядку подключения.

PS: помним про `GlobalHostStatusTracker` в pgJDBC

Load balance — проверяем

```
// те же три хоста...
var url = "jdbc:postgresql://localhost:5432,localhost:6543,localhost:7654/postgres
    ?user=postgres&targetServerType=preferSecondary
    &loadBalanceHosts=true";
// так же три раза подряд получаем соединение...
```

Можно получить, например, такой результат:

```
localhost:6543
localhost:5432
localhost:7654
```

Config?

application.properties:

```
spring.datasource.hikari.connection-timeout=33000
spring.datasource.hikari.data-source-properties.connectTimeout=10
spring.datasource.hikari.data-source-properties.hostRecheckSeconds=11
spring.datasource.hikari.data-source-properties.loginTimeout=33
spring.datasource.hikari.data-source-properties.loadBalanceHosts=true # если порядок хостов не важен
```

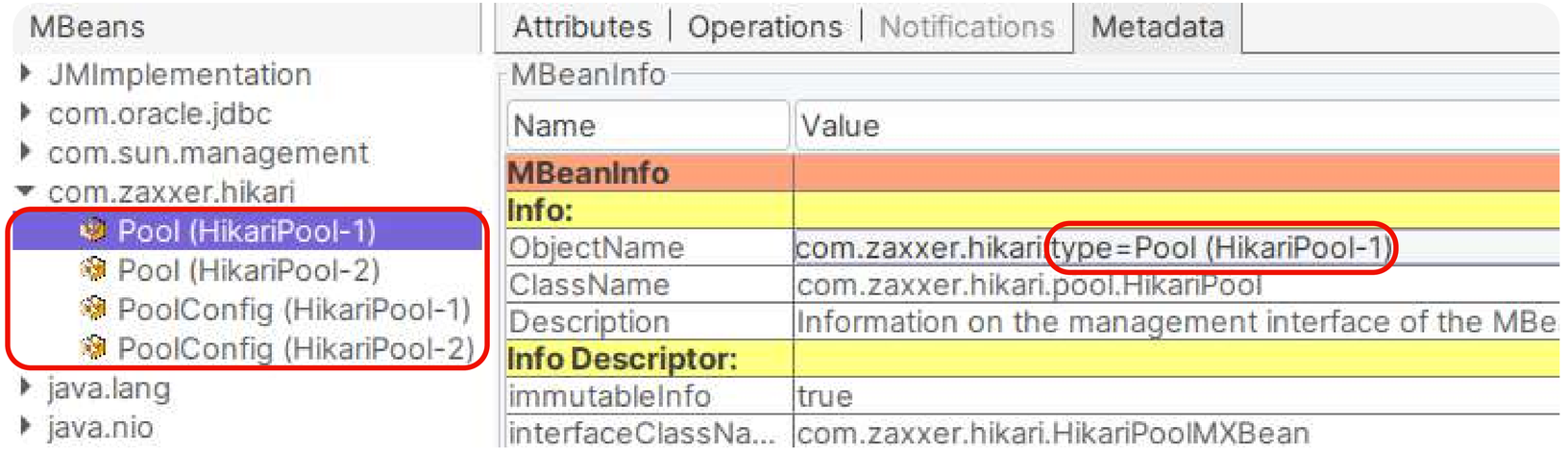
02 Мониторим

JMX? Pool name?

- `HikariConfig.setRegisterMbeans(true)`
- `spring.datasource.hikari.register-mbeans=true`

```
var configRW = new HikariConfig();
configRW.setJdbcUrl("jdbc:postgresql://dummy/db");
configRW.setInitializationFailTimeout(-1); // не подключаться на старте
configRW.setRegisterMbeans(true); // включаем регистрацию в JMX
var configRO = new HikariConfig();
configRW.copyStateTo(configRO);
var dsRW = new HikariDataSource(configRW);
var dsRO = new HikariDataSource(configRO);
// идем смотреть в visualvm...
```

Где какой пул?



JMX, pool name

Pool name:

- `HikariConfig.setPoolName("poolName")`
- `spring.datasource.hikari.pool-name=poolName`
- будет в логах, в JMX

JMX:

- `-Dhikaricp.jmx.register2.0`
- пока не документирована нигде, кроме changelog
- будут стандартные `ObjectName` в JMX

JMX, pool name — проверяем

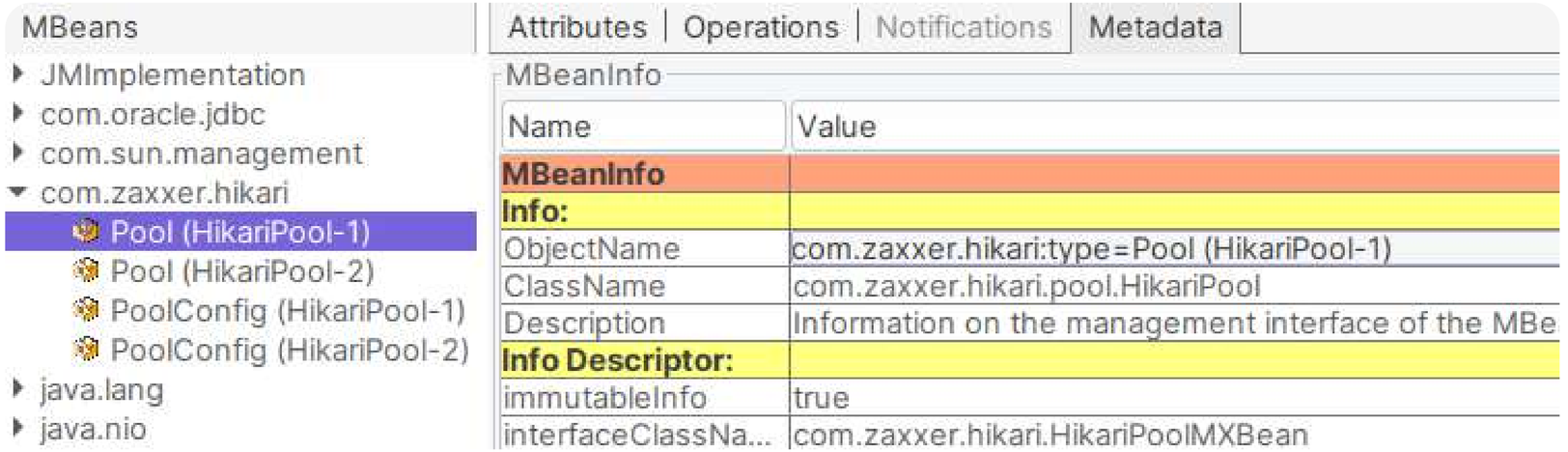
```
// -Dhikaricp.jmx.register2.0=true
...
configRW.setPoolName("pool-read-write");
configRO.setPoolName("pool-read-only");
configRO.setMaxLifetime(1); // случайная кривая настройка, чтобы получить ошибку в логе
...
System.out.println(Thread.getAllStackTraces()...map(Thread::getName)...);
// идем смотреть в visualvm...
```

Теперь понятно, где какой пул:

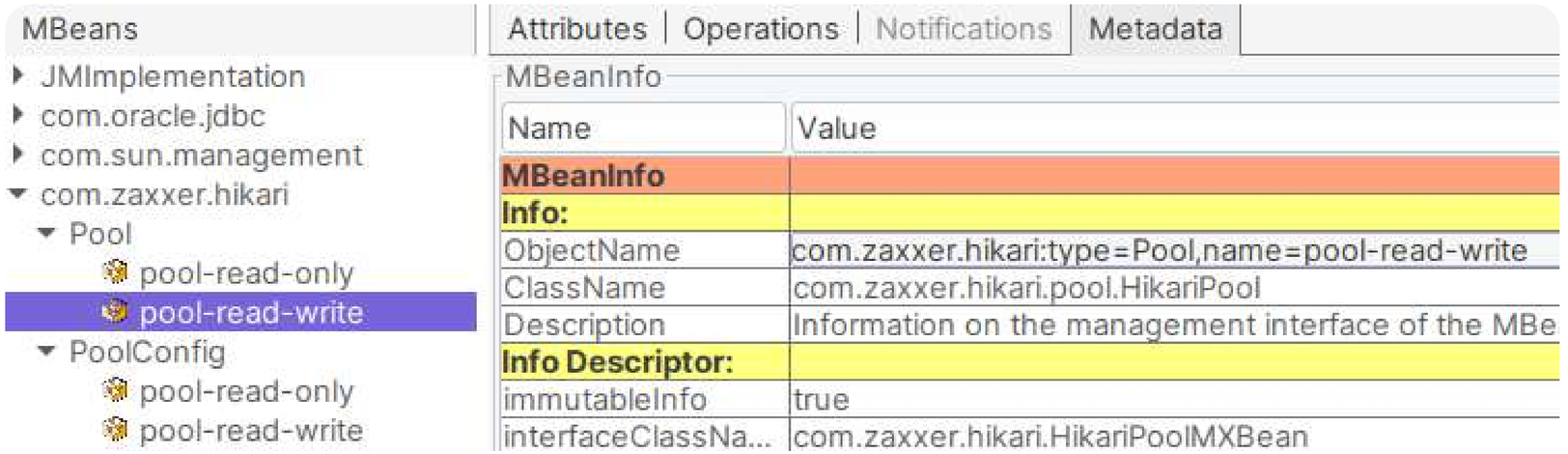
```
15:35:18.235 ... HikariConfig : pool-read-only - maxLifetime is less than...
[..., pool-read-write housekeeper, pool-read-only housekeeper, ...]
```

JMX, pool name — проверяем

без `-Dhikaricp.jmx.register2.0=true` :



с `-Dhikaricp.jmx.register2.0=true` :



Config?

application.properties:

```
spring.datasource.hikari.connection-timeout=33000
spring.datasource.hikari.data-source-properties.connectTimeout=10
spring.datasource.hikari.data-source-properties.hostRecheckSeconds=11
spring.datasource.hikari.data-source-properties.loginTimeout=33
spring.datasource.hikari.data-source-properties.loadBalanceHosts=true
spring.datasource.hikari.pool-name=poolName
spring.datasource.hikari.register-mbeans=true
```

sys props:

```
-Dhikaricp.jmx.register2.0=true
```

Application name?

Тормозят компоненты, но какие?

```
select pid, usename, client_addr, state, application_name, now()-query_start as duration, query
from pg_stat_activity where client_addr is not null and state != 'idle';
```

pid	username	client_addr	state	application_name	duration	query
217	user-rw	127.0.0.1	idle in transaction	PostgreSQL JDBC Driver	0.070176 secs	`select 1...`
737	user-rw	127.0.0.1	idle in transaction	PostgreSQL JDBC Driver	0.024577 secs	`select 2...`
603	user-rw	127.0.0.1	active	PostgreSQL JDBC Driver	5.427278 secs	`select 3...`
672	user-rw	127.0.0.1	idle in transaction	PostgreSQL JDBC Driver	**16.339994 secs**	`select 4...`
043	user-rw	127.0.0.1	idle in transaction	PostgreSQL JDBC Driver	**7.350931 secs**	`select 4...`
958	sergeyvalkov	127.0.0.1	active	sergeyvalkov	0.0 secs	`select pid, ...`

Application name

`PGProperty.APPLICATION_NAME`:

- `jdbc:postgresql://.../db?ApplicationName=appName`, с большой буквы
- `pgSimpleDataSource.setApplicationName("appName")`
- `spring.datasource.hikari.data-source-properties.ApplicationName=appName`

См. `pg_stat_activity.application_name` в PostgreSQL 9.0+

Application name — проверяем

Задаем в каждом компоненте свой `ApplicationName`, результат:

pid	username	client_addr	state	application_name	duration	query
819	user-rw	127.0.0.1	active	**JOBS 2 - IP**	**43.045039 secs**	`update 1...`
648	user-rw	127.0.0.1	active	API 1 - IP	0.050923 secs	`select 1...`
091	user-rw	127.0.0.1	active	**API 2 - IP**	**10.112919 secs**	`select 2...`
380	user-rw	127.0.0.1	active	API 1 - IP	0.052178 secs	`select 3...`
842	user-rw	127.0.0.1	active	API 1 - IP	0.003548 secs	`select 4...`
322	user-rw	127.0.0.1	idle in transaction	JOBS 1 - IP	0.01005 secs	`select 5...`
161	user-rw	127.0.0.1	idle in transaction	JOBS 1 - IP	0.002398 secs	`update 2...`
471	user-ro	127.0.0.1	active	JOBS 1 - IP	0.05833 secs	`select 6...`
986	sergeyvalkov	127.0.0.1	active	sergeyvalkov	0.0 secs	`select pid,...`

Application name — нюансы

`PGProperty.ASSUME_MIN_SERVER_VERSION`:

- `jdbc:postgresql://.../db?assumeMinServerVersion=10.0`, 10.0 для примера
- `pgSimpleDataSource.setAssumeMinServerVersion("10.0")`
- `spring.datasource.hikari.data-source-properties.assumeMinServerVersion=10.0`
- имеет смысл указывать значения ≥ 9.0
- подсказка драйверу, ускорение создания соединения за счёт отстуствия initial запросов вроде `SET application_name`

`pg_stat_activity` до, `jdbc:postgresql...&ApplicationName=appName`:

pid	state	application_name	query
73	idle	appName	`SET application_name = 'appName'`

после, `jdbc:postgresql...&ApplicationName=appName&assumeMinServerVersion=10.0`:

pid	state	application_name	query
77	idle	**appName**	**нет initial запросов**

Application name — нюансы

Flashback, получение соединения из pgJDBC:

- цикл по указанным хостам...
- ...создание сокета...
- SSL handshake...
- отправка первого сообщения ...,
  в том числе application name, если был указан `assumeMinServerVersion` ≥ 9.0
- аутентификация пользователя
- отправка initial `SET` — запросов, **если не был указан** `assumeMinServerVersion` ≥ 9.0

Экономия может быть до 5-10 миллисекунд на создание одного соединения, YMMV.

PS: пулер (PgBouncer) в режиме пулинга **транзакций + отсутствие** `assumeMinServerVersion` = включить `PGProperty.GROUP_STARTUP_PARAMETERS`, но лучше и быстрее — `assumeMinServerVersion`

Config?

application.properties:

```
spring.datasource.hikari.connection-timeout=33000
spring.datasource.hikari.data-source-properties.ApplicationName=appName
spring.datasource.hikari.data-source-properties.assumeMinServerVersion=10.0 # 10.0 для примера
spring.datasource.hikari.data-source-properties.connectTimeout=10
spring.datasource.hikari.data-source-properties.hostRecheckSeconds=11
spring.datasource.hikari.data-source-properties.loginTimeout=33
spring.datasource.hikari.data-source-properties.loadBalanceHosts=true
spring.datasource.hikari.pool-name=poolName
spring.datasource.hikari.register-mbeans=true
```

sys props:

```
-Dhikaricp.jmx.register2.0=true
```

03 Ускоряем

Rewrite batched inserts?

Хотим регулярно вставлять много строк в базу.

```
jdbcTemplate.batchUpdate(
    "insert into t(columns) values (?,...)",
    List.of(argsArray, argsArray, argsArray,...)
);
```

`JdbcTemplate.batchUpdate` под капотом использует `PreparedStatement.addBatch/executeBatch` .

На сервер улетает batch, но все равно долго!

Rewrite batched inserts?

https://www.postgresql.org/docs/current/sql-insert.html

В pg есть multirow values insert синтаксис,
можно же переписать самому
на `insert into t(...) values (...), (...), (...)...` ?

Можно, но есть способ лучше...

Rewrite batched inserts

`PgProperty.REWRITE_BATCHED_INSERTS`:

- `jdbc:postgresql://.../db?reWriteBatchedInserts=true`
- `pgSimpleDataSource.setReWriteBatchedInserts(true)`
- `spring.datasource.hikari.data-source-properties.reWriteBatchedInserts=true`

Работает только с `insert` запросами и только с `PreparedStatement.addBatch/executeBatch`.
Поддерживается `insert ... on conflict` (без параметров в `do update set`).

Драйвер сам перепишет `insert into t(...) values (...)...`
на `insert into t(...) values (...), (...), (...)...`

Насколько быстрее? Замеры и другие способы ускорить вставку большого числа строк в pg.
См. в докладе Дмитрия Фатова https://youtu.be/lpFDv50xn30?t=521,
у Дмитрия получилось ускорить на ~15%

Rewrite batched inserts - проверяем

```
// create table t (id int primary key, v1 int, v2 int)
var url = "jdbc:postgresql://localhost:5432/postgres
    ?user=postgres&reWriteBatchedInserts=true";
...
tx.executeWithoutResult(status -> {
    jdbcTemplate.batchUpdate(
        "insert into t(id,v1,v2) values (?,?,?) on conflict (id) do nothing",
        List.of(
            new Object[]{1, 1, 1},
            new Object[]{2, 2, 2}
        )
    ); // идем смотреть pg_stat_activity...
});
```

pid	state	query
322	idle in transaction	`insert into t (id,v1,v2) values ($1,$2,$3),($4,$5,$6) on conflict (id) do nothing`

Rewrite batched inserts - нюансы

Что вернет `PreparedStatement.executeBatch`, если `reWriteBatchedInserts=true`?

```
// create table t (id int primary key, v1 int)
System.out.println(Arrays.toString(jdbcTemplate.batchUpdate(
    // batchUpdate(...) вернет результат ps.executeBatch() as is
    "insert into t(id,v1) values (?,?) on conflict (id) do nothing",
    List.of(new Object[]{1, 1}, ...{2, 2}, ...{3, 3}))));
```

`Statement.executeBatch` javadoc:

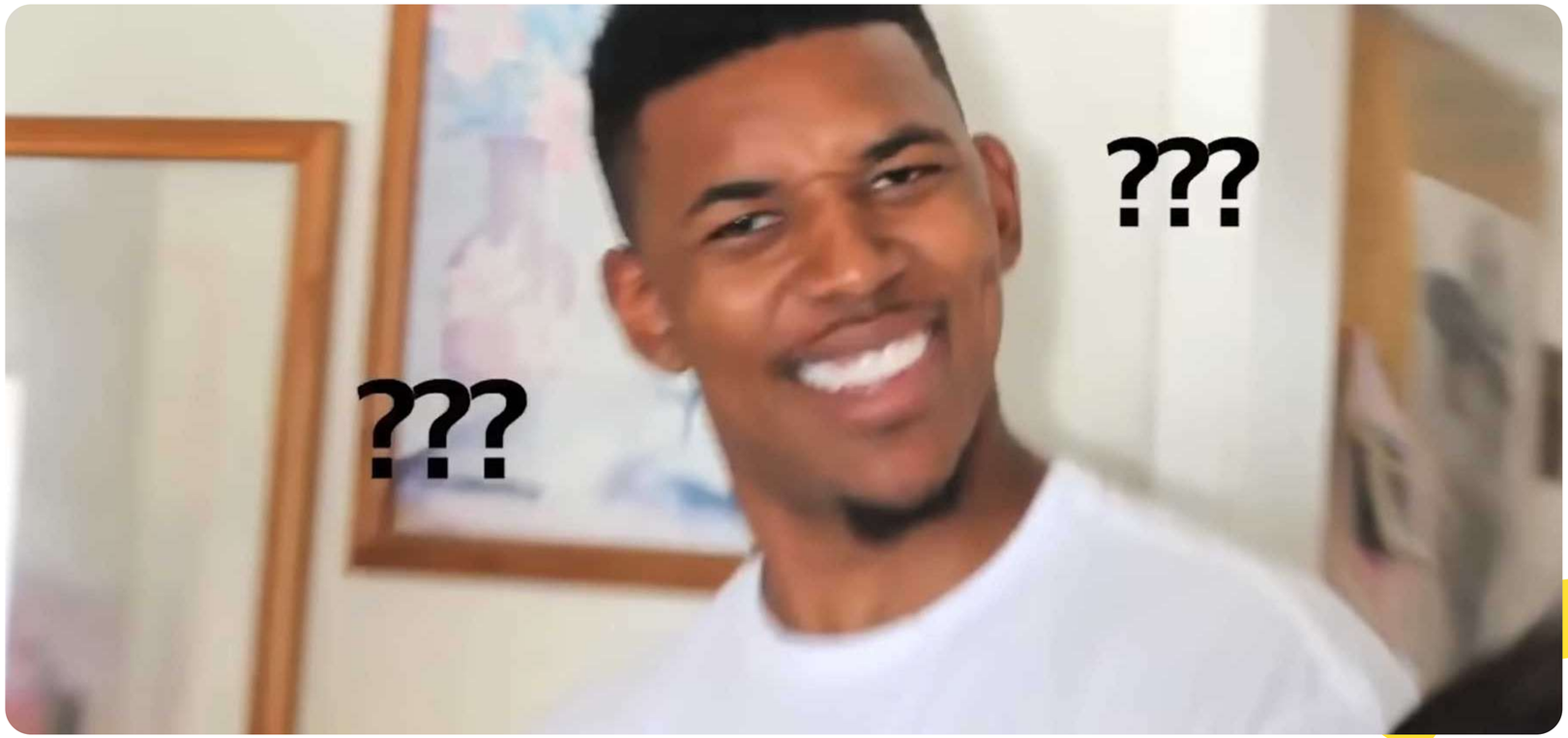
```
@return an array of update counts containing one element for each command in the batch.
The elements of the array are ordered according to the order in which commands were added to the batch.
```

`[1, 1, 1]`?

Rewrite batched inserts - нюансы

```
jdbcTemplate.batchUpdate(
    "insert into t(id,v1) values (?,?) on conflict do nothing",
    List.of(...{1, 1}, ...{2, 2}, ...{3, 3});
```

`[-2, -2, 1]` !



Rewrite batched inserts - нюансы

```
jdbcTemplate.batchUpdate(
    "insert into t(id,v1) values (?,?) on conflict do nothing",
    // List.of(...{1, 1}, ...{2, 2}, ...{3, 3}) = [-2, -2, 1]
    List.of(...{1, 1}, ...{2, 2}, ...{4, 4}));
```

`[-2, -2, 1]` ?

Rewrite batched inserts - нюансы

```
jdbcTemplate.batchUpdate(
    "insert into t(id,v1) values (?,?) on conflict do nothing",
    // List.of(...{1, 1}, ...{2, 2}, ...{3, 3}) = [-2, -2, 1]
    List.of(...{1, 1}, ...{2, 2}, ...{4, 4}));
```

`[0, 0, 1]` !

Rewrite batched inserts - нюансы

```
jdbcTemplate.batchUpdate(
    "insert into t(id,v1) values (?,?) on conflict do nothing",
    // List.of(...{1, 1}, ...{2, 2}, ...{3, 3}) = [-2, -2, 1]
    // List.of(...{1, 1}, ...{2, 2}, ...{4, 4}) = [0, 0, 1]
    List.of(...{1, 1}, ...{2, 2}, ...{4, 4}, ...{5, 5}));
```

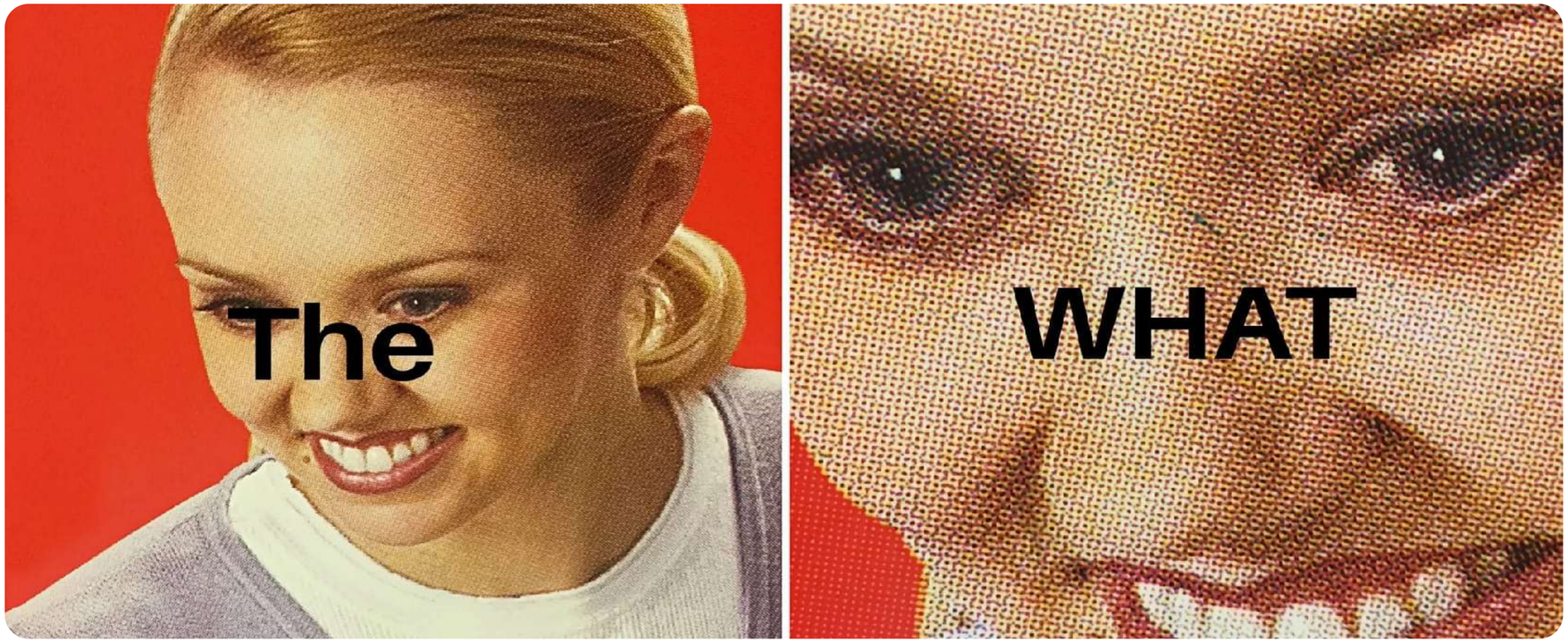
`[0, 0, 0, 1]` ?

`[-2, -2, -2, 1]` ?

Rewrite batched inserts - нюансы

```
jdbcTemplate.batchUpdate(
    "insert into t(id,v1) values (?,?) on conflict do nothing",
    // List.of(...{1, 1}, ...{2, 2}, ...{3, 3}) = [-2, -2, 1]
    // List.of(...{1, 1}, ...{2, 2}, ...{4, 4}) = [0, 0, 1]
    List.of(...{1, 1}, ...{2, 2}, ...{4, 4}, ...{5, 5}));
```

`[-2, -2, -2, -2]` !

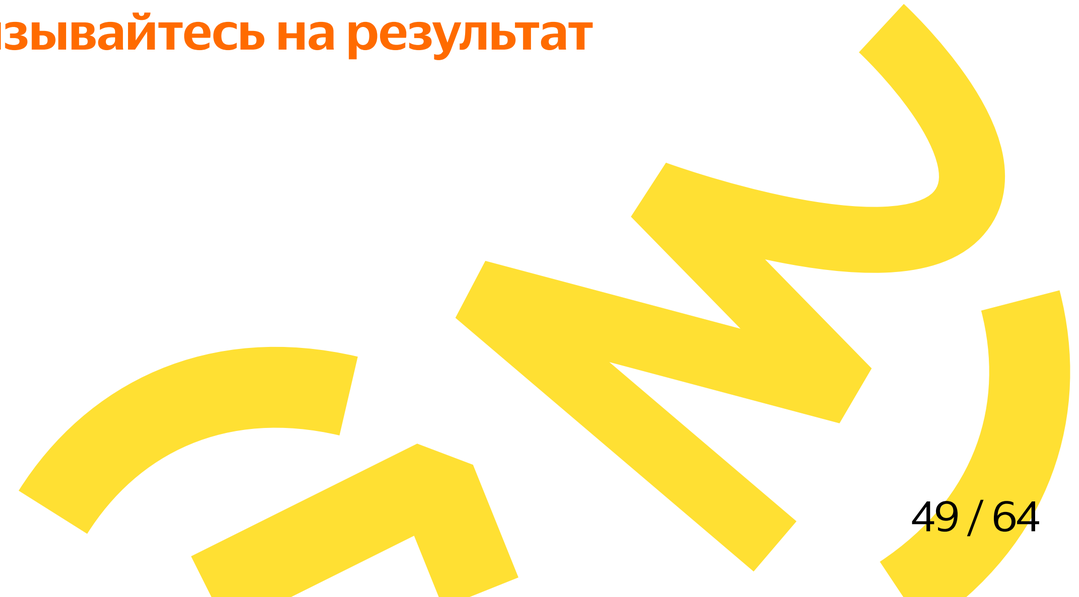


Rewrite batched inserts - нюансы

Драйвер группирует наборы параметров и не может угадать, какой конкретно набор параметров из каждой группы привел к вставке или изменению:

- `[-2, -2, 1]` - общий результат `Statement.SUCCESS_NO_INFO` для первой группы
- `[0, 0, 1]` - общий результат `0` для первой группы
- `[-2, -2, -2, -2]` - общий результат `Statement.SUCCESS_NO_INFO` для единственной группы

Если хотите `reWriteBatchedInserts=true`, то **не завязывайтесь на результат** `PreparedStatement.executeBatch`



Config?

application.properties:

```
spring.datasource.hikari.connection-timeout=33000
spring.datasource.hikari.data-source-properties.ApplicationName=appName
spring.datasource.hikari.data-source-properties.assumeMinServerVersion=10.0
spring.datasource.hikari.data-source-properties.connectTimeout=10
spring.datasource.hikari.data-source-properties.hostRecheckSeconds=11
spring.datasource.hikari.data-source-properties.loginTimeout=33
spring.datasource.hikari.data-source-properties.loadBalanceHosts=true
spring.datasource.hikari.data-source-properties.reWriteBatchedInserts=true # помним про executeBatch
spring.datasource.hikari.pool-name=poolName
spring.datasource.hikari.register-mbeans=true
```

sys props:

```
-Dhikaricp.jmx.register2.0=true
```

Parameter meta data?

Spring JDBC + `null` -параметры в запросах = `PreparedStatement.getParameterMetaData`

```
// create table t (id int primary key, v1 int, v2 int)
var url = "jdbc:postgresql://localhost:5432/postgres?user=postgres";
...
jdbcTemplate.batchUpdate(
    "insert into t(id,v1,v2) values (?,?,?)",
    List.of(
        new Object[]{1, null, null},
        new Object[]{2, null, null}
    ));
```

```
19:51:13.814 ... PreparedStatement.getParameterMetaData
19:51:13.815 ... PreparedStatement.getParameterMetaData
19:51:13.816 ... PreparedStatement.getParameterMetaData
19:51:13.816 ... PreparedStatement.getParameterMetaData
19:51:13.820 ... insert into t(id,v1,v2) values (?,?,?)
19:51:13.820 ... insert into t(id,v1,v2) values (?,?,?)
```

Parameter meta data?

- pgJDBC пока что не кэширует результат `PreparedStatement.getParameterMetaData` и каждый раз ходит в базу
- Spring JDBC вызывает `PreparedStatement.getParameterMetaData` для каждого null-параметра в каждом запросе, если явно не указывать тип параметра
- Логика живёт в `StatementCreatorUtils.setNull`, пытается дружить со всеми драйверами
- Началось со Spring 3.1.2 https://github.com/spring-projects/spring-framework/commit/79d9f7a5f731268a4a999ce45e0579f3c512a2f6

PS: примерная логика `StatementCreatorUtils.setNull` при отсутствии типа для `null` для разных драйверов:

```
if ("Informix..." || "Microsoft...SQL Server...") {
   ps.setObject(paramIndex, null);
} else if ("DB2..." || "jConnect..." || "SQLServer..." || "Apache Derby...") {
   ps.setNull(paramIndex, Types.VARCHAR);
}
```

No parameter meta data

`spring.jdbc.getParameterType.ignore`

- Ответ на баг 2014 года https://github.com/spring-projects/spring-framework/issues/16013, появилась в Spring 4.0.2
- Выключает вызов `PreparedStatement.getParameterMetadata` в `StatementCreatorUtils.setNull`, возвращает поведение Spring версии < 3.1.2
- Влияет на всё приложение целиком

Включить `-Dspring.jdbc.getParameterType.ignore=true`,
либо `src/main/resources/spring.properties`:

```
# https://docs.spring.io/spring-framework/reference/appendix.html#appendix-spring-properties
# https://docs.spring.io/spring-framework/reference/data-access/jdbc/advanced.html#jdbc-batch-list
# org.springframework.jdbc.core.StatementCreatorUtils#IGNORE_GETPARAMETERTYPE_PROPERTY_NAME
spring.jdbc.getParameterType.ignore=true
```

В application properties указывать бесполезно, см. `SpringProperties`

No parameter meta data — проверяем

Добавим к предыдущему примеру лишь `-Dspring.jdbc.getParameterType.ignore=true`

```
23:27:59.066 ... insert into t(id,v1,v2) values (?,?,?)
23:27:59.066 ... insert into t(id,v1,v2) values (?,?,?)
```

Лишние запросы ушли. А точно лишние?...

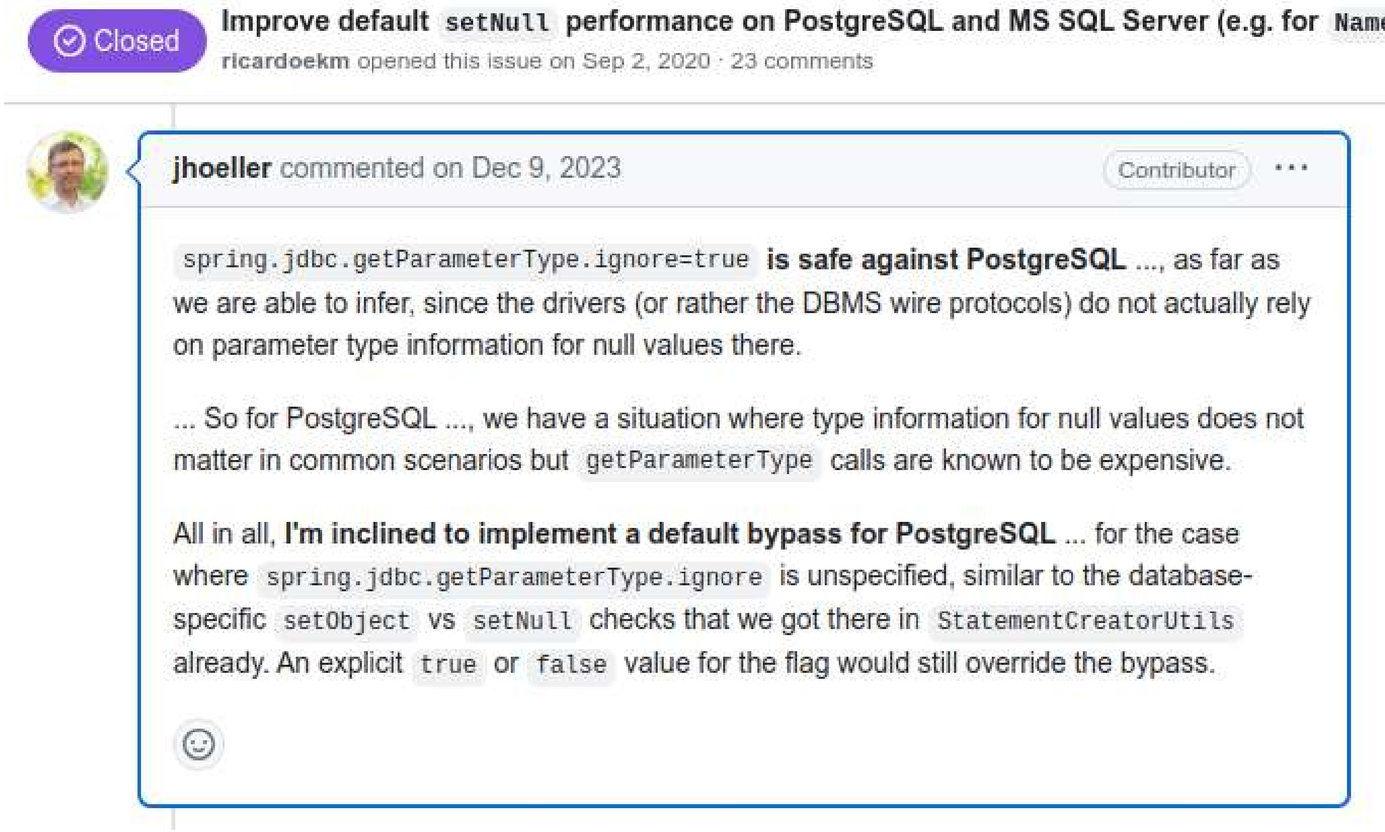
Почему не включить настройку по-умолчанию?...

Альтернатива — явно указывать типы:

```
jdbcTemplate.batchUpdate(
    "insert into t(id,v1,v2) values (?,?,?) on conflict (id) do nothing",
    List.of(
        new Object[]{1, null, null},
        new Object[]{2, null, null}
    ),
    new int[] {Types.INTEGER, Types.INTEGER, Types.INTEGER});
```

No parameter meta data — теперь из коробки

https://github.com/spring-projects/spring-framework/issues/25679#issuecomment-1848397915



Closed

Improve default `setNull` performance on PostgreSQL and MS SQL Server (e.g. for `NamedParam`

ricardoekm opened this issue on Sep 2, 2020 · 23 comments

jhoeller commented on Dec 9, 2023 — Contributor

`spring.jdbc.getParameterType.ignore=true` **is safe against PostgreSQL** ..., as far as we are able to infer, since the drivers (or rather the DBMS wire protocols) do not actually rely on parameter type information for null values there.

... So for PostgreSQL ..., we have a situation where type information for null values does not matter in common scenarios but `getParameterType` calls are known to be expensive.

All in all, **I'm inclined to implement a default bypass for PostgreSQL** ... for the case where `spring.jdbc.getParameterType.ignore` is unspecified, similar to the database-specific `setObject` vs `setNull` checks that we got there in `StatementCreatorUtils` already. An explicit `true` or `false` value for the flag would still override the bypass.

Флаг больше не требуется, начиная со Spring 6.1.2+ (Spring Boot 3.2.1+)

PS: https://github.com/pgjdbc/pgjdbc/issues/621 предварительно добавили в milestone 42.8.0

No parameter meta data — зачем типы для null?

```
// -Dspring.jdbc.getParameterType.ignore тут не имеет значения
var url = "jdbc:postgresql://localhost:5432/postgres
    ?user=postgres&prepareThreshold=1";
...
tx.executeWithoutResult(status -> {
    System.out.println(jdbcTemplate.queryForObject("select pg_typeof(abs(?))",
        new Object[]{null}, new int[]{Types.INTEGER}, String.class));
    // позже будет еще запрос
});
```

```
integer
```

Ожидаемо, пока всё ок.

PS: см. https://jdbc.postgresql.org/documentation/server-prepare/#server-prepared-statements

PPS: транзакция просто чтобы все запросы были в рамках одного соединения. Пул даст тот же эффект

PPPS: также включим логи для `org.postgresql.core.v3.SimpleQuery`, в pgJDBC используется java.util.logging

No parameter meta data — зачем типы для null?

```
...
tx.executeWithoutResult(status -> {
    ... "select pg_typeof(abs(?))", new Object[]{null}, new int[] {Types.INTEGER}, ...
    ... "select pg_typeof(abs(?))", new Object[]{null}, /* без типов, */ ...
});
```

Что вернет второй запрос?

No parameter meta data — зачем типы для null?

```
...
tx.executeWithoutResult(status -> {
    ... "select pg_typeof(abs(?))", new Object[]{null}, new int[] {Types.INTEGER}, ...
    ... "select pg_typeof(abs(?))", new Object[]{null}, /* без типов, */ ...
});
```

Неожиданный тип и повторый prepare запроса:

```
... org.postgresql.core.v3.SimpleQuery isPreparedFor
FINER: Statement S_2 does not match new parameter types.
Will have to un-prepare it and parse once again...
preparedType was INT4 (after describe INT4), current bind type is UNSPECIFIED

double precision
```

- не указали тип параметра в неоднозначном месте = неожиданный результат, может страдать performance
- сменили (даже неявно) тип параметра в том же запросе = повторный prepare, может страдать performance

No parameter meta data — зачем типы для null?

Если тип указывать неудобно, можно использовать и cast:

```
... "select pg_typeof(abs(?::integer))", new Object[]{null}, /* без типов, */ ...
```

```
integer
```

PS: помните про повторый prepare, cast в запросе от этого не спасает

No parameter meta data — резюме

Spring JDBC ≥ 3.1.2 (и < 6.1.2 для pgJDBC) на каждый `null` параметр в запросе без указанного типа вызывает `PreparedStatement.getParameterMetaData()` + `setNull(..., typeFromMetaData)`, а pgJDBC результат `getParameterMetaData()` пока что не кэширует

- Если у вас Spring < 3.1.2, то можно ничего не делать, но у вас на дворе ≤ 2012 год
- Если у вас Spring ≥ 4.0.2 и < 6.1.2 (Spring Boot < 3.2.1), то либо смело включайте `-Dspring.jdbc.getParameterType.ignore=true`, либо обновляйте Spring (Boot)
- Если у вас Spring ≥ 6.1.2 (Spring Boot ≥ 3.2.1), то ничего делать не надо, см. https://github.com/spring-projects/spring-framework/issues/25679

Когда типы `null`-параметров нужны и важны:

- Помните про перегрузку функций в PostgreSQL - лучше передавайте тип параметра (но можно и `cast` в запросе)
- Server-side prepare + выполнение запроса с разными типами, включая `Types.NULL` = повторный prepare

04 Итоговый конфиг

Config

application.properties:

```
# HikariCP любой, явно ставим значение ...data-source-properties.loginTimeout * 1000
spring.datasource.hikari.connection-timeout=33000

# pgJDBC 9.1.901+
spring.datasource.hikari.data-source-properties.ApplicationName=appName

# pgJDBC 9.4.1200+, имеет смысл указывать, начиная с версии сервера 9.0+
spring.datasource.hikari.data-source-properties.assumeMinServerVersion=10.0

# pgJDBC 9.3.1102+, таймаут подбирайте по вашему окружению
spring.datasource.hikari.data-source-properties.connectTimeout=10

# pgJDBC 9.4.1200+, чуть больше, чем connectTimeout
spring.datasource.hikari.data-source-properties.hostRecheckSeconds=11

# pgJDBC любой, пример (3 хоста) * (connectTimeout + 1s запас), или 0 для ожидания до упора
spring.datasource.hikari.data-source-properties.loginTimeout=33

# pgJDBC 9.4.1200+, если все хосты равнозначны, и порядок подключения не важен
spring.datasource.hikari.data-source-properties.loadBalanceHosts=true

# pgJDBC 9.4.1209+, "on conflict" исправлен в pgJDBC 42.2.2+,
# если не важен возвращаемый результат PreparedStatement.executeBatch
spring.datasource.hikari.data-source-properties.reWriteBatchedInserts=true

# HikariCP любой, можно также указать через spring.datasource.name=poolName
spring.datasource.hikari.pool-name=poolName

# HikariCP 1.3.0+, если нужны JMX метрики
spring.datasource.hikari.register-mbeans=true
```

sys props:

```
# HikariCP 4.0.0+, если нужны JMX метрики со стандарными ObjectName
-Dhikaricp.jmx.register2.0=true

# Spring [4.0.2, 6.1.1], backported to [3.2.8, 3.2.18], не требуется для Spring 6.1.2+
# и Spring Boot 3.2.1+, можно указать через spring.properties
-Dspring.jdbc.getParameterType.ignore=true
```

GitHub с конфигами

Config попроще, для самых маленьких

Если у вас маленький сервис, маленькая команда и всего один хост PostgreSQL, то многие настройки можно вполне оставить по-умолчанию.

application.properties:

```
spring.datasource.hikari.data-source-properties.ApplicationName=appName
spring.datasource.hikari.data-source-properties.assumeMinServerVersion=10.0
spring.datasource.hikari.data-source-properties.reWriteBatchedInserts=true
spring.datasource.hikari.pool-name=poolName
spring.datasource.hikari.register-mbeans=true
```

sys props:

```
-Dhikaricp.jmx.register2.0=true
-Dspring.jdbc.getParameterType.ignore=true
```

That's all, folks!

1. Пробегитесь по вашим настройкам.
   Надеюсь, найдёте, чем дополнить.
2. Регулярно обновляйте библиотеки.
   Минимум усилий могут дать много пользы.
3. Читайте changelog-и.
   Многие новые классные штуки не включены
   по-умолчанию из-за обратной совместимости.

Спасибо за внимание!

GitHub с конфигами