Reactive Relational Database Connectivity

Mark Paluch • Spring Data Engineer
@mp911de

© Copyright 2019 Pivotal Software, Inc. All rights Reserved.

Reactive Programming

- High-efficiency applications
- Fundamentally non-blocking
  - No opinion on async
- Key differentiators: (pull-push) back pressure, flow control


Subscribe
Publisher
Demand
Subscriber
Data


Subscribe
Publisher
request(n)
Subscriber
Data

Backpressure!

WebFlux and WebClient

```
@GetMapping("/health")
Mono<Health> compositeHealth() {
  return Mono.zip(
    webClient.get().uri("https://alpha-service/health")
      .retrieve().bodyToMono(Health.class),
    webClient.get().uri("https://bravo-service/health")
      .retrieve().bodyToMono(Health.class))
    .map(t -> composite(t.getT1(), t.getT2()));
}
```

Roadblocks

- Barriers to using Reactive everywhere
- Cross-process back pressure
  - RSocket
- Data Access
  - MongoDB, Apache Cassandra, Redis
  - No Relational Database Access

https://r2dbc.io

A specification designed from the ground up for reactive programming

Dependencies

- Reactive Streams
- Java 8

Design Principles

- Embrace Reactive Types and Patterns
- Non-blocking, all the way to the database
- Documented specification
- Shrink the driver SPI
- Enable multiple "humane" APIs

Driver SPI

- JDBC: same API for humane API and inhumane SPI for alternative clients like JPA, jOOQ, Jdbi, etc.
- API that users didn't like using and driver authors didn't like implementing
- Duplicating effort implementing the same "humane" affordances like **?** binding

Driver SPI - `ConnectionFactory`

```
    Publisher<Connection> create()

ConnectionFactoryMetadata getMetadata()
```

Driver SPI - `Connection`

```
Publisher<Void> beginTransaction()
Publisher<Void> close()
Publisher<Void> commitTransaction()
          Batch createBatch()
Publisher<Void> createSavepoint(String name)
      Statement createStatement(String sql)
Publisher<Void> releaseSavepoint(String name)
Publisher<Void> rollbackTransaction()
Publisher<Void> rollbackTransactionToSavepoint(String name)
Publisher<Void> setTransactionIsolationLevel(IsolationLevel isolationLevel)
```

Driver SPI - `Statement`

```
        Statement add()
        Statement bind(Object identifier, Object value)
        Statement bind(int index, Object value)
        Statement bind(int index, <primitive types> value)
        Statement bindNull(Object identifier, Class<?> type)
        Statement bindNull(int index, Class<?> type)
        Statement returnGeneratedValues(String… columnNames)
Publisher<Result> execute()
```

Driver SPI - `Result` and `Row`

```
Publisher<Integer> getRowsUpdated()

      Publisher<T> map(BiFunction<Row, RowMetadata, ? extends T> f)



     T get(Object identifier, Class<T> type);

Object get(Object identifier);
```

Simple Select

```
Publisher<Object> values = connectionFactory.create()
  .flatMapMany(conn ->
    conn.createStatement("SELECT value FROM test")
        .execute()
        .flatMap(result ->
              result.map((row, metadata) -> row.get("value"))))
```

Simple Select

```
Publisher<String> values = connectionFactory.create()
  .flatMapMany(conn ->
    conn.createStatement("SELECT value FROM test")
        .execute()
        .flatMap(result ->
              result.map((row, metadata) -> row.get("value",
                                                    String.class))))
```

Simple Prepared Insert

```
Publisher<Result> results = connectionFactory.create()
  .flatMapMany(conn ->
    conn.createStatement("INSERT INTO test VALUES($1, $2)")
      .bind("$1", 100).bind("$2", 200).add()
      .bind("$1", 300).bind("$2", 400).execute())
```

Transactional Prepared Insert

```
Publisher<Result> results = connectionFactory.create()
  .flatMapMany(conn ->
    conn.beginTransaction()
      .thenMany(conn.createStatement("INSERT INTO test VALUES($1)")
        .bind("$1", 100).add()
        .bind("$1", 200).execute())
      .delayUntil(p -> conn.commitTransaction())
      .onErrorResume(t ->
        conn.rollbackTransaction().then(Mono.error(t))))
```

Great! But a Bit Verbose

- Minimal set of implementation specific operations
- Definitely usable, but very verbose and prone to errors
  - Explicit transaction management is analogous to `try-catch-finally-try-catch` in JDBC
- We need a "humane" client API. In fact we need ***many*** humane client APIs!

Simple Select

```
Flux<String> values = r2dbc.withHandle(handle ->
  handle.select("SELECT value FROM test")
    .mapRow(row -> row.get("value", String.class)))
```

Simple Prepared Insert

```
Flux<Integer> updatedRows = r2dbc.withHandle(handle ->
  handle.createUpdate("INSERT INTO test VALUES($1, $2)")
    .bind("$1", 100).bind("$2", 200).add()
    .bind("$1", 300).bind("$2", 400).execute())
```

Transactional Prepared Insert

```
Flux<Integer> updatedRows = r2dbc.inTransaction(handle ->
  handle.createUpdate("INSERT INTO test VALUES($1, $2)")
    .bind("$1", 100).bind("$2", 200).add()
    .bind("$1", 300).bind(„$2", 400).execute())
```

Spring Data DatabaseClient

```
DatabaseClient client = DatabaseClient.create(connectionFactory);

Flux<Person> rows = client.execute()
          .sql("SELECT * FROM person WHERE name = :name")
          .bind("John Doe")
          .as(Person.class)
          .fetch()
          .all();
```

Spring Data Repository

```
interface CustomerRepository extends
    ReactiveCrudRepository<Customer, Long> {

  @Query("SELECT * FROM … WHERE lastname = :lastname")
  Flux<Customer> findByLastname(String lastname);
}


repository.findByLastname("Matthews")
  .doOnEach(c -> System.out.println(c.firstname))
```

R2DBC Connection URL

```
r2dbc:pool:postgresql://localhost:5432/database?key=value


ConnectionFactory connectionFactory =
        ConnectionFactories.get("r2dbc:postgresql://myhost/database?
        driver=foo");
```

What Can You Do Today?

- Alpha-quality and not used in production
- Driver implementations for H2, Microsoft SQL Server, PostgreSQL, `r2dbc-over-adba`
  - Batching
  - Extensive Type Conversion
  - Savepoints
  - Transactions
  - Transaction Isolation
  - Leveraging Database-specific features

R2DBC Eco-System

- Specification documentation
- Driver implementations

- R2DBC SPI
- R2DBC Proxy
- Connection Pooling

- Community
  - MySQL Driver

- Client Implementations
  - Spring Data R2DBC
  - `r2dbc-client`

R2DBC Proxy

- Interception Proxy
- Community Contribution
- Top-Level R2DBC Project
- Observability
  - Metrics
  - Tracing
  - APM

What R2DBC gives you

- Move Thread congestion out of JVM
- Achieve more with less Threads

- Doesn’t change law of physics

- Database laws still apply
  - Obey wire protocol rules
  - ACID rules

What About the Alternatives?

- Wrap JDBC in a thread pool
  - Unbounded queue leads to resource exhaustion
  - Bounded queue leads to blocking
- ADBA - the  in the room
  - Should use Java 9's `Flow` to get proper Reactive Streams back pressure, but currently implemented with `CompletableFuture` which does not
  - No implementations JARs available

Safe Harbor Statement

The following is intended to outline the general direction of Pivotal's offerings. It is intended for information purposes only and may not be incorporated into any contract. Any information regarding pre-release of Pivotal offerings, future updates or other planned modifications is subject to ongoing evaluation by Pivotal and is subject to change. This information is provided without warranty or any kind, express or implied, and is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions regarding Pivotal's offerings. These purchasing decisions should only be based on features currently available. The development, release, and timing of any features or functionality described for Pivotal's offerings in this presentation remain at the sole discretion of Pivotal. Pivotal has no obligation to update forward looking information in this presentation.

What Does the Future Hold?

- Continuing additions/improvements to SPI
  - BLOB/CLOB
  - Stored Procedures
- Additional drivers
  - DB2
  - Prefer database vendors to own drivers long-term
- Need at least one additional client
  - MicroProfile, MyBatis, JDBI, jOOQ

- Ideally, R2DBC influences ADBA (or successor)
  - Spring doesn't generally create specs, but we feel strongly enough to carry this forward

Resources

Get Engaged!

- Website
  https://r2dbc.io
- Twitter
  @r2dbc
- GitHub
  https://github.com/r2dbc
- Mailing List
  https://groups.google.com/forum/#!forum/r2dbc
- Weekly Call
  Fridays 0630 PT/0930 ET/1530 CET

Transforming How The World Builds Software

© Copyright 2019 Pivotal Software, Inc. All rights Reserved.