Using multiple datasources with Spring Boot and Spring Data 💈⇄🌱⇄💈

Jannik Hell · Follow
3 min read · May 4, 2017

2.3K 48

Spring Boot with **Spring Data** makes it easy to access a database through so called Repositories. But what if you want to access **multiple databases** maybe even with different Database Management Systems?

Luckily Spring provides a way of doing this.

I provided an example project with two PostgreSQL datasources on GitHub: https://github.com/jahe/spring-boot-multiple-datasources

1. Add an additional datasource configuration to your application.properties

```
# Oracle DB - "foo"
spring.datasource.url=jdbc:oracle:thin:@//db-server-foo:1521/FOO
spring.datasource.username=fooadmin
spring.datasource.password=foo123
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver

# PostgreSQL DB - "bar"
bar.datasource.url=jdbc:postgresql://db-server-bar:5432/bar
bar.datasource.username=baradmin
bar.datasource.password=bar123
bar.datasource.driver-class-name=org.postgresql.Driver
```

2. Set the SQL Dialect to “default” in your application.properties to let Spring autodetect the different SQL Dialects of each datasource

```
spring.jpa.database=default
```

3. Create a Java Package for each datasource with two nested Packages “domain” and “repo”

```
src/main/java
- com.foobar
  - foo
    - domain
    - repo
  - bar
    - domain
    - repo
```

4. Create a Configuration Class for the Oracle database “foo” named “FooDbConfig.java”

```
package com.foobar;

@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(
  entityManagerFactoryRef = "entityManagerFactory",
  basePackages = { "com.foobar.foo.repo" }
)
public class FooDbConfig {

  @Primary
  @Bean(name = "dataSource")
  @ConfigurationProperties(prefix = "spring.datasource")
  public DataSource dataSource() {
    return DataSourceBuilder.create().build();
  }

  @Primary
  @Bean(name = "entityManagerFactory")
  public LocalContainerEntityManagerFactoryBean
  entityManagerFactory(
    EntityManagerFactoryBuilder builder,
    @Qualifier("dataSource") DataSource dataSource
  ) {
    return builder
      .dataSource(dataSource)
      .packages("com.foobar.foo.domain")
      .persistenceUnit("foo")
      .build();
  }

  @Primary
  @Bean(name = "transactionManager")
  public PlatformTransactionManager transactionManager(
    @Qualifier("entityManagerFactory") EntityManagerFactory
    entityManagerFactory
  ) {
    return new JpaTransactionManager(entityManagerFactory);
  }
}
```

5. Create a Configuration Class for the PostgreSQL database "bar" named "BarDbConfig.java"

```
package com.foobar;

@Configuration
@EnableTransactionManagement
@EnableJpaRepositories(
  entityManagerFactoryRef = "barEntityManagerFactory",
  transactionManagerRef = "barTransactionManager",
  basePackages = { "com.foobar.bar.repo" }
)
public class BarDbConfig {

  @Bean(name = "barDataSource")
  @ConfigurationProperties(prefix = "bar.datasource")
  public DataSource dataSource() {
    return DataSourceBuilder.create().build();
  }

  @Bean(name = "barEntityManagerFactory")
  public LocalContainerEntityManagerFactoryBean
  barEntityManagerFactory(
    EntityManagerFactoryBuilder builder,
    @Qualifier("barDataSource") DataSource dataSource
  ) {
    return
      builder
        .dataSource(dataSource)
        .packages("com.foobar.bar.domain")
        .persistenceUnit("bar")
        .build();
  }

  @Bean(name = "barTransactionManager")
  public PlatformTransactionManager barTransactionManager(
    @Qualifier("barEntityManagerFactory") EntityManagerFactory
    barEntityManagerFactory
  ) {
    return new JpaTransactionManager(barEntityManagerFactory);
  }
}
```

6. Create an Entity “Foo.java” for the Oracle database “foo”

```
package com.foobar.foo.domain;

@Entity
@Table(name = "FOO")
public class Foo {

  @Id
  @GeneratedValue
  @Column(name = "ID")
  private Long id;

  @Column(name = "FOO")
  private String foo;

  Foo(String foo) {
    this.foo = foo;
  }

  Foo() {
    // Default constructor needed by JPA
  }
}
```

7. Create a Repository “FooRepository.java” for the Oracle database “foo”

```
package com.foobar.foo.repo;

@Repository
public interface FooRepository extends JpaRepository<Foo, Long> {

  Foo findById(Long id);

}
```

8. Create an Entity “Bar.java” for the PostgreSQL database “bar”

```
package com.foobar.bar.domain;

@Entity
@Table(name = "BAR")
public class Bar {

  @Id
  @GeneratedValue
  @Column(name = "ID")
  private Long id;

  @Column(name = "BAR")
  private String bar;

  Bar(String bar) {
    this.bar = bar;
  }

  Bar() {
    // Default constructor needed by JPA
  }
}
```

9. Create a Repository “BarRepository.java” for the PostgreSQL database “bar”

```
package com.foobar.bar.repo;

@Repository
public interface BarRepository extends JpaRepository<Bar, Long> {

  Bar findById(Long id);

}
```

10. Create the Spring Boot Main Class “Application.java”

```
package com.foobar;

@SpringBootApplication
public class Application {
```

11. Use the Repositories in a REST Controller (or somewhere else)

```
package com.foobar;

@RestController
public class FooBarController {

  private final FooRepository fooRepo;
  private final BarRepository barRepo;

  @Autowired
  FooBarController(FooRepository fooRepo, BarRepository barRepo) {
    this.fooRepo = fooRepo;
    this.barRepo = barRepo;
  }

  @RequestMapping("/foobar/{id}")
  public String fooBar(@PathVariable("id") Long id) {
    Foo foo = fooRepo.findById(id);
    Bar bar = barRepo.findById(id);

    return foo.getFoo() + " " + bar.getBar();
  }
}
```

Done. 👍

Thanks for reading and feel free to comment. 💛

Follow me on Twitter if you like: [@joeclever]

Example project with two PostgreSQL datasources: https://github.com/jahe/spring-boot-multiple-datasources

Java Spring Boot Spring Spring Data Multiple Datasources

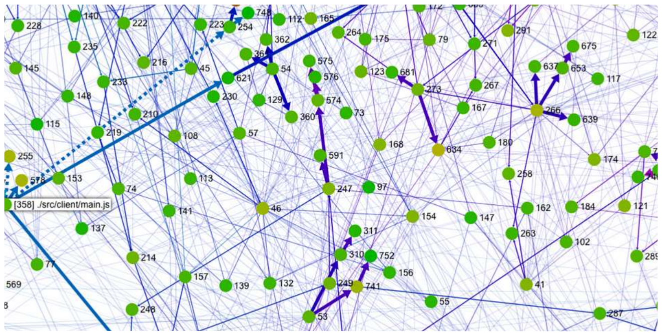
Written by Jannik Hell

Follow

314 Followers

Developer, @joeclever

More from Jannik Hell



Jannik Hell

Jannik Hell