Spring Boot configure and use two data sources

Asked 9 years, 1 month ago Modified 1 year, 6 months ago Viewed 568k times

341

How can I configure and use two data sources?

For example, here is what I have for the first data source:

application.properties

```
#first db
spring.datasource.url = [url]
spring.datasource.username = [username]
spring.datasource.password = [password]
spring.datasource.driverClassName = oracle.jdbc.OracleDriver

#second db ...
```

Application class

```
@SpringBootApplication
public class SampleApplication
{
    public static void main(String[] args) {
        SpringApplication.run(SampleApplication.class, args);
    }
}
```

How do I modify `application.properties` to add another data source? How do I autowire it to be used by a different repository?

java spring spring-boot spring-mvc datasource

Share Edit Follow

edited Jun 15, 2022 at 7:33

Mark Rotteveel
107k • 215 • 149 • 205

asked May 19, 2015 at 23:03

juventus
3,422 • 3 • 12 • 7

1 The 2nd answer most voted below (from @Surasin) used to be a good one if you don't need distributed transactions across both dbs. But **ChainedTransactionManager** has been deprecated. Anyone landing here in 2023 looking for a robust solution for distributed transactions must read this answer stackoverflow.com/a/71392118/7066647 – dbaltor Jan 20, 2023 at 12:53

10 Answers

Sorted by: Highest score (default)

Here you go.

391

Add in your application.properties file:

```
#first db
spring.datasource.url = [url]
spring.datasource.username = [username]
spring.datasource.password = [password]
spring.datasource.driverClassName = oracle.jdbc.OracleDriver

#second db ...
spring.secondDatasource.url = [url]
spring.secondDatasource.username = [username]
spring.secondDatasource.password = [password]
spring.secondDatasource.driverClassName = oracle.jdbc.OracleDriver
```

Add in any class annotated with @Configuration the following methods:

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

@Bean
@ConfigurationProperties(prefix="spring.secondDatasource")
public DataSource secondaryDataSource() {
    return DataSourceBuilder.create().build();
}
```

Share Edit Follow

edited Sep 20, 2020 at 12:02
Marco Altieri
3,806 ● 2 ● 34 ● 49

answered May 20, 2015 at 8:46
K. Siva Prasad Reddy
12.2k ● 14 ● 72 ● 101

35 Take a look at baeldung.com/spring-data-jpa-multiple-databases which describes the same what you are looking for. – K. Siva Prasad Reddy May 21, 2015 at 2:34

Sometimes you may need to assign datasource, transactionManager, and SqlSessionFactory as primary all. – Dai Kaixian Dec 15, 2016 at 10:37

8 @K. Siva Prasad Reddy OK but I have 2 different JPARepositories - how does Spring Boot know which DataSource to use? Every JPARepository shoudl use different database – Matley Mar 18, 2019 at 14:50

4 @Matley This blog post javadevjournal.com/spring-boot/... might be what you are looking for. – K. Siva Prasad Reddy Mar 19, 2019 at 9:52

1 will this work if one is oracle db and another is hsql. its not working for me – harsha kumar Reddy Aug 29, 2022 at 7:24

173

Update 2022-05-29 with Spring Boot 1.5.8.RELEASE which should work with Spring Boot 2.x

Most answers do not provide how to use them (as datasource itself and as transaction), only how to config them.

Moreover you should know how to commit/rollback transactions of both datasources at the same time.

You can see the runnable example and some explanation in https://github.com/surasint/surasint-examples/tree/master/spring-boot-jdbi/10_spring-boot-two-databases (see what you can try in README.txt)

I copied some code here.

First you have to set application.properties like this

```
#Database
database1.datasource.url=jdbc:mysql://localhost/testdb
database1.datasource.username=root
database1.datasource.password=root
database1.datasource.driver-class-name=com.mysql.jdbc.Driver

database2.datasource.url=jdbc:mysql://localhost/testdb2
database2.datasource.username=root
database2.datasource.password=root
database2.datasource.driver-class-name=com.mysql.jdbc.Driver
```

Then define them as providers (@Bean) like this:

```
@Bean(name = "datasource1")
@ConfigurationProperties("database1.datasource")
@Primary
public DataSource dataSource(){
    return DataSourceBuilder.create().build();
}

@Bean(name = "datasource2")
@ConfigurationProperties("database2.datasource")
public DataSource dataSource2(){
    return DataSourceBuilder.create().build();
}
```

Note that I have `@Bean(name="datasource1")` and `@Bean(name="datasource2")`, then you can use it when we need datasource as `@Qualifier("datasource1")` and `@Qualifier("datasource2")` , for example

```
@Qualifier("datasource1")
@Autowired
private DataSource dataSource;
```

If you do care about transaction, you have to define DataSourceTransactionManager for both of them, like this:

```
@Bean(name="tm1")
@Autowired
@Primary
DataSourceTransactionManager tm1(@Qualifier ("datasource1") DataSource
datasource) {
    DataSourceTransactionManager txm  = new
DataSourceTransactionManager(datasource);
```

```
    return txm;
}

@Bean(name="tm2")
@Autowired
DataSourceTransactionManager tm2(@Qualifier ("datasource2") DataSource
datasource) {
    DataSourceTransactionManager txm  = new
DataSourceTransactionManager(datasource);
    return txm;
}
```

Then you can use it like

```
@Transactional //this will use the first datasource because it is @primary
```

or

```
@Transactional("tm2")
```

The most important part, which you will hardly find an example in anywhere: if you want a method to commit/rollback transactions of both databases, you need ChainedTransactionManager for tm1 and tm2 , like this:

```
@Bean(name = "chainedTransactionManager")
public ChainedTransactionManager getChainedTransactionManager(@Qualifier ("tm1")
DataSourceTransactionManager tm1, @Qualifier ("tm2") DataSourceTransactionManager
tm2){
    return new ChainedTransactionManager(tm1, tm2);
}
```

To use it, add this annotation in a method @Transactional(value="chainedTransactionManager") for example

```
@Transactional(value="chainedTransactionManager")
public void insertAll() {
    UserBean test = new UserBean();
    test.setUsername("username" + new Date().getTime());
    userDao.insert(test);

    userDao2.insert(test);
}
```

This should be enough. See example and detail in the link above.

Share Edit Follow

edited May 29, 2022 at 13:36

answered Jan 7, 2018 at 19:45

Surasin Tancharoen

5,730 ● 4 ● 33 ● 40

1 Hi @Surasin Tancharoen, we are trying to keep two data sources with same data so that if one fails, the application will run on the other data source. Will be above approach be fine? – Arun Sudhakaran May 13, 2020 at 5:02

1 @ArunSudhakaran No. This solution will not work as a backup. If you are looking for high availability, then most databases already have configurations to run multiple databases with a single virtual IP. try that instead. – Raja Anbazhagan Jul 8, 2021 at 15:15

4 Link surasint.com/spring-boot-with-multiple-databases-example does not work anymore. – Avec Sep 27, 2021 at 11:49

36

Refer the official documentation

Creating more than one data source works same as creating the first one. You might want to mark one of them as @Primary if you are using the default auto-configuration for JDBC or JPA (then that one will be picked up by any @Autowired injections).

```
@Bean
@Primary
@ConfigurationProperties(prefix="datasource.primary")
public DataSource primaryDataSource() {
    return DataSourceBuilder.create().build();
}

@Bean
@ConfigurationProperties(prefix="datasource.secondary")
public DataSource secondaryDataSource() {
    return DataSourceBuilder.create().build();
}
```

Share Edit Follow

edited Jan 21, 2019 at 3:36

answered May 20, 2015 at 9:45

Faraj Farook
14.7k ● 17 ● 73 ● 97

Thank you for the link to the official documentation on this. – Manoj Shrestha Sep 10, 2021 at 19:29

Hi, what about if I want to use not the primary database, how should I tell JPA which one to pick? Because I try to use this approach but I have an error for no such table in primary database (the table is not in the primary, but in the secondary). – Даяна Димитрова Jul 12, 2022 at 6:41

24

I also had to setup connection to 2 datasources from Spring Boot application, and it was not easy - the solution mentioned in the Spring Boot documentation didn't work. After a long digging through the internet I made it work and the main idea was taken from this article and bunch of other places.

The following solution is written in **Kotlin** and works with **Spring Boot 2.1.3** and **Hibernate Core 5.3.7**. Main issue was that it was not enough just to setup different **DataSource** configs, but it was also necessary to configure **EntityManagerFactory** and **TransactionManager** for both databases.

Here is config for the first (Primary) database:

```
@Configuration
@EnableJpaRepositories(
    entityManagerFactoryRef = "firstDbEntityManagerFactory",
    transactionManagerRef = "firstDbTransactionManager",
    basePackages = ["org.path.to.firstDb.domain"]
)
@EnableTransactionManagement
class FirstDbConfig {

    @Bean
    @Primary
    @ConfigurationProperties(prefix = "spring.datasource.firstDb")
    fun firstDbDataSource(): DataSource {
        return DataSourceBuilder.create().build()
    }

    @Primary
    @Bean(name = ["firstDbEntityManagerFactory"])
    fun firstDbEntityManagerFactory(
        builder: EntityManagerFactoryBuilder,
        @Qualifier("firstDbDataSource") dataSource: DataSource
    ): LocalContainerEntityManagerFactoryBean {
        return builder
            .dataSource(dataSource)
            .packages(SomeEntity::class.java)
            .persistenceUnit("firstDb")
            // Following is the optional configuration for naming strategy
            .properties(
                singletonMap(
                    "hibernate.naming.physical-strategy",

"org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl"
                )
            )
            .build()
    }

    @Primary
    @Bean(name = ["firstDbTransactionManager"])
    fun firstDbTransactionManager(
        @Qualifier("firstDbEntityManagerFactory") firstDbEntityManagerFactory:
EntityManagerFactory
    ): PlatformTransactionManager {
        return JpaTransactionManager(firstDbEntityManagerFactory)
    }
}
```

And this is config for second database:

```
@Configuration
@EnableJpaRepositories(
    entityManagerFactoryRef = "secondDbEntityManagerFactory",
    transactionManagerRef = "secondDbTransactionManager",
    basePackages = ["org.path.to.secondDb.domain"]
)
@EnableTransactionManagement
class SecondDbConfig {

    @Bean
    @ConfigurationProperties("spring.datasource.secondDb")
    fun secondDbDataSource(): DataSource {
        return DataSourceBuilder.create().build()
    }
```

```
    @Bean(name = ["secondDbEntityManagerFactory"])
    fun secondDbEntityManagerFactory(
        builder: EntityManagerFactoryBuilder,
        @Qualifier("secondDbDataSource") dataSource: DataSource
    ): LocalContainerEntityManagerFactoryBean {
        return builder
            .dataSource(dataSource)
            .packages(EntityFromSecondDb::class.java)
            .persistenceUnit("secondDb")
            .build()
    }

    @Bean(name = ["secondDbTransactionManager"])
    fun secondDbTransactionManager(
        @Qualifier("secondDbEntityManagerFactory") secondDbEntityManagerFactory:
EntityManagerFactory
    ): PlatformTransactionManager {
        return JpaTransactionManager(secondDbEntityManagerFactory)
    }
}
```

The properties for datasources are like this:

```
spring.datasource.firstDb.jdbc-url=
spring.datasource.firstDb.username=
spring.datasource.firstDb.password=

spring.datasource.secondDb.jdbc-url=
spring.datasource.secondDb.username=
spring.datasource.secondDb.password=
```

Issue with properties was that I had to define **jdbc-url** instead of **url** because otherwise I had an exception.

p.s. Also you might have different naming schemes in your databases, which was the case for me. Since Hibernate 5 does not support all previous naming schemes, I had to use solution from this answer - maybe it will also help someone as well.

Share Edit Follow

answered Mar 12, 2019 at 19:18

AbstractVoid
3,879 ● 3 ● 41 ● 40

2 I had problems with entity and table naming. And it's helped me along with your answer: `mapOf("hibernate.physical_naming_strategy" to "org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy", "hibernate.implicit_naming_strategy" to "org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy" )` – rvit34 Dec 17, 2019 at 15:24

1 I replace `url` with `jdbcUrl`, it works, thank you – Galley Jan 6, 2022 at 9:34

```
# Here '1stDB' is the database name
spring.datasource.url=jdbc:mysql://localhost/A
spring.datasource.username=root
```

6

```
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.jdbc.Driver


# Here '2ndDB' is the database name
spring.second-datasourcee.url=jdbc:mysql://localhost/B
spring.second-datasource.username=root
spring.second-datasource.password=root
spring.second-datasource.driver-class-name=com.mysql.jdbc.Driver
```

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

@Bean
@ConfigurationProperties(prefix = "spring.second-datasource")
public DataSource secondDataSource() {
   return DataSourceBuilder.create().build();
}
```

Share Edit Follow

edited Nov 30, 2022 at 9:55

Toni
4,757 ● 3 ● 15 ● 47

answered May 29, 2018 at 5:57

Raju Ranjan
71 ● 1 ● 2

Not working for me. And saying this: Error creating bean with name 'dataSourceScriptDatabaseInitializer' defined in class path resource [org/springframework/boot/autoconfigure/sql/init/DataSourceInitializationConfiguration.class]: Invocation of init method failed; nested exception is java.lang.IllegalArgumentException: jdbcUrl is required with driverClassName. – Amir Keshavarz Jun 27, 2021 at 4:10

3 @AmirKeshavarz my answer might be a little ... to late, but you have to modify the datasource.url to datasource.jdbc-url and it will work – user9802118 Sep 30, 2021 at 21:32

5

Here is the Complete solution

```
#First Datasource (DB1)
db1.datasource.url: url
db1.datasource.username:user
db1.datasource.password:password

#Second Datasource (DB2)
db2.datasource.url:url
db2.datasource.username:user
db2.datasource.password:password
```

Since we are going to get access two different databases (db1, db2), we need to configure each data source configuration separately like:

```
public class DB1_DataSource {
@Autowired
private Environment env;
@Bean
```

```
@Primary
public LocalContainerEntityManagerFactoryBean db1EntityManager() {
    LocalContainerEntityManagerFactoryBean em = new
LocalContainerEntityManagerFactoryBean();
    em.setDataSource(db1Datasource());
    em.setPersistenceUnitName("db1EntityManager");
    HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    em.setJpaVendorAdapter(vendorAdapter);
    HashMap<string, object=""> properties = new HashMap<>();
    properties.put("hibernate.dialect",
            env.getProperty("hibernate.dialect"));
    properties.put("hibernate.show-sql",
            env.getProperty("jdbc.show-sql"));
    em.setJpaPropertyMap(properties);
    return em;
}

@Primary
@Bean
public DataSource db1Datasource() {

    DriverManagerDataSource dataSource
            = new DriverManagerDataSource();
    dataSource.setDriverClassName(
            env.getProperty("jdbc.driver-class-name"));
    dataSource.setUrl(env.getProperty("db1.datasource.url"));
    dataSource.setUsername(env.getProperty("db1.datasource.username"));
    dataSource.setPassword(env.getProperty("db1.datasource.password"));

    return dataSource;
}

@Primary
@Bean
public PlatformTransactionManager db1TransactionManager() {

    JpaTransactionManager transactionManager
            = new JpaTransactionManager();
    transactionManager.setEntityManagerFactory(
            db1EntityManager().getObject());
    return transactionManager;
}
}
```

Second Datasource :

```
public class DB2_DataSource {

@Autowired
private Environment env;

@Bean
public LocalContainerEntityManagerFactoryBean db2EntityManager() {
    LocalContainerEntityManagerFactoryBean em
            = new LocalContainerEntityManagerFactoryBean();
    em.setDataSource(db2Datasource());
    em.setPersistenceUnitName("db2EntityManager");
    HibernateJpaVendorAdapter vendorAdapter
            = new HibernateJpaVendorAdapter();
    em.setJpaVendorAdapter(vendorAdapter);
    HashMap<string, object=""> properties = new HashMap<>();
    properties.put("hibernate.dialect",
            env.getProperty("hibernate.dialect"));
    properties.put("hibernate.show-sql",
```

```
                env.getProperty("jdbc.show-sql"));
        em.setJpaPropertyMap(properties);
        return em;
    }

    @Bean
    public DataSource db2Datasource() {
        DriverManagerDataSource dataSource
                = new DriverManagerDataSource();
        dataSource.setDriverClassName(
                env.getProperty("jdbc.driver-class-name"));
        dataSource.setUrl(env.getProperty("db2.datasource.url"));
        dataSource.setUsername(env.getProperty("db2.datasource.username"));
        dataSource.setPassword(env.getProperty("db2.datasource.password"));

        return dataSource;
    }

    @Bean
    public PlatformTransactionManager db2TransactionManager() {
        JpaTransactionManager transactionManager
                = new JpaTransactionManager();
        transactionManager.setEntityManagerFactory(
                db2EntityManager().getObject());
        return transactionManager;
    }
    }
```

Here you can find the complete Example on my blog : Spring Boot with Multiple DataSource Configuration

Share Edit Follow

edited Apr 20, 2018 at 11:27

answered Nov 30, 2017 at 22:28

Chandra Shekhar Goka

874 ● 12 ● 19

this code will not run how 2 @primary annotation work together in same db. – Rohit Chaurasiya Jul 24, 2020 at 2:51

indeed, but this code mentions important part with manual properties setting instead of plain and simple `return DataSourceBuilder.create().build();` which is apparently not working, hence my upvote goes here – im_infamous Oct 20, 2020 at 15:38

what if your 2 databases are from different vendor? How do you set hibernate dialect in that case? – pixel Jul 11, 2022 at 17:03

3

My requirement was slightly different but used two data sources.

I have used two data sources for same JPA entities from same package. One for executing DDL at the server startup to create/update tables and another one is for DML at runtime.

The DDL connection should be closed after DDL statements are executed, to prevent further usage of super user previlleges anywhere in the code.

Properties

```
spring.datasource.url=jdbc:postgresql://Host:port
ddl.user=ddluser
ddl.password=ddlpassword
dml.user=dmluser
dml.password=dmlpassword
spring.datasource.driver-class-name=org.postgresql.Driver
```

Data source config classes

//1st Config class for DDL Data source

```
  public class DatabaseDDLConfig {
        @Bean
        public LocalContainerEntityManagerFactoryBean
ddlEntityManagerFactoryBean() {
            LocalContainerEntityManagerFactoryBean entityManagerFactoryBean = new
LocalContainerEntityManagerFactoryBean();
            PersistenceProvider persistenceProvider = new
            org.hibernate.jpa.HibernatePersistenceProvider();
            entityManagerFactoryBean.setDataSource(ddlDataSource());
            entityManagerFactoryBean.setPackagesToScan(new String[] {
            "com.test.two.data.sources"});
            HibernateJpaVendorAdapter vendorAdapter = new
            HibernateJpaVendorAdapter();
            entityManagerFactoryBean.setJpaVendorAdapter(vendorAdapter);
            HashMap<String, Object> properties = new HashMap<>();
            properties.put("hibernate.dialect",
            "org.hibernate.dialect.PostgreSQLDialect");
            properties.put("hibernate.physical_naming_strategy",
            "org.springframework.boot.orm.jpa.hibernate.
            SpringPhysicalNamingStrategy");
            properties.put("hibernate.implicit_naming_strategy",
            "org.springframework.boot.orm.jpa.hibernate.
            SpringImplicitNamingStrategy");
            properties.put("hibernate.hbm2ddl.auto", "update");
            entityManagerFactoryBean.setJpaPropertyMap(properties);
            entityManagerFactoryBean.setPersistenceUnitName("ddl.config");
            entityManagerFactoryBean.setPersistenceProvider(persistenceProvider);
            return entityManagerFactoryBean;
        }


    @Bean
    public DataSource ddlDataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName(env.getProperty("spring.datasource.driver-
class-name"));
        dataSource.setUrl(env.getProperty("spring.datasource.url"));
        dataSource.setUsername(env.getProperty("ddl.user");
        dataSource.setPassword(env.getProperty("ddl.password"));
        return dataSource;
    }

    @Bean
    public PlatformTransactionManager ddlTransactionManager() {
        JpaTransactionManager transactionManager = new JpaTransactionManager();

transactionManager.setEntityManagerFactory(ddlEntityManagerFactoryBean().getObject()
        return transactionManager;
    }
}
```

//2nd Config class for DML Data source

```
public class DatabaseDMLConfig {

    @Bean
    @Primary
    public LocalContainerEntityManagerFactoryBean dmlEntityManagerFactoryBean() {
        LocalContainerEntityManagerFactoryBean entityManagerFactoryBean = new
LocalContainerEntityManagerFactoryBean();
        PersistenceProvider persistenceProvider = new
org.hibernate.jpa.HibernatePersistenceProvider();
        entityManagerFactoryBean.setDataSource(dmlDataSource());
        entityManagerFactoryBean.setPackagesToScan(new String[] {
"com.test.two.data.sources" });
        JpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
        entityManagerFactoryBean.setJpaVendorAdapter(vendorAdapter);
        entityManagerFactoryBean.setJpaProperties(defineJpaProperties());
        entityManagerFactoryBean.setPersistenceUnitName("dml.config");
        entityManagerFactoryBean.setPersistenceProvider(persistenceProvider);
        return entityManagerFactoryBean;
    }

    @Bean
    @Primary
    public DataSource dmlDataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName(env.getProperty("spring.datasource.driver-
class-name"));
        dataSource.setUrl(envt.getProperty("spring.datasource.url"));
        dataSource.setUsername("dml.user");
        dataSource.setPassword("dml.password");
        return dataSource;
    }

    @Bean
    @Primary
    public PlatformTransactionManager dmlTransactionManager() {
        JpaTransactionManager transactionManager = new JpaTransactionManager();

transactionManager.setEntityManagerFactory(dmlEntityManagerFactoryBean().getObject()
        return transactionManager;
    }


  }
```

//Usage of DDL data sources in code.

```
public class DDLServiceAtStartup {

//Import persistence unit ddl.config for ddl purpose.

@PersistenceUnit(unitName = "ddl.config")
private EntityManagerFactory entityManagerFactory;

public void executeDDLQueries() throws ContentServiceSystemError {
    try {
        EntityManager entityManager = entityManagerFactory.createEntityManager();
        entityManager.getTransaction().begin();
        entityManager.createNativeQuery("query to create/update
table").executeUpdate();
        entityManager.flush();
```

```
        entityManager.getTransaction().commit();
        entityManager.close();

        //Close the ddl data source to avoid from further use in code.
        entityManagerFactory.close();
    } catch(Exception ex) {}
}
```

//Usage of DML data source in code.

```
public class DDLServiceAtStartup {
  @PersistenceUnit(unitName = "dml.config")
  private EntityManagerFactory entityManagerFactory;

  public void createRecord(User user) {
     userDao.save(user);
  }
}
```

Share Edit Follow

answered Nov 29, 2018 at 20:45

Anil Konduru
978 ● 11 ● 20

2

@Primary annotation when used against a method like below works good if the two data sources are on the same db location/server.

```
@Bean(name = "datasource1")
@ConfigurationProperties("database1.datasource")
@Primary
public DataSource dataSource(){
  return DataSourceBuilder.create().build();
}

@Bean(name = "datasource2")
@ConfigurationProperties("database2.datasource")
public DataSource dataSource2(){
  return DataSourceBuilder.create().build();
}
```

If the data sources are on different servers its better to use @Component along with @Primary annotation. The following code snippet works well on two different data sources at different locations

```
database1.datasource.url = jdbc:mysql://127.0.0.1:3306/db1
database1.datasource.username = root
database1.datasource.password = mysql
database1.datasource.driver-class-name=com.mysql.jdbc.Driver

database2.datasource1.url = jdbc:mysql://192.168.113.51:3306/db2
database2.datasource1.username = root
database2.datasource1.password = mysql
database2.datasource1.driver-class-name=com.mysql.jdbc.Driver

@Configuration
@Primary
@Component
```

```
@ComponentScan("com.db1.bean")
class DBConfiguration1{
    @Bean("db1Ds")
    @ConfigurationProperties(prefix="database1.datasource")
    public DataSource primaryDataSource() {
        return DataSourceBuilder.create().build();
    }

}

@Configuration
@Component
@ComponentScan("com.db2.bean")
class DBConfiguration2{
    @Bean("db2Ds")
    @ConfigurationProperties(prefix="database2.datasource1")
    public DataSource primaryDataSource() {
        return DataSourceBuilder.create().build();
    }

}
```

Share Edit Follow

answered Sep 19, 2018 at 8:03

Sangameshrh
21 • 1

1

I used mybatis - springboot 2.0 tech stack, solution:

```
//application.properties - start
    sp.ds1.jdbc-url=jdbc:mysql://localhost:3306/mydb?useSSL=false
    sp.ds1.username=user
    sp.ds1.password=pwd
    sp.ds1.testWhileIdle=true
    sp.ds1.validationQuery=SELECT 1
    sp.ds1.driverClassName=com.mysql.jdbc.Driver


    sp.ds2.jdbc-url=jdbc:mysql://localhost:4586/mydb?useSSL=false
    sp.ds2.username=user
    sp.ds2.password=pwd
    sp.ds2.testWhileIdle=true
    sp.ds2.validationQuery=SELECT 1
    sp.ds2.driverClassName=com.mysql.jdbc.Driver

//application.properties - end

//configuration class

    @Configuration
    @ComponentScan(basePackages = "com.mypkg")
    public class MultipleDBConfig {


        public static final String SQL_SESSION_FACTORY_NAME_1 =
"sqlSessionFactory1";
        public static final String SQL_SESSION_FACTORY_NAME_2 =
"sqlSessionFactory2";

        public static final String MAPPERS_PACKAGE_NAME_1 = "com.mypg.mymapper1";
        public static final String MAPPERS_PACKAGE_NAME_2 = "com.mypg.mymapper2";
```

```
        @Bean(name = "mysqlDb1")
        @Primary
        @ConfigurationProperties(prefix = "sp.ds1")
        public DataSource dataSource1() {
            System.out.println("db1 datasource");
            return DataSourceBuilder.create().build();
        }

        @Bean(name = "mysqlDb2")
        @ConfigurationProperties(prefix = "sp.ds2")
        public DataSource dataSource2() {
            System.out.println("db2 datasource");
            return  DataSourceBuilder.create().build();
        }

        @Bean(name = SQL_SESSION_FACTORY_NAME_1)
        @Primary
        public SqlSessionFactory sqlSessionFactory1(@Qualifier("mysqlDb1")
DataSource dataSource1) throws Exception {
            System.out.println("sqlSessionFactory1");
            SqlSessionFactoryBean sqlSessionFactoryBean = new
SqlSessionFactoryBean();
            sqlSessionFactoryBean.setTypeHandlersPackage(MAPPERS_PACKAGE_NAME_1);
            sqlSessionFactoryBean.setDataSource(dataSource1);
            SqlSessionFactory sqlSessionFactory =
sqlSessionFactoryBean.getObject();

sqlSessionFactory.getConfiguration().setMapUnderscoreToCamelCase(true);

sqlSessionFactory.getConfiguration().setJdbcTypeForNull(JdbcType.NULL);
            return sqlSessionFactory;
        }

        @Bean(name = SQL_SESSION_FACTORY_NAME_2)
        public SqlSessionFactory sqlSessionFactory2(@Qualifier("mysqlDb2")
DataSource dataSource2) throws Exception {
            System.out.println("sqlSessionFactory2");
            SqlSessionFactoryBean diSqlSessionFactoryBean = new
SqlSessionFactoryBean();

diSqlSessionFactoryBean.setTypeHandlersPackage(MAPPERS_PACKAGE_NAME_2);
            diSqlSessionFactoryBean.setDataSource(dataSource2);
            SqlSessionFactory sqlSessionFactory =
diSqlSessionFactoryBean.getObject();

sqlSessionFactory.getConfiguration().setMapUnderscoreToCamelCase(true);

sqlSessionFactory.getConfiguration().setJdbcTypeForNull(JdbcType.NULL);
            return sqlSessionFactory;
        }

        @Bean
        @Primary
        public MapperScannerConfigurer mapperScannerConfigurer1() {
            System.out.println("mapperScannerConfigurer1");
            MapperScannerConfigurer configurer = new MapperScannerConfigurer();
            configurer.setBasePackage(MAPPERS_PACKAGE_NAME_1);
            configurer.setSqlSessionFactoryBeanName(SQL_SESSION_FACTORY_NAME_1);
            return configurer;
        }

        @Bean
        public MapperScannerConfigurer mapperScannerConfigurer2() {
            System.out.println("mapperScannerConfigurer2");
            MapperScannerConfigurer configurer = new MapperScannerConfigurer();
```

```
            configurer.setBasePackage(MAPPERS_PACKAGE_NAME_2);
            configurer.setSqlSessionFactoryBeanName(SQL_SESSION_FACTORY_NAME_2);
            return configurer;
        }



    }
```

Note : 1)@Primary -> @primary

2)---."jdbc-url" in properties -> After Spring Boot 2.0 migration: jdbcUrl is required with driverClassName

Share Edit Follow

edited Aug 29, 2018 at 7:11

answered Aug 14, 2018 at 14:11

Akhil S Kamath

1,042 ● 13 ● 24

1

> ***declaring a data source in Spring Boot application.properties***

```
spring.datasource.company.url=jdbc:mysql://localhost/company_db?
createDatabaseIfNotExist=true&autoReconnect=true&useSSL=false&allowPublicKeyRetrieva
spring.datasource.company.username=root
spring.datasource.company.password=root
spring.datasource.company.platform=mysql


spring.datasource.employee.url=jdbc:mysql://localhost/employee_db?
createDatabaseIfNotExist=true&autoReconnect=true&useSSL=false&allowPublicKeyRetrieva
spring.datasource.employee.username=root
spring.datasource.employee.password=root
spring.datasource.employee.platform=mysql
```

> use multiple data sources, we need to declare multiple beans with different mappings within Spring's application context. using a configuration class

```
@Configuration
@EnableJpaRepositories(basePackages =
"com.example.multiple.datasources.entity.company",
        entityManagerFactoryRef = "companyEntityManagerFactory",
        transactionManagerRef = "companyTransactionManager")
public class CompanyDataSourceConfiguration {

    @Bean
    @ConfigurationProperties("spring.datasource.company")
    public DataSourceProperties companyDataSourceProperties() {
        return new DataSourceProperties();
    }

    @Bean
    @ConfigurationProperties("spring.datasource.company.configuration")
```

```
    public DataSource companyDataSource() {
        return companyDataSourceProperties().initializeDataSourceBuilder()
                .type(HikariDataSource.class).build();
    }

    @Bean(name = "companyEntityManagerFactory")
    public LocalContainerEntityManagerFactoryBean
companyEntityManagerFactory(EntityManagerFactoryBuilder builder) {
        return
builder.dataSource(companyDataSource()).packages(Company.class).build();
    }

    @Bean
    public PlatformTransactionManager companyTransactionManager(
            final @Qualifier("companyEntityManagerFactory")
LocalContainerEntityManagerFactoryBean companyEntityManagerFactory
    ) {
        return new
JpaTransactionManager(companyEntityManagerFactory.getObject());

    }


}
```

> we need to declare one of the datasources as @Primary. This is because EntityManagerFactoryBuilder is declared in JpaBaseConfiguration and this class need a single data source injected.

```
@Configuration
@EnableJpaRepositories(basePackages =
"com.example.multiple.datasources.entity.employee",
        entityManagerFactoryRef = "employeeEntityManagerFactory",
        transactionManagerRef = "employeeTransactionManager")
public class EmployeeDatasourceConfiguration {

    @Bean
    @Primary
    @ConfigurationProperties("spring.datasource.employee")
    public DataSourceProperties employeeDataSourceProperties() {
        return new DataSourceProperties();

    }

    @Bean
    @Primary
    @ConfigurationProperties("spring.datasource.employee.configuration")
    public DataSource employeeDataSource() {
        return
employeeDataSourceProperties().initializeDataSourceBuilder().type(HikariDataSource.c
    }

    @Primary
    @Bean("employeeEntityManagerFactory")
    public LocalContainerEntityManagerFactoryBean
employeeEntityManagerFactory(EntityManagerFactoryBuilder builder) {
        return
builder.dataSource(employeeDataSource()).packages(Employee.class).build();
    }
```

```
    @Primary
    @Bean
    public PlatformTransactionManager employeeTransactionManager(
            final @Qualifier("employeeEntityManagerFactory")
LocalContainerEntityManagerFactoryBean employeeEntityManagerFactory
    ) {
        return new
JpaTransactionManager(employeeEntityManagerFactory.getObject());

    }

}
```

Share Edit Follow

answered May 24, 2022 at 11:31

ngg
1,651 ● 20 ● 17

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.