# SpringBoot---JPA配置多个数据库连接

在工作中可能涉及到连接多个数据库进行数据之间的验证，在某些场景下我们可能需要在一个项目中配置多个数据，本篇文章简要介绍在Spring Boot中配置JPA的多数据库连接，以配置两个数据库为例。

本例中使用gradle进行版本管理。

1. 添加数据库配置

**manager**:  
 **spring**:  
 **datasource**:  
 **driver-class-name**: org.mariadb.jdbc.Driver  
 **jdbcUrl**: jdbc:mysql://${**MANAGER\_DB\_HOSTNAME:localhost**}:3306/${**MANAGER\_DB\_NAME:manager**}?useSSL=false  
 **username**: ${**MANAGER\_DB\_USER\_NAME:root**}  
 **password**: ${**MANAGER\_DB\_PASSWORD:admin**}  
**slave**:  
 **spring**:  
 **datasource**:  
 **driver-class-name**: org.mariadb.jdbc.Driver  
 **jdbcUrl**: jdbc:mysql://${**SLAVE\_DB\_HOSTNAME:localhost**}:3306/${**SLAVE\_DB\_NAME:slave**}?useSSL=false  
 **username**: ${**SLAVE\_DB\_USER\_NAME:root**}  
 **password**: ${**SLAVE\_DB\_PASSWORD:admin**}

2.创建数据源配置类

@Configuration  
public class DataSourceConfig {  
 */\*\*  
 \* create DataSource instance of manager  
 \*  
 \** ***@return*** *\*/* @Primary  
 @Bean(name = "managerDataSource")  
 @ConfigurationProperties(prefix = "manager.spring.datasource")  
 public DataSource v1DataSource() {  
 return DataSourceBuilder.*create*().build();  
 }  
  
 */\*\*  
 \* create DataSource instance of slave  
 \*  
 \** ***@return*** *\*/* @Bean(name = "slaveDataSource")  
 @ConfigurationProperties(prefix = "slave.spring.datasource")  
 public DataSource v2DataSource() {  
 return DataSourceBuilder.*create*().build();  
 }  
}

3,manager数据库配置

@Configuration  
@EnableTransactionManagement  
@EnableJpaRepositories(  
 entityManagerFactoryRef = "managerEntityManagerFactory",  
 transactionManagerRef = "managerTransactionManager",  
 basePackageClasses = {ManagerRepository.class}  
)  
public class ManagerConfig {  
 @Autowired  
 @Qualifier(value = "managerDataSource")  
 private DataSource managerDataSource;  
  
 @Primary  
 @Bean(name = "managerEntityManager")  
 public EntityManager entityManager(EntityManagerFactoryBuilder builder) {  
 return managerEntityManagerFactory(builder).getObject().createEntityManager();  
 }  
  
 @Primary  
 @Bean(name = "managerEntityManagerFactory")  
 public LocalContainerEntityManagerFactoryBean managerEntityManagerFactory(  
 EntityManagerFactoryBuilder builder) {  
 return builder  
 .dataSource(managerDataSource)  
 .packages(ManagerEntity.class)  
 .persistenceUnit("managerPersistenceUnit")  
 .build();  
 }  
  
 @Primary  
 @Bean(name = "managerTransactionManager")  
 PlatformTransactionManager managerTransactionManager(EntityManagerFactoryBuilder builder) {  
 return new JpaTransactionManager(managerEntityManagerFactory(builder).getObject());  
 }  
}

public interface ManagerEntity {  
}

4.slave配置

@Configuration  
@EnableTransactionManagement  
@EnableJpaRepositories(  
 entityManagerFactoryRef = "slaveEntityManagerFactory",  
 transactionManagerRef = "slaveTransactionManager",  
 basePackageClasses = {SlaveRepository.class}  
)  
public class SlaveConfig {  
 @Autowired  
 @Qualifier(value = "slaveDataSource")  
 private DataSource slaveDataSource;  
  
 @Bean(name = "slaveEntityManager")  
 public EntityManager entityManager(EntityManagerFactoryBuilder builder) {  
 return slaveEntityManagerFactory(builder).getObject().createEntityManager();  
 }  
  
 @Bean(name = "slaveEntityManagerFactory")  
 public LocalContainerEntityManagerFactoryBean slaveEntityManagerFactory(  
 EntityManagerFactoryBuilder builder) {  
 return builder  
 .dataSource(slaveDataSource)  
 .packages(ManagerEntity.class)  
 .persistenceUnit("managerPersistenceUnit")  
 .build();  
 }  
  
 @Bean(name = "slaveTransactionManager")  
 PlatformTransactionManager slaveTransactionManager(EntityManagerFactoryBuilder builder) {  
 return new JpaTransactionManager(slaveEntityManagerFactory(builder).getObject());  
 }  
}

public interface SlaveEntity {  
}