1 问题起源

- 为了方便他人操作自己的项目，在项目中配置好需要用到的数据库表和相关数据
- 只要在资源路径中添加 `schema.sql` 和 `data.sql` 文件，`springboot` 在运行的时候就会自动关联数据库创建相应的表和数据

2 源码跟踪

2.1 核心类：DataSourceAutoConfiguration

```
@Configuration
@ConditionalOnClass({DataSource.class, EmbeddedDatabaseType.class})
@EnableConfigurationProperties({DataSourceProperties.class})
//DataSourceAutoConfiguration 这个类导入了另一个类：DataSourceInitializationConfiguration
@Import({DataSourcePoolMetadataProvidersConfiguration.class,
DataSourceInitializationConfiguration.class})
public class DataSourceAutoConfiguration {...}

@Configuration
//这个类导入了DataSourceInitializationConfiguration.Registrar：完成了
DataSourceInitializerPostProcessor后置处理器的注册
//这个类导入了DataSourceInitializerInvoker：这个类完成了sql文件自动执行
@Import({DataSourceInitializerInvoker.class,
DataSourceInitializationConfiguration.Registrar.class})
class DataSourceInitializationConfiguration {
    DataSourceInitializationConfiguration() {
    }
    static class Registrar implements ImportBeanDefinitionRegistrar {
        private static final String BEAN_NAME = "dataSourceInitializerPostProcessor";
        Registrar() {
        }
        public void registerBeanDefinitions(AnnotationMetadata importingClassMetadata,
BeanDefinitionRegistry registry) {
```

```
            if (!registry.containsBeanDefinition("dataSourceInitializerPostProcessor"))
{
                GenericBeanDefinition beanDefinition = new GenericBeanDefinition();
                beanDefinition.setBeanClass(DataSourceInitializerPostProcessor.class);
                beanDefinition.setRole(2);
                beanDefinition.setSynthetic(true);
                registry.registerBeanDefinition("dataSourceInitializerPostProcessor",
beanDefinition);
            }

        }
    }
}
```

2.2 关键类：DataSourceInitializerInvoker

- 这个类中初始化了一个 `DataSourceInitializer`：其中封装了主数据源；`DataSourceProperties`：取自于 `spring.datasource` 配置属性
- 调用 `dataSourceInitializer.createSchema()` 方法运行 `sql文件`

```
//这个类中有3个关键方法
class DataSourceInitializerInvoker implements InitializingBean {
//当初始化完后，就会调用这个方法，来运行sql文件
    public void afterPropertiesSet() {
        DataSourceInitializer initializer = this.getDataSourceInitializer();
        if (initializer != null) {
            //这里运行schema.sql文件
            boolean schemaCreated = this.dataSourceInitializer.createSchema();
            if (schemaCreated) {
                //这里同理运行data.sql文件
                this.initialize(initializer);
            }
        }
    }
}
   private DataSourceInitializer getDataSourceInitializer() {
       if (this.dataSourceInitializer == null) {
           DataSource ds = (DataSource)this.dataSource.getIfUnique();
           if (ds != null) {
               this.dataSourceInitializer = new DataSourceInitializer(ds,
this.properties, this.applicationContext);
           }
       }

       return this.dataSourceInitializer;
   }
```

2.3 核心方法：dataSourceInitializer.createSchema()

- 首先查找到Scripts资源
- 在运行Scripts资源

```
public boolean createSchema() {
    //关键：是从这里定位到Scripts资源的！！
    List<Resource> scripts = this.getScripts("spring.datasource.schema",
this.properties.getSchema(), "schema");
    //从properties获取SchemaUsername，chemaPassword
        String username = this.properties.getSchemaUsername();
        String password = this.properties.getSchemaPassword();
        //这里运行scripts资源
        this.runScripts(scripts, username, password);
    }
    return !scripts.isEmpty();
}
```

2.3.1 Scripts 资源定位：getScripts()

```
//如何获取Scripts，默认获取地址是哪个
private List<Resource> getScripts(String propertyName, List<String> resources, String
fallback) {
    //如果配置文件中定义了schema位置，则直接用这个位置定位资源
    if (resources != null) {
        return this.getResources(propertyName, resources, true);
    } else {
        //默认获取地址：classpath*:schema-all.sql,classpath*:schema.sql
        String platform = this.properties.getPlatform();
        List<String> fallbackResources = new ArrayList();
        fallbackResources.add("classpath*:" + fallback + "-" + platform + ".sql");
        fallbackResources.add("classpath*:" + fallback + ".sql");
        return this.getResources(propertyName, fallbackResources, false);
    }
}
```

2.3.2 运行 Scripts

```
private void runScripts(List<Resource> resources, String username, String password) {
    if (!resources.isEmpty()) {
        Iterator var5 = resources.iterator();
        while(var5.hasNext()) {
            Resource resource = (Resource)var5.next();
            populator.addScript(resource);
        }
        //这里获取的还是@primary指定的数据源
        DataSource dataSource = this.dataSource;
        //关键是这一步，如果username这两个多有的话，数据源就变成spring.datasource.url了，哈哈，这
是干啥，直接用@primary数据源不就行了吗，但是也好，初始化schema的数据源，独立于其他的数据源
        //如果要用@primary的数据源，不指定data-username，schema-username即可即可
        if (StringUtils.hasText(username) && StringUtils.hasText(password)) {
            dataSource =
DataSourceBuilder.create(this.properties.getClassLoader()).driverClassName(this.propert
ies.determineDriverClassName()).url(this.properties.determineUrl()).username(username).
password(password).build();
        }
```

```
            DatabasePopulatorUtils.execute(populator, dataSource);
        }
    }
```

3 总结

- 在初始化第一个数据源时，会调用 `afterPropertiesSet` 方法，进而运行 `sql文件`
- 默认使用主数据源运行 `classpath*:schema-all.sql,classpath*:schema.sql` 和 `classpath*:data-all.sql,classpath*:data.sql`
- 只能通过 `spring.datasource-` 来指定 `sql文件` 的路径
- 如果不想使用主数据源运行 `sql文件` ，可以通过 `spring.datasource-` 来指定数据源，但必须指定对应的用户名密码
- 完全可以把核心代码抽取出来，自定义运行

参考

源码地址