第 3-7 课：Spring Boot 集成 Druid 监控数据源

Druid 介绍

Druid 是阿里巴巴开源平台上的一个项目，整个项目由数据库连接池、插件框架和 SQL 解析器组成，该项目主要是为了扩展 JDBC 的一些限制，可以让程序员实现一些特殊的需求，比如向密钥服务请求凭证、统计 SQL 信息、SQL 性能收集、SQL 注入检查、SQL 翻译等，程序员可以通过定制来实现自己需要的功能。

Druid 首先是一个数据库连接池，但它不仅仅是一个数据库连接池，还包含了一个 ProxyDriver，一系列内置的 JDBC 组件库，一个 SQL Parser。在 Java 的世界中 Druid 是监控做的最好的数据库连接池，在功能、性能、扩展性方面，也有不错的表现。

Druid 可以做什么

- 替换其他 Java 连接池，Druid 提供了一个高效、功能强大、可扩展性好的数据库连接池。
- 可以监控数据库访问性能，Druid 内置提供了一个功能强大的 StatFilter 插件，能够详细统计 SQL 的执行性能，这对于线上分析数据库访问性能有很大帮助。
- 数据库密码加密。直接把数据库密码写在配置文件中，这是不好的行为，容易导致安全问题，DruidDruiver 和 DruidDataSource 都支持 PasswordCallback。
- SQL 执行日志，Druid 提供了不同的 LogFilter，能够支持 Common-Logging、Log4j 和 JdkLog，可以按需要选择相应的 LogFilter，监控应用的数据库访问情况。
- 扩展 JDBC，如果你要对 JDBC 层有编程的需求，可以通过 Druid 提供的 Filter 机制，很方便编写 JDBC 层的扩展插件。

Spring Boot 集成 Druid

非常令人高兴的是，阿里为 Druid 也提供了 Spring Boot Starter 的支持。官网这样解释：Druid Spring Boot Starter 用于帮助你在 Spring Boot 项目中轻松集成 Druid 数据库连接池和监控。

Druid Spring Boot Starter 主要做了哪些事情呢？其实这个组件包很简单，主要提供了很多自动化的配置，按照 Spring Boot 的理念对很多内容进行了预配置，让我们在使用的时候更加的简单和方便。

MyBatis 中使用 Druid 作为连接池

在前面课程中的 spring-boot-mybatis-annotation 上去集成。

引入依赖包

```
<dependency>
   <groupId>com.alibaba</groupId>
   <artifactId>druid-spring-boot-starter</artifactId>
   <version>1.1.10</version>
</dependency>
```

- druid-spring-boot-starter 的最新版本为 1.1.10，会自动依赖 Druid 相关包。

application 配置

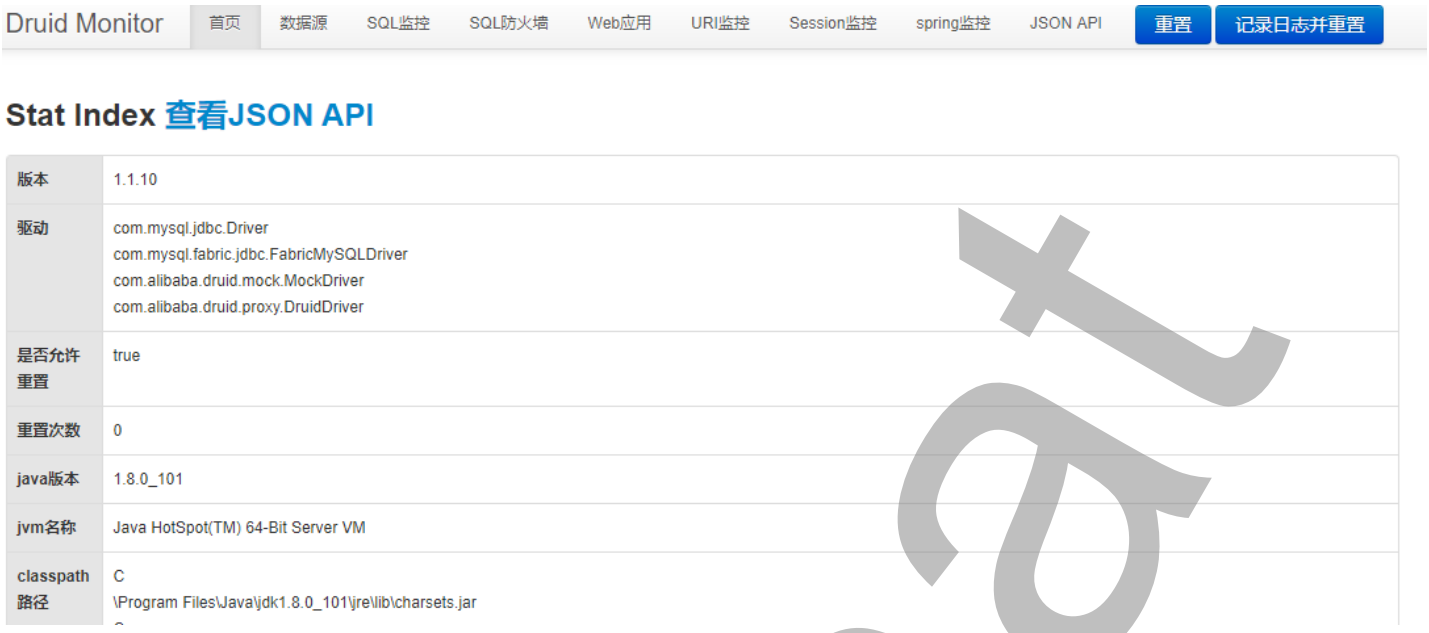
Druid Spring Boot Starter 配置属性的名称完全遵照 Druid，可以通过 Spring Boot 配置文件来配置 Druid 数据库连接池和监控，如果没有配置则使用默认值。

```
# 实体类包路径
mybatis.type-aliases-package=com.neo.model

spring.datasource.type: com.alibaba.druid.pool.DruidDataSource
spring.datasource.url=jdbc:mysql://localhost:3306/test?serverTimezone=UTC&useUnico
de=true&characterEncoding=utf-8&useSSL=true
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# 初始化大小、最小、最大连接数
spring.datasource.druid.initial-size=3
spring.datasource.druid.min-idle=3
spring.datasource.druid.max-active=10

# 配置获取连接等待超时的时间
spring.datasource.druid.max-wait=60000

# 监控后台账号和密码
spring.datasource.druid.stat-view-servlet.login-username=admin
spring.datasource.druid.stat-view-servlet.login-password=admin

# 配置 StatFilter
spring.datasource.druid.filter.stat.log-slow-sql=true
spring.datasource.druid.filter.stat.slow-sql-millis=2000
```

在以前项目的基础上，增加了对 Druid 连接池的配置，以及 SQL 监控的配置，druid-spring-boot-starter 默认情况下开启 StatFilter 的监控功能。Druid Spring Boot Starter 不限于对以上配置属性提供支持，DruidDataSource 内提供 setter 方法的可配置属性都将被支持。

更多配置内容请参考 druid-spring-boot-starter。

配置完成后，直接启动项目访问地址：http://localhost:8080/druid，就会出现 Druid 监控后台的登录页面，输入账户和密码后，就会进入首页。



首页会展示项目使用的 JDK 版本、数据库驱动、JVM 相关统计信息。根据上面的菜单可以看出 Druid 的功能非常强大，支持数据源、SQL 监控、SQL 防火墙、URI 监控等很多功能。

我们这里重点介绍一下 SQL 监控，具体的展示信息如下：

这里的 SQL 监控会将项目中具体执行的 SQL 打印出来，展示此 SQL 执行了多少次、每次返回多少数据、执行的时间分布是什么。这些功能非常的实用，方便我们在实际生产中查找出慢 SQL，最后对 SQL 进行调优。

从这个例子可发现，使用 Spring Boot 集成 Druid 非常的简单，只需要添加依赖，简单配置就可以。

MyBatis + Druid 多数据源

接下来为大家介绍 MyBatis 多数据源中是如何使用 Druid 数据库连接池的。

配置文件

首先我们需要配置两个不同的数据源：

```
spring.datasource.druid.one.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.druid.one.url = jdbc:mysql://localhost:3306/test1?serverTimezone
=UTC&useUnicode=true&characterEncoding=utf-8&useSSL=true
spring.datasource.druid.one.username = root
spring.datasource.druid.one.password = root

spring.datasource.druid.two.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.druid.two.url = jdbc:mysql://localhost:3306/test2?serverTimezone
=UTC&useUnicode=true&characterEncoding=utf-8&useSSL=true
spring.datasource.druid.two.username = root
spring.datasource.druid.two.password = root
```

第一个数据源以 spring.datasource.druid.one.* 为前缀连接数据库 test1，第二个数据源以 spring.datasource.druid.two.* 为前缀连接数据库 test2。

强烈注意：Spring Boot 2.X 版本不再支持配置继承，多数据源的话每个数据源的所有配置都需要单独配置，否则配置不会生效。

```
#  StatViewServlet 配置
spring.datasource.druid.stat-view-servlet.login-username=admin
spring.datasource.druid.stat-view-servlet.login-password=admin

# 配置 StatFilter
spring.datasource.druid.filter.stat.log-slow-sql=true
spring.datasource.druid.filter.stat.slow-sql-millis=2000

# Druid 数据源 1 配置
spring.datasource.druid.one.initial-size=3
spring.datasource.druid.one.min-idle=3
spring.datasource.druid.one.max-active=10
spring.datasource.druid.one.max-wait=60000

# Druid 数据源 2 配置
spring.datasource.druid.two.initial-size=6
spring.datasource.druid.two.min-idle=6
spring.datasource.druid.two.max-active=20
spring.datasource.druid.two.max-wait=120000
```

filter 和 stat 作为 Druid 的公共信息配置，其他数据源的配置需要各个数据源单独配置。

注入多数据源

首先为两个数据源创建不同的 Mapper 包路径，将以前的 UserMapper 复制到包 com.neo.mapper.one 和 com.neo.mapper.two 路径下，并且分别重命名为 UserOneMapper、UserTwoMapper。

定义一个 MultiDataSourceConfig 类，对两个不同的数据源进行加载：

```
@Configuration
public class MultiDataSourceConfig {
    @Primary
    @Bean(name = "oneDataSource")
    @ConfigurationProperties("spring.datasource.druid.one")
    public DataSource dataSourceOne(){
        return DruidDataSourceBuilder.create().build();
    }
    @Bean(name = "twoDataSource")
    @ConfigurationProperties("spring.datasource.druid.two")
    public DataSource dataSourceTwo(){
        return DruidDataSourceBuilder.create().build();
    }
}
```

必须指明一个为默认的主数据源，使用注解：@Primary。加载配置两个数据源的 DataSourceConfig 和前面课程中 MyBatis 多数据源使用的配置一致没有变化。

> 注意：在多数据源的情况下，我们不需要再启动类添加 @MapperScan("com.xxx.mapper") 的注解。

测试使用

以上所有的配置内容都完成后，启动项目访问这个地址：http://localhost:8080/druid，单击数据源查看数据库连接信息。

> 如果数据源没有信息，先访问地址：http://localhost:8080/getUsers，用来触发数据库连接。在没有 SQL 使用的情况下，页面监控不到数据源的配置信息，SQL 监控页面也监控不到 SQL 的执行。

显示效果如下：

Basic Info For DataSource-12342458 View JSON API

* 用户名	root	指定建立连接时使用的用户名
* 连接地址	jdbc:mysql://localhost:3306/test1?useUnicode=true&characterEncoding=utf-8&useSSL=true	jdbc连接字符串
* 数据库类型	mysql	数据库类型
* 驱动类名	com.mysql.jdbc.Driver	jdbc驱动的类名
* filter类名	com.alibaba.druid.filter.stat.StatFilter	filter的类名
* 获取连接时检测	false	是否在获得连接后检测其可用性
* 空闲时检测	true	是否在连接空闲一段时间后检测其可用性
* 连接放回连接池时检测	false	是否在连接放回连接池后检测其可用性
* 初始化连接大小	3	连接池建立时创建的初始化连接数
* 最小空闲连接数	3	连接池中最小的活跃连接数
* 最大连接数	10	连接池中最大的活跃连接数
* 查询超时时间	0	查询超时时间
* 事务查询超时时间	0	事务查询超时时间
* 登录超时时间	0	

摘录自数据源1的显示信息

Keyword	value	解释
连接地址	jdbc:mysql://localhost:3306/test1?useUnicode=true&characterEncoding=utf-8	JDBC 连接字符串
初始化连接大小	3	连接池建立时创建的初始化连接数
最小空闲连接数	3	连接池中最小的活跃连接数
最大连接数	10	连接池中最大的活跃连接数
MaxWait	10000	配置获取连接等待超时的时间

摘录自数据源2的显示信息

Keyword	value	解释
连接地址	jdbc:mysql://localhost:3306/test2?useUnicode=true&characterEncoding=utf-8	JDBC 连接字符串
初始化连接大小	6	连接池建立时创建的初始化连接数
最小空闲连接数	6	连接池中最小的活跃连接数
最大连接数	20	连接池中最大的活跃连接数
MaxWait	120000	配置获取连接等待超时的时间

通过这两个数据源的连接信息来看，两个数据源的配置信息已经生效。

Spring Data JPA 中使用 Druid 作为连接池

Spring Data JPA 集成 Druid 的方式和 MyBatis 大体相同。

引入相关依赖包：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
   <groupId>com.alibaba</groupId>
   <artifactId>druid-spring-boot-starter</artifactId>
   <version>1.1.10</version>
</dependency>
```

添加 Web 依赖是因为需要在启动的时候保持容器运行，同时在项目中添加了 Web 访问，内容如下：

```
@RestController
public class UserController {
    @Autowired
    private UserRepository userRepository;
    @RequestMapping("/getUsers")
    public List<User> getUsers() {
        List<User> users=userRepository.findAll();
        return users;
    }
}
```

内容比较简单获取所有的用户信息并展示出来。

Application 中添加以下信息：

```
# 初始化大小、最小、最大链接数
spring.datasource.druid.initial-size=3
spring.datasource.druid.min-idle=3
spring.datasource.druid.max-active=10

# 配置获取连接等待超时的时间
spring.datasource.druid.max-wait=60000

#  StatViewServlet 配置
spring.datasource.druid.stat-view-servlet.login-username=admin
spring.datasource.druid.stat-view-servlet.login-password=admin

# 配置 StatFilter
spring.datasource.druid.filter.stat.log-slow-sql=true
spring.datasource.druid.filter.stat.slow-sql-millis=2000
```

好了，这样就成功的在 JPA 项目中配置好了 Druid 的使用。启动项目先访问地址：
http://localhost:8080/getUsers，再访问 http://localhost:8080/druid，查看 SQL 执行记录，如下：

会发现有 create table addres... 和 drop table if exist... 这样的 SQL 语句，这是因为我们将 JPA 的策略设置为 create，spring.jpa.properties.hibernate.hbm2ddl.auto=create，意味着每次重启的时候对会重新创建表，方便我们在测试的时候使用。

JPA + Druid + 多数据源

因为 Druid 官方还没有针对 Spring Boot 2.0 进行优化，在某些场景下使用就会出现问题，比如在 JPA 多数据源的情况下直接使用 Druid 提供的 druid-spring-boot-starter 就会报错，既然 druid-spring-boot-starter 不支持，那么我们就使用 Druid 的原生包进行封装。

在前面示例项目 spring-boot-multi-Jpa 的基础上进行改造。

添加依赖

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid</artifactId>
    <version>1.1.10</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
<!--<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid-spring-boot-starter</artifactId>
    <version>1.1.10</version>
</dependency>-->
```

删掉对 druid-spring-boot-starter 包的依赖，添加 Druid 的依赖包，添加 log4j 的原因是因为 Druid 依赖于 log4j 打印日志。

多数据源配置

配置文件我们做这样的设计，将多个数据源相同配置抽取出来共用，每个数据源个性配置信息单独配置。

数据库1的配置，以 spring.datasource.druid.one 开头：

```
spring.datasource.druid.one.url=jdbc:mysql://localhost:3306/test1?serverTimezone=U
TC&useUnicode=true&characterEncoding=utf-8&useSSL=true
spring.datasource.druid.one.username=root
spring.datasource.druid.one.password=root
spring.datasource.druid.one.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.druid.one.initialSize=3
spring.datasource.druid.one.minIdle=3
spring.datasource.druid.one.maxActive=10
```

数据库2的配置，以 spring.datasource.druid.two 开头：

```
spring.datasource.druid.two.url=jdbc:mysql://localhost:3306/test2?serverTimezone=U
TC&useUnicode=true&characterEncoding=utf-8&useSSL=true
spring.datasource.druid.two.username=root
spring.datasource.druid.two.password=root
spring.datasource.druid.two.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.druid.two.initialSize=6
spring.datasource.druid.two.minIdle=20
spring.datasource.druid.two.maxActive=30
```

多数据源的共同配置，以 spring.datasource.druid 开头，是多个数据源的公共配置项。

```
配置获取连接等待超时的时间
spring.datasource.druid.maxWait=60000
#配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒
spring.datasource.druid.timeBetweenEvictionRunsMillis=60000
#配置一个连接在池中最小生存的时间，单位是毫秒
spring.datasource.druid.minEvictableIdleTimeMillis=600000
spring.datasource.druid.maxEvictableIdleTimeMillis=900000
spring.datasource.druid.validationQuery=SELECT 1 FROM DUAL
#y检测连接是否有效
spring.datasource.druid.testWhileIdle=true
#是否在从池中取出连接前进行检验连接池的可用性
spring.datasource.druid.testOnBorrow=false
#是否在归还到池中前进行检验连接池的可用性
spring.datasource.druid.testOnReturn=false
# 是否打开 PSCache，
spring.datasource.druid.poolPreparedStatements=true
#并且指定每个连接上 PSCache 的大小
spring.datasource.druid.maxPoolPreparedStatementPerConnectionSize=20
#配置监控统计拦截的 filters
spring.datasource.druid.filters=stat,wall,log4j
#通过 connectProperties 属性来打开 mergeSQL 功能，慢 SQL 记录
spring.datasource.druid.connectionProperties=druid.stat.mergeSql=true;druid.stat.s
lowSqlMillis=600
```

更多的配置信息请参考这里。

我们定义一个 DruidConfig 来加载所有的公共配置项，如下：

```
@Component
@ConfigurationProperties(prefix="spring.datasource.druid")
public class DruidConfig {
    protected String url;
    protected String username;
    protected String password;
    protected String driverClassName;
    protected int initialSize;
    protected int minIdle;
    protected int maxActive;
    protected int maxWait;
    protected int timeBetweenEvictionRunsMillis;
    protected long minEvictableIdleTimeMillis;
    protected long maxEvictableIdleTimeMillis;
    protected String validationQuery;
    protected boolean testWhileIdle;
    protected boolean testOnBorrow;
    protected boolean testOnReturn;
    protected boolean poolPreparedStatements;
    protected int maxPoolPreparedStatementPerConnectionSize;
    protected String filters;
    protected String connectionProperties;
    // 省略 getter setter
}
```

再定义一个 DruidOneConfig 来加载数据源 1 的配置项，并继承 DruidConfig：

```
@Component
@ConfigurationProperties(prefix="spring.datasource.druid.one")
public class DruidOneConfig  extends  DruidConfig{
    private String url;
    private String username;
    private String password;
    private String driverClassName;
    private int initialSize;
    private int minIdle;
    private int maxActive;
    // 省略 getter setter
}
```

再定义一个 DruidTwoConfig 来加载数据源 2 的配置项并继承 DruidConfig，代码和 DruidOneConfig 类基本一致。

启动时加载

创建类 DruidDBConfig 在启动的时候注入配置的多数据源信息。

```
@Configuration
public class DruidDBConfig {
    @Autowired
    private DruidConfig druidOneConfig;
    @Autowired
    private DruidConfig druidTwoConfig;
    @Autowired
    private DruidConfig druidConfig;

}
```

在类中创建 initDruidDataSource() 方法，初始化 Druid 数据源各属性。各个数据库的个性化配置从 config 对读取，公共配置项从 druidConfig 对象获取。

```
private DruidDataSource initDruidDataSource(DruidConfig config) {
    DruidDataSource datasource = new DruidDataSource();

    datasource.setUrl(config.getUrl());
    datasource.setUsername(config.getUsername());
    datasource.setPassword(config.getPassword());
    datasource.setDriverClassName(config.getDriverClassName());
    datasource.setInitialSize(config.getInitialSize());
    datasource.setMinIdle(config.getMinIdle());
    datasource.setMaxActive(config.getMaxActive());

    // common config
    datasource.setMaxWait(druidConfig.getMaxWait());
    datasource.setTimeBetweenEvictionRunsMillis(druidConfig.getTimeBetweenEviction
RunsMillis());
    datasource.setMinEvictableIdleTimeMillis(druidConfig.getMinEvictableIdleTimeMi
llis());
    datasource.setMaxEvictableIdleTimeMillis(druidConfig.getMaxEvictableIdleTimeMi
llis());
    datasource.setValidationQuery(druidConfig.getValidationQuery());
    datasource.setTestWhileIdle(druidConfig.isTestWhileIdle());
    datasource.setTestOnBorrow(druidConfig.isTestOnBorrow());
    datasource.setTestOnReturn(druidConfig.isTestOnReturn());
    datasource.setPoolPreparedStatements(druidConfig.isPoolPreparedStatements());
    datasource.setMaxPoolPreparedStatementPerConnectionSize(druidConfig.getMaxPool
PreparedStatementPerConnectionSize());
    try {
        datasource.setFilters(druidConfig.getFilters());
    } catch (SQLException e) {
        logger.error("druid configuration initialization filter : {0}", e);
    }
    datasource.setConnectionProperties(druidConfig.getConnectionProperties());

    return datasource;
}
```

启动时调用 initDruidDataSource() 方法构建不同的数据源。

```
@Bean(name = "primaryDataSource")
public DataSource dataSource() {
    return initDruidDataSource(druidOneConfig);
}

@Bean(name = "secondaryDataSource")
@Primary
public DataSource secondaryDataSource() {
    return initDruidDataSource(druidTwoConfig);
}
```

下面通过不同的数据源构建 entityManager，最后注入到 Repository 的逻辑和以前一样，变化的地方只是在数据源构建和开启监控页面。

开启监控页面

因为我们使用了原生的 Druid 包，因此需要手动开启监控、配置统计相关内容。

```
@Configuration
public class DruidConfiguration {
    @Bean
    public ServletRegistrationBean<StatViewServlet> druidStatViewServlet() {
        ServletRegistrationBean<StatViewServlet> servletRegistrationBean = new Ser
vletRegistrationBean<>(new StatViewServlet(), "/druid/*");
        servletRegistrationBean.addInitParameter("loginUsername", "admin");
        servletRegistrationBean.addInitParameter("loginPassword", "admin");
        servletRegistrationBean.addInitParameter("resetEnable", "false");
        return servletRegistrationBean;
    }

    @Bean
    public FilterRegistrationBean<WebStatFilter> druidStatFilter() {
        FilterRegistrationBean<WebStatFilter> filterRegistrationBean = new FilterR
egistrationBean<>(new WebStatFilter());
        filterRegistrationBean.setName("DruidWebStatFilter");
        filterRegistrationBean.addUrlPatterns("/*");
        filterRegistrationBean.addInitParameter("exclusions", "*.js,*.gif,*.jpg,*.
png,*.css,*.ico,/druid/*");
        return filterRegistrationBean;
    }
}
```

配置完成后，重启启动项目访问地址 http://localhost:8080/druid/sql.html 就可以看到有两个数据源的 SQL 操作语句，证明多数据源 SQL 监控配置成功。

到此 JPA + Druid + 多数据源的集成完成了。

总结

Druid 是一款非常优秀的数据库连接池开源软件，使用 Druid 提供的 druid-spring-boot-starter 可以非常简单地对 Druid 进行集成。Druid 提供了很多预置的功能，非常方便我们对 SQL 进行监控、分析。Druid 对 Spring Boot 2.0 的支持还不够完善，对于使用 Druid 的特殊场景，可以使用 Druid 原生包自行进行封装。

> 点击这里下载源码。