# 微服务电商平台 Spring Cloud 技术白皮书

广州朗尊软件科技有限公司

| | |
|---|---|
| **文档名称** | 《微服务电商平台Spring Cloud技术白皮书》 |
| **内容描述** | 采用Spring cloud的技术架构重构现有的电商平台 |

| **修订历史** | | | | |
|---|---|---|---|---|
| **日期** | **版本** | **修订者** | **修订说明** | **评审人员** |
| 2018-10.10 | V1.0 | Newway | 初稿 | |
| 2019-10.8 | V2.0 | Newway | 升级架构 | |
| | | | | |

# 一、 引言

## 1.1 目的

本文档全面系统地表述业务平台系统的构架，并通过使用多种视图来从不同角度描述系统的各个主要方面，以满足相关涉众（客户、设计人员、测试人员等）对目标系统的不同关注焦点。

本文档记录并表述了架构师对系统构架方面做出的重要决策：

1、 客户关注项目能否按期完成并满足业务需求、用户关心系统的可靠性和可用性；需要对互相冲突的需求进行协商和权衡。

2、 客户决策人关注架构对开发及测试进度的影响，对工程实施效率的影响，对成本控制的影响，对采用技术是否成熟可靠的影响，对满足业务需求发展可扩展性的影响等；

3、 项目经理关注架构对设计开发的指导能否保证各个小组的任务能否在很大程度上都独立完成，各部分交互是否规范和可控；据此规划和分配项目资源，并跟踪每个小组的进展。

4、 设计人员关注实现架构目标的具体设计策略；需要对各组成部分的资源争用进行沟通。

5、 开发人员关注架构规定的不能违反的限制、和可以利用的自由。

6、 测试人员关注按照构架设计系统的总体测试框架；并可以指定必须组合在一起的个部分正确的黑盒行为。

7、 工程人员关注架构对实施部署的指导意义。

8、 质量人员关注架构为符合性检查提供的基础，以确保实现忠于架构规定；

### 1.2 释义和缩略语

| 缩写、术语 | 解释 |
| --- | --- |
| 微服务 | 微服务架构从本质上说其实就是分布式架构，与其说是一种新架构，不如说是一种微服务架构风格。 |
| Spring Cloud | Spring Cloud 是一个微服务框架，相比 Dubbo 等 RPC 框架，Spring Cloud 提供的全套的分布式系统解决方案。 |
| Dubbo | Dubbo 是阿里提供的一个分布式服务框架，致力于提供高性能和透明化的 RPC 远程服务调用方案，以及 SOA 服务治理方案。 |

## 二、 系统概述

### 2.1 需求概述

采用 spring cloud 实现电商平台。实现前后端分离和数据库的分库。

特点是：

易扩展，按需伸缩。

易于开发和维护。

服务拆分粒度更细，有利于资源重复利用，有利于提高开发效率。

适于互联网时代，产品迭代周期更短。

### 2.2 系统环境

jdk：jdk8
os：可在 windows，linux,unix 等主流操作系统上运行
database：mysql

### 2.3 建设目标

一、 高可用、高并发、海量数据、高稳定性、容灾机制

二、 易扩展

三、 高效开发

1. 一种设计风格，将原本独立的系统拆分成为多个小型服务，这些小型服务都在各自独立的进程

中运行。

2. 被拆分的每一个小型服务都围绕滋生系统中的某一项耦合度较高的业务活着功能进行构建，每个服务都有自己的数据存储，业务，自动化测试以及独立部署机制。

3. 服务与服务之间调用采用 RESTful 规范的 api 进行通信。

# 三、 系统总体架构设计

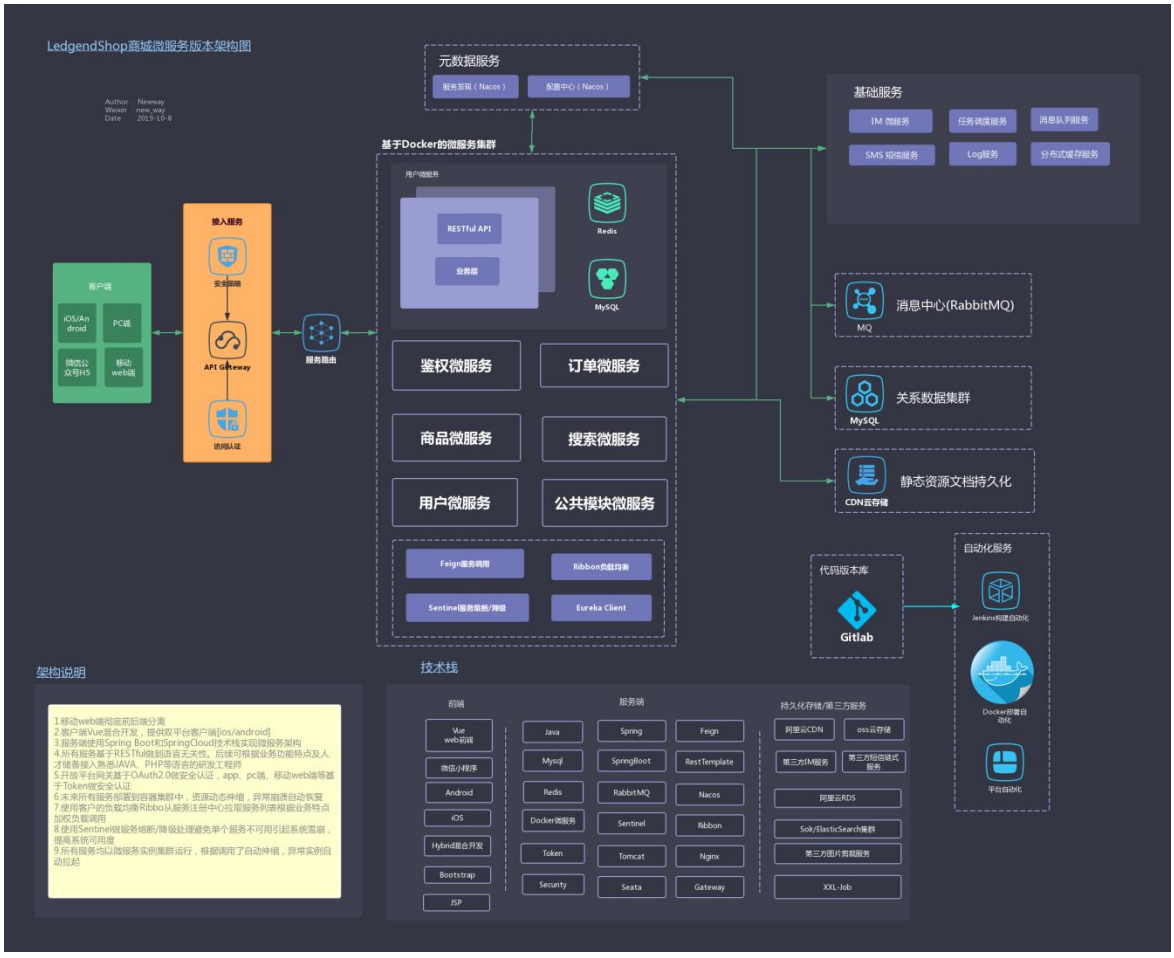
## 3.1 架构技术选型

| | | |
|---|---|---|
| 基础技术栈 | 开发框架 | Spring Boot、Spring MVC 、Spring |
| | 持久化框架 | Spring Jdbc Template |
| | 缓存 | Redis /云服务/ caffeine/页面缓存 |
| | Session 同步 | Spring Session + Redis |
| | 消息中间件 | RabbitMQ /阿里云服务 |
| | 服务器安全框架 | Spring Security，OAuth 2.0 |
| | 分布式定时任务 | xxl-job |
| | 分布式文件存储 | 云服务(OSS)/ FASTDFS |
| | 分布式全文检索引擎 | Elasticsearch |
| | 前端页面渲染 | Vue/JSP |
| 微服务技术栈 | 分布式服务框架 | Spring Cloud |
| | 服务注册与发现 | Nacos |
| | 配置中心 | Nacos |
| | 断路器 | Sentinel |
| | 本地服务负载均衡 | Ribbon |
| | 服务接口通信 | FeignClient |
| | API 网关 | Spring Cloud gateway |

| | | |
|---|---|---|
| 部署、运维技术栈 | 应用服务器 | Tomcat/ undertow |
| | 前端负载均衡 | Nginx/Lvs/云服务 |
| | 服务部署和弹性伸缩 | Jenkins+Docker |
| | 可持续自动化测试与集成 | Jenkins+SVN |
| | 日志收集框架 | Elasticsearch、Logstash、filebeat、Kibana |
| | 集群部署 | DockersSwarm / K8S |
| | 数据库集群 | Mysql Cluster /云服务 RDS |

## 3.2 技术总体架构

根据电商分布式业务平台概念性架构，根据具体业务需求，分析得出的架构（总体）如下：



## 整体设计思路

1、我们把整个系统根据业务拆分成若干个子系统或微服务。

2、每个子系统可以部署多个应用，多个应用之间使用负载均衡。

3、服务注册中心 Nacos，所有的服务都在注册中心注册，负载均衡也是通过在注册中心注册的服务来使用一定策略来实现。

4、所有的客户端都通过同一个网关地址访问后台的服务，通过路由配置 Spring cloud gateway 网关来判断一个 URL 请求由哪个服务处理。请求转发到服务上的时候使用负载均衡 Ribbon

5、服务之间采用 feign 进行调用。

6、使用断路器 hystrix，及时处理服务调用时的超时和错误，防止由于其中一个服务的问题而导致整体系统的瘫痪。

7、需要一个监控功能，监控每个服务调用花费的时间等。

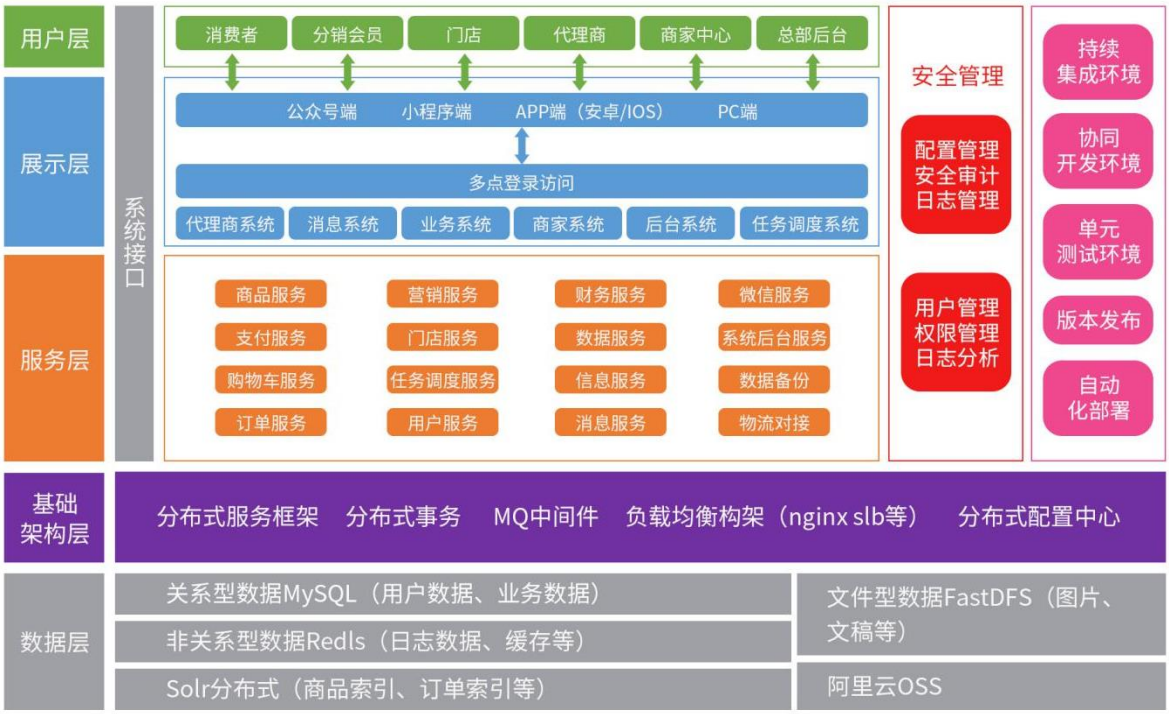
8、使用 Nacos 进行统一的配置管理，Nacos 支持动配置参数和版本历史。

9、Hystrix，监控和断路器。我们只需要在服务接口上添加 Hystrix 标签，就可以实现对这个接口的监控和断路器功能。

10、Hystrix Dashboard，监控面板，他提供了一个界面，可以监控各个服务上的服务调用所消耗的时间等。

11、Turbine，监控聚合，使用 Hystrix 监控，我们需要打开每一个服务实例的监控信息来查看。而 Turbine 可以帮助我们把所有的服务实例的监控信息聚合到一个地方统一查看，这样就不需要挨个打开一个个的页面查看。

### 3.3 业务架构图




从业务架构图可以看出：

1、系统做了前后端分离，web 前端通过 http/https 协议调用微服务的 API 网关 spring cloud gateway，由 API 网关再经过路由服务调用相应的微服务 ；前端的 web 可按业务进行划分，web 应用可以做集群部署;

2、不同微服务之间通过 REST 方式互相调用，每一个微服务可以独立部署，也可以做成集群方式;

3、微服务之间通过消息中间件实现消息交互机制。

二、配套服务与功能实现 ：

1、需要进行相应的自动化服务实现，包括自动化构建、自动化安装部署、自动化测试、自动化平台发布（Docker 实现）;

2、管理服务，对于微服务架构，必须配套相应的监控与管理服务、日志管理服务等;

3、协作服务，运用 DevOps 思想提升开发、测试、运维的高效沟通与协作，实现开发与运维的一体化 。

# 四、 架构要点

## 4.1 前后端分离

为满足未来互联网业务平台的海量访问需求，及应对目前复杂的应用环境，本系统平台采用分布式架构体系。分布式系统可以解决当前业务复杂多变的应用环境的诸多问题：

1. 协作与互联的需要，分布式系统的重要作用之一就是能够整合分散的信息和服务。
2. 性能和可伸缩的需要，分布式系统可以应对业务不断增长带来的负荷的增长，可以通过分布式系统的扩展和组合能力以获得性能和系统处理能力的提升。
3. 容错的需要，分布式系统可以提供冗余的服务，这种冗余性有助于将单点失败的影响控制在最小范围内，可以显著提高系统在出现部分故障时的可靠性。
4. 经济性的需要，分布式系统的水平扩展能力同时可以通过增加较廉价的机器来提升系统处理能力，比集中式的大型机具有更高的性价比。

## 4.2 服务拆分原则

1. 粒度微小

   根据业务功能划分服务粒度，总的原则是服务内部高内聚，服务之间低耦合

2. 责任单一

   每个服务只做一件事，即单一职责原则

3. 隔离性原则

   每个服务相互隔离，且不互相影响

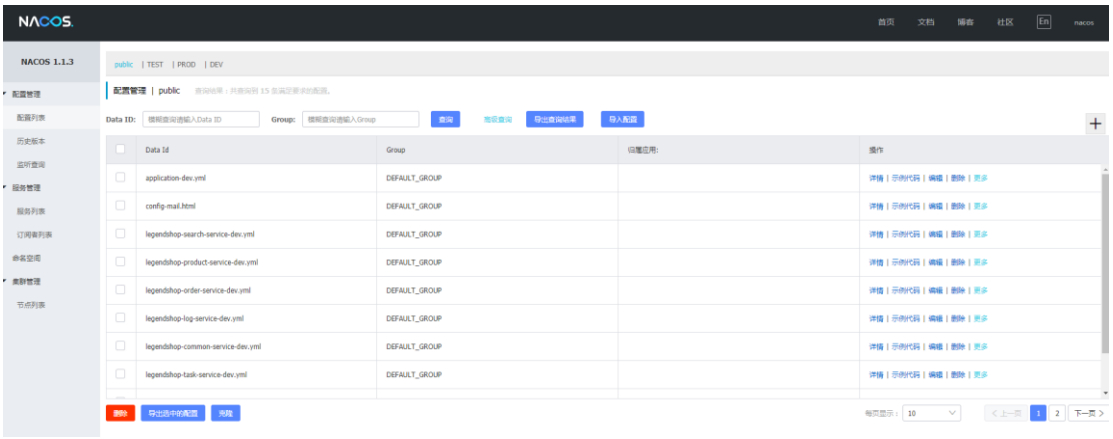
4. 业务无关优先原则：

   基础服务，是一些基础组件，与具体的业务无关。比如：短信服务、邮件服务。这里的服务最容易划分出来做微服务，也是我们第一优先级分离出来的服务。

## 4.3 统一配置管理

实现各微服务的统一参数配置以及版本管理，可采用 Spring Cloud Config 或者阿里巴巴的 Nacos 配置中心。

Nacos 配置管理平台,能够集中化管理应用不同环境、不同集群的配置,配置修改后能够实时推送到应用端,并且具备规范的权限、流程治理等特性。




## 4.4 服务间调用的权限验证

一般我们的 API 接口都需要某种授权才能访问，登陆成功以后，然后通过 token 或者 cookie 等方式才能调用接口。

使用 Spring Cloud Netfix 框架的话，登录的时候，把登录请求转发到相应的用户服务上，登陆成功后，会设置 cookie 或 header token 等。然后客户端接下来的请求就会带着这些验证信息，从 gateway 网关传到相应的服务上进行验证。

gateway 网关在把请求转发到后台的服务的时候，会默认把一些 header 传到服务端，如：Cookie、Set-Cookie、Authorization。这样，客户端请求的相关 headers 就可以传递到服务端，服务端设置的 cookie 也可以传到客户端。

## 4.5 监控服务框架

使用 Hystrix 组件进行服务的监控，使用 Nagios 进行服务器等资源的监控。

1、Hystrix，监控和断路器。我们只需要在服务接口上添加 Hystrix 标签，就可以实现对这个接口的监控和断路器功能。

2、Hystrix Dashboard，监控面板，他提供了一个界面，可以监控各个服务上的服务调用所消耗的时间等。

3、Turbine，监控聚合，使用 Hystrix 监控，我们需要打开每一个服务实例的监控信息来查看。而 Turbine 可以帮助我们把所有的服务实例的监控信息聚合到一个地方统一查看。这样就不需要挨个打开一个个的页面一个个查看。

## 4.6 统一日志管理

不同微服务部署在不同节点上，登录每个节点查看日志是比较麻烦的，同时对于需要关联多个微服务日志联合查看分析的情况将更加麻烦。伴随节点数量的增加，如果没有合适的管理机制与工具，定位问题、发现问题的复杂性将越来越大，将成指数级增长，因此需要进行统一日志管理。

1、为了加强应用日志的监控、管理，建议标准、统一的日志监控方式，有必要对应用系统日志制定统一的书写规范，实现各应用日志标准化管理和维护；

2、开发并使用统一的日志组件，为所有微服务提供统一的日志服务，由 log4j 或 Slf4j 封装；

3、在每个服务节点上部署日志采集 Agent 组件，由此 Agent 进行日志的采集与转发；

4、建立统一的日志中心，所有日志写入日志中心。

可以采用的是 EFK 集合框架进行日志的收集。

## 4.7 负载均衡方案

负载均衡的作用(解决的问题)：

1.解决并发压力，提高应用处理性能(增加吞吐量，加强网络处理能力);

2.提供故障转移，实现高可用;

3.通过添加或减少服务器数量，提供网站伸缩性(扩展性);

4.安全防护;(负载均衡设备上做一些过滤，黑白名单等处理)

软负载均衡分类:

根据实现技术不同，可分为 DNS 负载均衡，HTTP 负载均衡，IP 负载均衡，链路层负载均衡等。

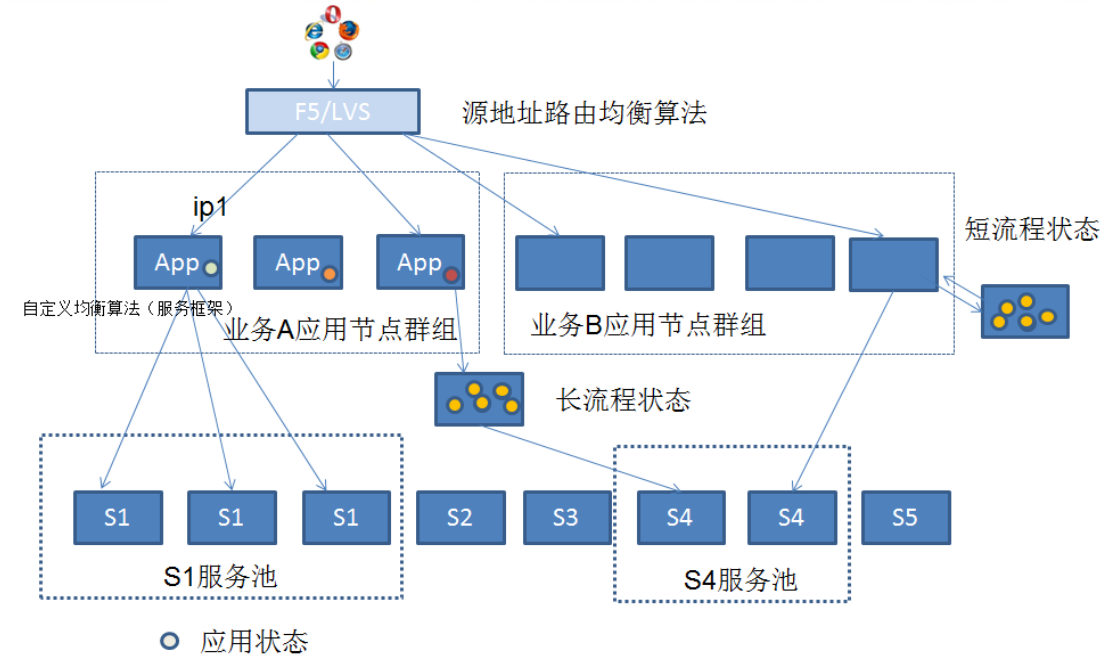
软件负载均衡

1. Nginx/阿里云 SLB + Tomcat 应用服务器
2. 采用 docker swarm 进行集群部署

**硬件负载：**

业界领先的有两款，F5 和 A10;

应用前端负载均衡的分布式架构见下图，前端使用 F5/LVS 做负载均衡。




## 4.8 持续集成

1、持续集成：每个微服务独立执行持续集成。

2、版本集成：由统一的集成工具，实现自动化的版本集成，将所有微服务集成到统一的版本发布包中。

3、持续集成可制作多种场景的版本，包括测试环境、开发环境、生产环境。

4、统计测试覆盖率等指标数据。

5、工具：Jenkins。

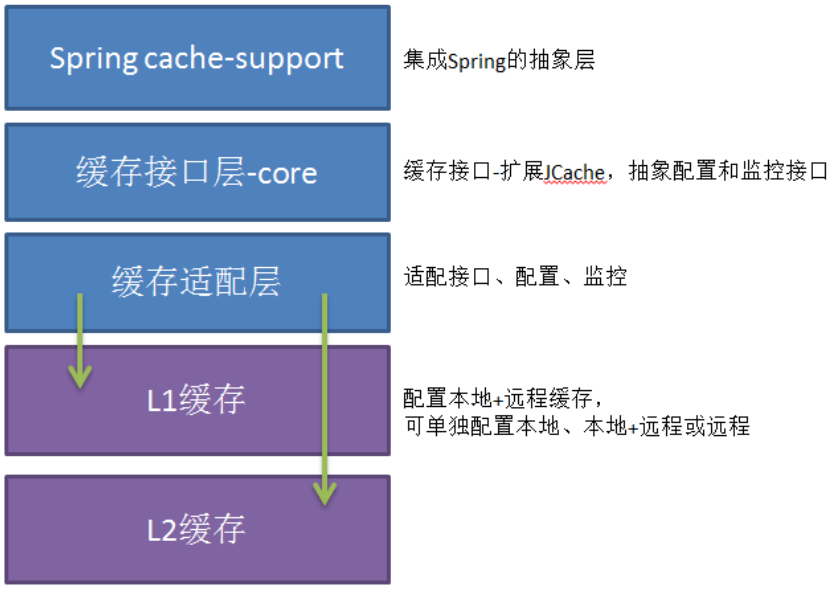
## 4.9 缓存框架

缓存框架目的是构建统一的缓存访问组件及可管理的缓存服务。

缓存框架提供统一的缓存 API，并封装多个第三方缓存组件，屏蔽第三方缓存的差异，在替换第三方缓存组件时，代码不需要任何修改。

目前采用常用的互联网缓存技术栈: redis/memcahced/ehcached，目前系统采用 redis 作为二级缓存，采用 caffeine 作为 JVM 的内部缓存，同时也支持页面级别的缓存。

缓存框架的系统结构图见下图所示。




## 4.10 数据访问框架

数据访问框架提供了数据访问的过滤、路由功能，以及数据的分片(Sharding，采用 sharding jdbc 实现)的功能。

数据访问的路由可以根据配置方法名称参数的规则路由到不同的 Connection；分片的算法规则包括普通 Hash、取模 Hash、顺序 Hash、读写分离 Hash 和混合分布多种策略可供选择。

其特点包括：

1. 应用层不需要关心驱动的变化；
2. 数据源与应用层隔离；
3. 应用层不需要关心分表分库与物理层映射关系。