·**·去中心化治理**

在实施微服务架构的过程中，通过采用轻量级的契约定义接口，使得我们对于服务本

身所采用的具体技术平台不再那么敏感，这样整个微服务架构系统中的各个组件就能针对

不同的业务特点从而选择不同的技术平台。

·**·去中心化管理数据**

我们在实施微服务架构时，都希望让每一个服务来管理其自有的数据库，这就是数据

管理的去中心化。

在去中心化过程中，我们除了将原数据库中的存储内容拆分到新的同平台的其他数据

库实例中之外（例如把原本存储在一个库中的所有表拆分后，分别存储到不同的库或者用

户中），也可以将一些具有特殊结构或业务特性的数据存储到一些其他技术的数据库实例

中（例如把日志信息存储到MongoDB中或把用户登录信息存储到Redis中）。

虽然数据管理的去中心化可以让数据管理更加细致化，通过采用更合适的技术可让数

据存储和性能达到最优。但是，由于数据存储于不同的数据库实例中后，数据的一致性也

成为微服务架构中亟待解决的问题之一。分布式的事务管理本身就是一个难题，所以在微

服务架构中，我们更加强调在各服务之间进行“无服务”的调用，对于数据的一致性，只要

求数据在最后的处理状态保持一致即可。若在过程中发现错误，通过补偿机制来进行处理，

使得错误数据能够达到最终的一致性。

·**·容错设计**

在单体应用中，一般不存在单个组件故障而其他部件还在运行的情况，通常都是一个

挂掉从而导致全部都挂掉。而在微服务架构中，由于服务都运行在独立的进程中，所以存

在部分服务出现故障，而其他服务正常运行的情况。

所以，在微服务架构中，快速检测出故障源并尽可能地自动恢复服务是必须被设计和

考虑的。通常，我们都希望在每个服务中实现监控和日志记录的组件，比如服务状态、断

路器状态、吞吐量、网路延迟等关键数据的仪表盘等。

·**·Spring Cloud简介**

Spring Cloud是一个基于Spring Boot实现的微服务架构开发工具。它为微服务架构中

涉及的配置管理、服务治理、断路器、智能路由、微服务、控制总线、全局锁、决策竞选、

分布式会话和集群状态管理等操作提供了一种简单的开发方式。

Spring Cloud（下文简称SC）包含了多个子项目（针对分布式系统中涉及的多个不同

开源产品，还可能会增加），如下所述。

·SC Config：配置管理工具，支持使用Git存储配置内容，可以使用它实现应用配置的

外部化存储，并支持客户端配置信息更新、加密/解密配置内容等。

·SC Netflix：核心组件，对多个Netflix OSS开源套件进行整合。

·Eureka：服务治理组件，包含服务注册中心、服务注册与发现机制的实现。

·Hystrix：容错管理组件，实现断路器模式，帮助服务依赖中出现的延迟和为故障

提供强大的容错能力。

·Ribbon：客户端负载均衡的服务调用组件。

·Feign：基于Ribbon和Hystrix的声明式服务调用组件。

·Zuul：网关组件，提供智能路由、访问过滤等功能。

·Archaius：外部化配置组件。

·SC Bus：事件、消息总线，用于传播集群中的状态变化或事件，

·SC Cluster：针对Zookeeper、Redis、Hazeleast、Consul的选举算法和通用模式的

实现。

·SC Stream：通过Redis、Rabbit或者Kafka实现的消费微服务，可以通过简单的声明

式模型来发送和接收消息。

·SC AWS