SpringCloud文档整理

目 录

[一、SpringCloud简介 3](#_Toc511237066)

[二、开源Demo 6](#_Toc511237067)

[SpringCloud： 6](#_Toc511237068)

[LCN分布式事务框架Demo: 9](#_Toc511237069)

[Oauth2认证中心： 11](#_Toc511237070)

[1)介绍： 11](#_Toc511237071)

[配置授权类型（Grant Types）： 12](#_Toc511237072)

[Oauth2数据库表说明 14](#_Toc511237073)

[Spring Security TokenStore 15](#_Toc511237074)

[三、面临的问题及解决方案 16](#_Toc511237075)

[四、rabbitMQ配置 17](#_Toc511237076)

[1.安装Erlang 17](#_Toc511237077)

[2.安装RabbitMQ 18](#_Toc511237078)

[3.配置 18](#_Toc511237079)

# 一、SpringCloud简介

springcloud为开发人员提供了快速构建分布式系统的一些工具，包括配置管理、服务发现、断路器、路由、微代理、事件总线、全局锁、决策竞选、分布式会话等等。它运行环境简单，可以在开发人员的电脑上跑。

[**服务的注册与发现（Eureka）**](http://blog.csdn.net/forezp/article/details/69696915)

eureka是一个高可用的组件，它没有后端缓存，每一个实例注册之后需要向注册中心发送心跳（因此可以在内存中完成）。

当client向server注册时，它会提供一些元数据，例如主机和端口，URL，主页等。可以在server中对所有注册的client进行监控与管理，Eureka server 从每个client实例接收心跳消息。 如果心跳超时，则通常将该实例从注册server中删除。

[**服务消费者（Feign）**](http://blog.csdn.net/forezp/article/details/69808079)

Feign是一个声明式的伪Http客户端，它使得写Http客户端变得更简单。使用Feign，只需要创建一个接口并注解。它具有可插拔的注解特性，可使用Feign 注解和JAX-RS注解。Feign支持可插拔的编码器和解码器。Feign默认集成了Ribbon，并和Eureka结合，默认实现了负载均衡的效果。

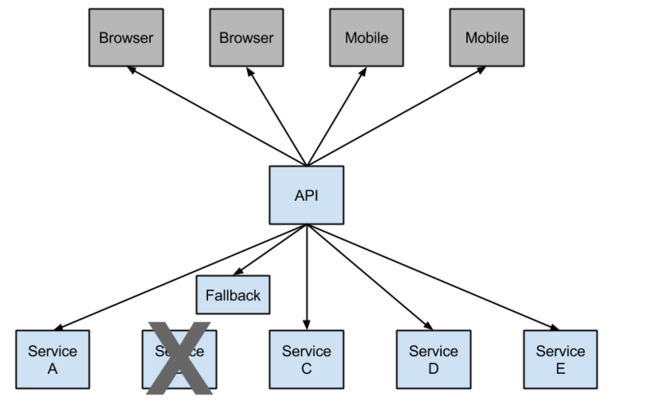
并且可以@FeignClient注解消费指定的模块或服务。



[**断路器（Hystrix）**](http://blog.csdn.net/forezp/article/details/69934399)

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在Spring Cloud可以用RestTemplate+Ribbon和Feign来调用。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身的原因，服务并不能保证100%可用，如果单个服务出现问题，调用这个服务就会出现线程阻塞，此时若有大量的请求涌入，Servlet容器的线程资源会被消耗完毕，导致服务瘫痪。服务与服务之间的依赖性，故障会传播，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。

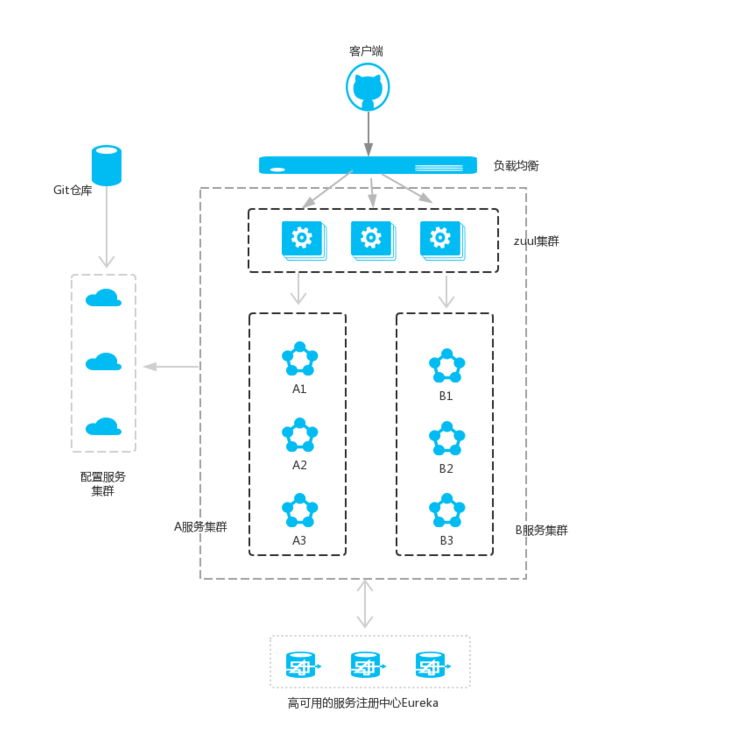
较底层的服务如果出现故障，会导致连锁故障。当对特定的服务的调用的不可用达到一个阀值（Hystric 是5秒20次） 断路器将会被打开。断路打开后，可用避免连锁故障，fallback方法可以直接返回一个固定值。



[**路由网关(zuul)**](http://blog.csdn.net/forezp/article/details/69939114)

在微服务架构中，需要几个基础的服务治理组件，包括服务注册与发现、服务消费、负载均衡、断路器、智能路由、配置管理等，由这几个基础组件相互协作，共同组建了一个简单的微服务系统。

Zuul的主要功能是路由转发和过滤器。路由功能是微服务的一部分，比如／api/user转发到到user服务，/api/shop转发到到shop服务。zuul默认和Ribbon结合实现了负载均衡的功能。



# 二、开源Demo

## SpringCloud：

1）地址：<https://github.com/zhangxd1989/spring-boot-cloud>

2）技术栈：

①Spring boot - 微服务的入门级微框架，用来简化 Spring 应用的初始搭建以及开发过程。

②Eureka - 云端服务发现，一个基于 REST 的服务，用于定位服务，以实现云端中间层服务发现和故障转移。

③Spring Cloud Config - 配置管理工具包，让你可以把配置放到远程服务器，集中化管理集群配置，目前支持本地存储、Git 以及 Subversion。

④Hystrix - 熔断器，容错管理工具，旨在通过熔断机制控制服务和第三方库的节点,从而对延迟和故障提供更强大的容错能力。

⑤Zuul - Zuul 是在云平台上提供动态路由，监控，弹性，安全等边缘服务的框架。Zuul 相当于是设备和 Netflix 流应用的 Web 网站后端所有请求的前门。

⑥Spring Cloud Bus - 事件、消息总线，用于在集群（例如，配置变化事件）中传播状态变化，可与 Spring Cloud Config 联合实现热部署。

⑦Spring Cloud Sleuth - 日志收集工具包，封装了 Dapper 和 log-based 追踪以及 Zipkin 和 HTrace 操作，为 SpringCloud 应用实现了一种分布式追踪解决方案。

⑧Ribbon - 提供云端负载均衡，有多种负载均衡策略可供选择，可配合服务发现和断路器使用。

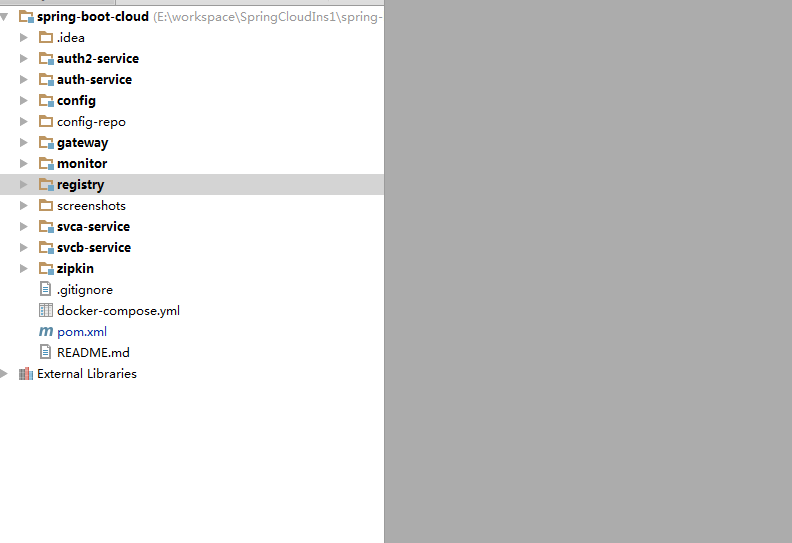
⑨Turbine - Turbine 是聚合服务器发送事件流数据的一个工具，用来监控集群下 hystrix 的 metrics 情况。

⑩Spring Cloud Stream - Spring 数据流操作开发包，封装了与 Redis、Rabbit、Kafka 等发送接收消息。

⑪Feign - Feign 是一种声明式、模板化的 HTTP 客户端。

⑫Spring Cloud OAuth2 - 基于 Spring Security 和 OAuth2 的安全工具包，为你的应用程序添加安全控制。

3)项目结构:



registry - 服务注册与发现

config - 外部配置

monitor - 监控

zipkin - 分布式跟踪

gateway - 代理所有微服务的接口网关

auth-service - OAuth2 认证服务

svca-service - 业务服务A

svcb-service - 业务服务B

4)启动方式：

①配置 rabbitmq

②修改 hosts 将主机名指向到本地，127.0.0.1 registry config monitor rabbitmq auth-service或者修改各服务配置文件中的相应主机名为本地 ip

③启动 registry、config、monitor、zipkin

④启动 gateway、auth-service、svca-service、svcb-service

## LCN分布式事务框架Demo:

1)资料地址：

[**https://blog.csdn.net/zhou280648461/article/details/78918747**](https://blog.csdn.net/zhou280648461/article/details/78918747)

<https://github.com/codingapi/tx-lcn>

<https://github.com/codingapi/tx-lcn/wiki/LCN%E5%8E%9F%E7%90%86>

2)简介：

LCN分布式事务框架其本身并不创建事务，而是基于对本地事务的协调从而达到事务一致性的效果。

创建事务组：是指在事务发起方开始执行业务代码之前先调用TxManager创建事务组对象，然后拿到事务标示GroupId的过程。

添加事务组：添加事务组是指参与方在执行完业务方法以后，将该模块的事务信息添加通知给TxManager的操作。

关闭事务组：是指在发起方执行完业务代码以后，将发起方执行结果状态通知给TxManager的动作。

当执行完关闭事务组的方法以后，TxManager将根据事务组信息来通知相应的参与模块提交或回滚事务。

事务控制原理

LCN事务控制原理是由事务模块TxClient下的代理连接池与TxManager的协调配合完成的事务协调控制。

TxClient的代理连接池实现了javax.sql.DataSource接口，并重写了close方法，事务模块在提交关闭以后TxClient连接池将执行"假关闭"操作，等待TxManager协调完成事务以后在关闭连接。

事务补偿机制：

事务补偿是指在执行某个业务方法时，本应该执行成功的操作却因为服务器挂机或者网络抖动等问题导致事务没有正常提交，此种场景就需要通过补偿来完成事务，从而达到事务的一致性。

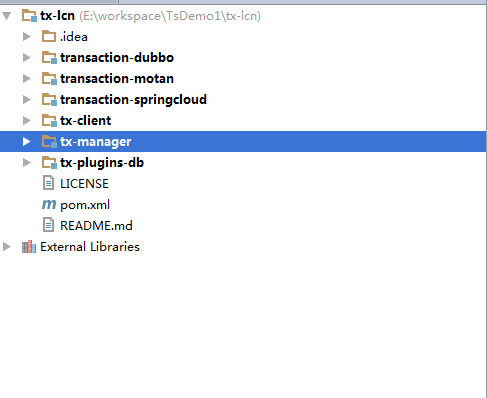
补偿机制的触发条件：

当执行关闭事务组步骤时，若发起方接受到失败的状态后将会把该次事务识别为待补偿事务，然后发起方将该次事务数据异步通知给TxManager。TxManager接受到补偿事务以后先通知补偿回调地址，然后再根据是否开启自动补偿事务状态来补偿或保存该次切面事务数据。

补偿事务机制

LCN的补偿事务原理是模拟上次失败事务的请求，然后传递给TxClient模块然后再次执行该次请求事务。

3)项目结构：



4)启动方式：

配置tx-manager下的配置文件，注册进入springcloud注册中心。

在tx-manager下启动TxManagerApplication。

## Oauth2认证中心：

### 1)介绍：

授权类型的列表

authorization\_code：授权码类型。

implicit：隐式授权类型。

password：资源所有者（即用户）密码类型。

client\_credentials：客户端凭据（客户端ID以及Key）类型。

refresh\_token：通过以上授权获得的刷新令牌来获取新的令牌。

可以用 @EnableAuthorizationServer 注解来配置OAuth2.0 授权服务机制，通过使用@Bean注解的几个方法一起来配置这个授权服务。下面咱们介绍几个配置类，这几个配置是由Spring创建的独立的配置对象，它们会被Spring传入AuthorizationServerConfigurer中：

ClientDetailsServiceConfigurer：用来配置客户端详情服务（ClientDetailsService），客户端详情信息在这里进行初始化，你能够把客户端详情信息写死在这里或者是通过数据库来存储调取详情信息。

AuthorizationServerSecurityConfigurer：用来配置令牌端点(Token Endpoint)的安全约束.

AuthorizationServerEndpointsConfigurer：用来配置授权（authorization）以及令牌（token）的访问端点和令牌服务(token services)。

### 配置授权类型（Grant Types）：

授权是使用 AuthorizationEndpoint 这个端点来进行控制的，你能够使用 AuthorizationServerEndpointsConfigurer 这个对象的实例来进行配置(AuthorizationServerConfigurer 的一个回调配置项，见上的概述) ，如果你不进行设置的话，默认是除了资源所有者密码（password）授权类型以外，支持其余所有标准授权类型的（RFC6749），我们来看一下这个配置对象有哪些属性可以设置吧，如下列表：

authenticationManager：认证管理器，当你选择了资源所有者密码（password）授权类型的时候，请设置这个属性注入一个 AuthenticationManager 对象。

userDetailsService：如果啊，你设置了这个属性的话，那说明你有一个自己的 UserDetailsService 接口的实现，或者你可以把这个东西设置到全局域上面去（例如 GlobalAuthenticationManagerConfigurer 这个配置对象），当你设置了这个之后，那么 "refresh\_token" 即刷新令牌授权类型模式的流程中就会包含一个检查，用来确保这个账号是否仍然有效，假如说你禁用了这个账户的话。

authorizationCodeServices：这个属性是用来设置授权码服务的（即 AuthorizationCodeServices 的实例对象），主要用于 "authorization\_code" 授权码类型模式。

implicitGrantService：这个属性用于设置隐式授权模式，用来管理隐式授权模式的状态。

tokenGranter：这个属性就很牛B了，当你设置了这个东西（即 TokenGranter 接口实现），那么授权将会交由你来完全掌控，并且会忽略掉上面的这几个属性，这个属性一般是用作拓展用途的，即标准的四种授权模式已经满足不了你的需求的时候，才会考虑使用这个。

配置客户端详情信息（Client Details)：

ClientDetailsServiceConfigurer (AuthorizationServerConfigurer 的一个回调配置项，见上的概述) 能够使用内存或者JDBC来实现客户端详情服务（ClientDetailsService），有几个重要的属性如下列表：

clientId：（必须的）用来标识客户的Id。

secret：（需要值得信任的客户端）客户端安全码，如果有的话。

scope：用来限制客户端的访问范围，如果为空（默认）的话，那么客户端拥有全部的访问范围。

authorizedGrantTypes：此客户端可以使用的授权类型，默认为空。

authorities：此客户端可以使用的权限（基于Spring Security authorities）。

管理令牌（Managing Token）：

AuthorizationServerTokenServices 接口定义了一些操作使得你可以对令牌进行一些必要的管理，在使用这些操作的时候请注意以下几点：

当一个令牌被创建了，你必须对其进行保存，这样当一个客户端使用这个令牌对资源服务进行请求的时候才能够引用这个令牌。

当一个令牌是有效的时候，它可以被用来加载身份信息，里面包含了这个令牌的相关权限。

当你自己创建 AuthorizationServerTokenServices 这个接口的实现时，你可能需要考虑一下使用 DefaultTokenServices 这个类，里面包含了一些有用实现，你可以使用它来修改令牌的格式和令牌的存储。默认的，当它尝试创建一个令牌的时候，是使用随机值来进行填充的，除了持久化令牌是委托一个 TokenStore 接口来实现以外，这个类几乎帮你做了所有的事情。并且 TokenStore 这个接口有一个默认的实现，它就是 InMemoryTokenStore ，如其命名，所有的令牌是被保存在了内存中。除了使用这个类以外，你还可以使用一些其他的预定义实现，下面有几个版本，它们都实现了TokenStore接口：

InMemoryTokenStore：这个版本的实现是被默认采用的，它可以完美的工作在单服务器上（即访问并发量压力不大的情况下，并且它在失败的时候不会进行备份），大多数的项目都可以使用这个版本的实现来进行尝试，你可以在开发的时候使用它来进行管理，因为不会被保存到磁盘中，所以更易于调试。

JdbcTokenStore：这是一个基于JDBC的实现版本，令牌会被保存进关系型数据库。使用这个版本的实现时，你可以在不同的服务器之间共享令牌信息，使用这个版本的时候请注意把"spring-jdbc"这个依赖加入到你的classpath当中。

JwtTokenStore：这个版本的全称是 JSON Web Token（JWT），它可以把令牌相关的数据进行编码（因此对于后端服务来说，它不需要进行存储，这将是一个重大优势），但是它有一个缺点，那就是撤销一个已经授权令牌将会非常困难，所以它通常用来处理一个生命周期较短的令牌以及撤销刷新令牌（refresh\_token）。另外一个缺点就是这个令牌占用的空间会比较大，如果你加入了比较多用户凭证信息。JwtTokenStore 不会保存任何数据，但是它在转换令牌值以及授权信息方面与 DefaultTokenServices 所扮演的角色是一样的。

配置授权端点的URL（Endpoint URLs）：

AuthorizationServerEndpointsConfigurer 这个配置对象(AuthorizationServerConfigurer 的一个回调配置项，见上的概述) 有一个叫做 pathMapping() 的方法用来配置端点URL链接，它有两个参数：

第一个参数：String 类型的，这个端点URL的默认链接。

第二个参数：String 类型的，你要进行替代的URL链接。

以上的参数都将以 "/" 字符为开始的字符串，框架的默认URL链接如下列表，可以作为这个 pathMapping() 方法的第一个参数：

/oauth/authorize：授权端点。

/oauth/token：令牌端点。

/oauth/confirm\_access：用户确认授权提交端点。

/oauth/error：授权服务错误信息端点。

/oauth/check\_token：用于资源服务访问的令牌解析端点。

/oauth/token\_key：提供公有密匙的端点，如果你使用JWT令牌的话。

### Oauth2数据库表说明

<http://andaily.com/spring-oauth-server/db_table_description.html>

oauth\_client\_details：

在项目中,主要操作oauth\_client\_details表的JdbcClientDetailsService.java. 也可以根据实际的需要,去扩展或修改该类的实现.

oauth\_client\_token：

该表用于在客户端系统中存储从服务端获取的token数据, 在[spring-oauth-server](http://git.oschina.net/shengzhao/spring-oauth-server)项目中未使用到. 对oauth\_client\_token表的主要操作在JdbcClientTokenServices.java类中,。

oauth\_access\_token：

在项目中,主要操作oauth\_access\_token表的对象是JdbcTokenStore.java.

oauth\_refresh\_token：

 在项目中,主要操作oauth\_refresh\_token表的对象是JdbcTokenStore.java. (与操作oauth\_access\_token表的对象一样);更多的细节请参考该类. 如果客户端的grant\_type不支持refresh\_token,则不会使用该表.

oauth\_code：

在项目中,主要操作oauth\_code表的对象是JdbcAuthorizationCodeServices.java.. 只有当grant\_type为"authorization\_code"时,该表中才会有数据产生; 其他的grant\_type没有使用该表.

### [Spring Security TokenStore](https://blog.csdn.net/DuShiWoDeCuo/article/details/78929333)

<https://blog.csdn.net/DuShiWoDeCuo/article/details/78929333>

1. InMemoryTokenStore

这个是OAuth2默认采用的实现方式。在单服务上可以体现出很好特效（即并发量不大，并且它在失败的时候不会进行备份），大多项目都可以采用此方法。根据名字就知道了，是存储在内存中，毕竟存在内存，而不是磁盘中，调试简易。

2. JdbcTokenStore

这个是基于JDBC的实现，令牌（Access Token）会保存到数据库。这个方式，可以在多个服务之间实现令牌共享。

3. JwtTokenStore

jwt全称 JSON Web Token。这个实现方式不用管如何进行存储（内存或磁盘），因为它可以把相关信息数据编码存放在令牌里。JwtTokenStore 不会保存任何数据，但是它在转换令牌值以及授权信息方面与 DefaultTokenServices 所扮演的角色是一样的。

4. RedisTokenStore

由于TokenStore作用就是对于OAuth2令牌持久化接口，而我们在实际开发中，对于内存的使用是慎之又慎，而对于存储到数据库也是根据项目需求进行调配。因此就想，可不可以用redis来进行存储持久化我们的OAuth2令牌。OAuth2还有那些实现了TokenStore的，找到了一个RedisTokenStore。

# 三、面临的问题及解决方案

1.关于webservice

springboot可以集成webservice框架作为微服务注册进入springcloud。所有的webservice调用都交给微服务处理。

2.分布式事务

LCN分布式事务框架可以解决分布式事务问题。

3.微服务的管理监控

springcloud可以通过hystrix面板监控所有注册的微服务心跳状态与请求状态。

4.容灾容错与熔断

hystrix熔断器，springCloud可以通过熔断器为服务增加容错能力，降级处理。

springcloud通过turbine来监控集群下hystrix的metrics情况。

5.权限验证

springboot可用整合shiro使用权限验证

springboot可以整合Oauth2做授权管理

6.日志处理

kafka做大数据量的日志处理

7.单点登录

配合Oauth2授权管理可以做单点登录

8.跨域问题

JSONP、NGINX代理、CORS

CORS:

在zuul网关路由层解决跨域问题，只要在网关配置。

https://blog.csdn.net/qq1013598664/article/details/74277240

zuul跨域异常：<https://segmentfault.com/a/1190000010722941>(The 'Access-Control-Allow-Origin' header contains multiple values '\*, \*', but only one is allowed)

9.中间件

10.前后端分离

11.操作回溯

12.问题定位

13.快速部署启动

14.CMS内容管理

# 四、rabbitMQ配置

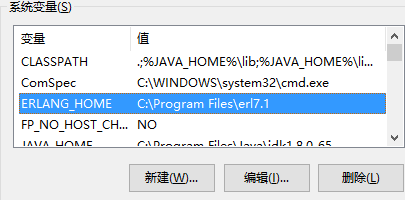
## 1.安装Erlang

所以在安装rabbitMQ之前，需要先安装Erlang 。

http://www.erlang.org/downloads

全部点击“下一步”。

有的选择其他的安装方式，可能需要添加一下系统环境变量（正常安装的也要检查下）：



## 2.安装RabbitMQ

下载运行[rabbitmq-server-3.6.5](https://pan.baidu.com/s/1cqpG0u" \t "_blank) ，需要其他版本或者32位系统的，可以去[官网](http://www.rabbitmq.com/download.html" \t "_blank)下载。

依旧可以不改变默认进行安装。

需要注意：默认安装的RabbitMQ 监听端口是5672

## 3.配置

激活 RabbitMQ's Management Plugin

使用RabbitMQ 管理插件，可以更好的可视化方式查看Rabbit MQ 服务器实例的状态。

打开命令窗口：

输入命令：

"C:\Program Files\RabbitMQ Server\rabbitmq\_server-3.6.5\sbin\rabbitmq-plugins.bat" enable rabbitmq\_management

使用浏览器打开[http://localhost:15672](http://localhost:15672/" \t "_blank)访问Rabbit Mq的管理控制台，使用刚才创建的账号登陆系统：



