SpringCloud之八

消息总线SpringCloudBus

一、Spring Cloud Bus 介绍

Spring Cloud Bus 将分布式的节点用轻量的消息代理连接起来。它可以用于广播配置文件的更改或者服务之间的通讯，也可以用于监控。本文要讲述的是用Spring Cloud Bus实现通知微服务架构的配置文件的更改。

一个关键的想法是，Bus就像一个扩展的Spring Boot应用程序的分布式执行器，但也可以用作应用程序之间的通信渠道。当前唯一的实现是使用AMQP代理作为传输，但是相同的基本功能集（还有一些取决于传输）在其他传输的路线图上。

官网截图如下：

消息总线：

在微服务架构的系统中，我们通常会使用轻量级的消息代理来构建一个共用的消息主题让系统中所有微服务实例都连接上来，由于该主题中产生的消息会被所有实例监听和消费，所以我们称它为消息总线。

消息代理(Message Broker)：

是一种消息验证、传输、路由的架构模式。它在应用程序之间起到通信调度并最小化应用之间的依赖的作用，使得应用程序可以高效地解耦通信过程。消息代理是一个中间件产品，它的核心是一个消息的路由程序，用来实现接收和分发消息， 并根据设定好的消息处理流来转发给正确的应用。 它包括独立的通信和消息传递协议，能够实现组织内部和组织间的网络通信。设计代理的目的就是为了能够从应用程序中传入消息，并执行一些特别的操作，下面这些是在企业应用中，我们经常需要使用消息代理的场景：

- 将消息路由到一个或多个目的地。
- 消息转化为其他的表现方式。
- 执行消息的聚集、消息的分解，并将结果发送到它们的目的地，然后重新组合响应返回给消息用户。
- 调用Web服务来检索数据。
- 响应事件或错误。
- 使用发布 - 订阅模式来提供内容或基千主题的消息路由。

目前已经有非常多的开源产品可以供大家使用， 比如：

- ActiveMQ
- Kafka
- RabbitMQ
- RocketMQ
- 等.....

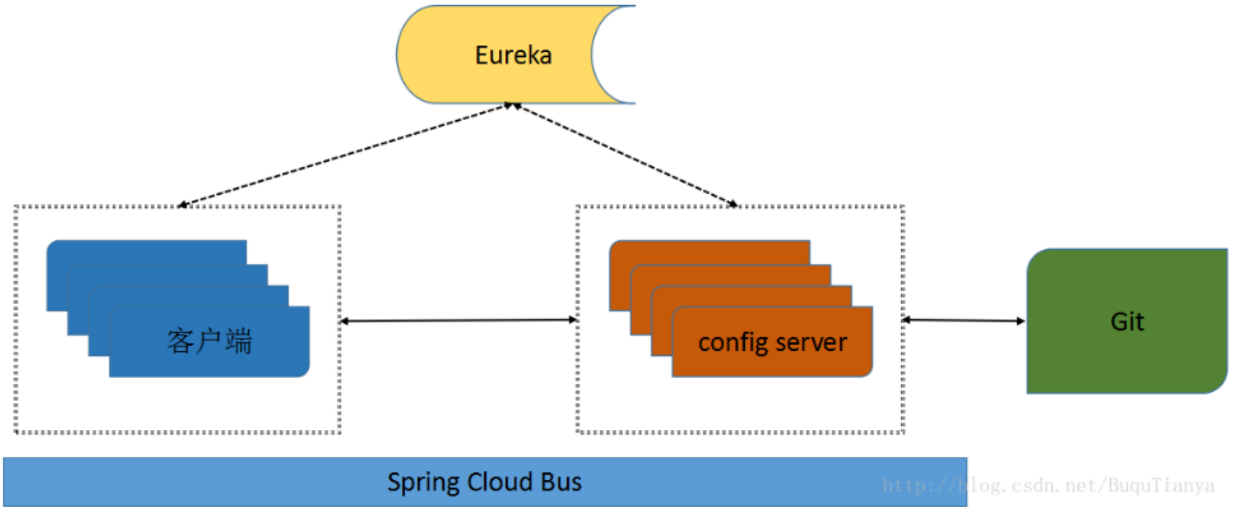
Spring Cloud Bus联通：

spring cloud bus在整个后端服务中起到联通的作用，联通后端的多台服务器。
我们为什么需要他做联通呢？
后端服务器一般都做了集群化，很多台服务器，而且在大促活动期经常发生服务的扩容、缩容、上线、下线。这样，后端服务器的数量、IP就会变来变去，如果我们想进行一些线上的管理和维护工作，就需要维护服务器的IP。
比如我们需要更新配置、比如我们需要同时失效所有服务器上的某个缓存，都需要向所有的相关服务器发送命令，也就是调用一个接口。

你可能会说，我们一般会采用zookeeper的方式，统一存储服务器的ip地址，需要的时候，向对应服务器发送命令。这是一个方案，但是他的解耦性、灵活性、实时性相比消息总线都差那么一点。

总的来说，就是在我们需要把一个操作散发到所有后端相关服务器的时候，就可以选择使用cloud bus了。

使用cloud bus之后，我们的服务端架构会变成这样：



当前spring cloud bus提供了两个可用的接口:

1. /bus/env用于设置某一个配置项
2. /bus/refresh用于刷新所有绑定到刷新点的配置项。

这两个接口是使用spring boot actuator方式发布出来的（可以参见：深入SpringBoot:自定义Endpoint一文），接收到消息后会使用spring的stream框架（可以参考：张开涛的解Spring事件驱动模型一文）把消息传播到所有注册的相关服务器。

/bus/env的参数格式：

```
name=&value=&destination=
```

/bus/refresh的参数格式：

```
destination=
```

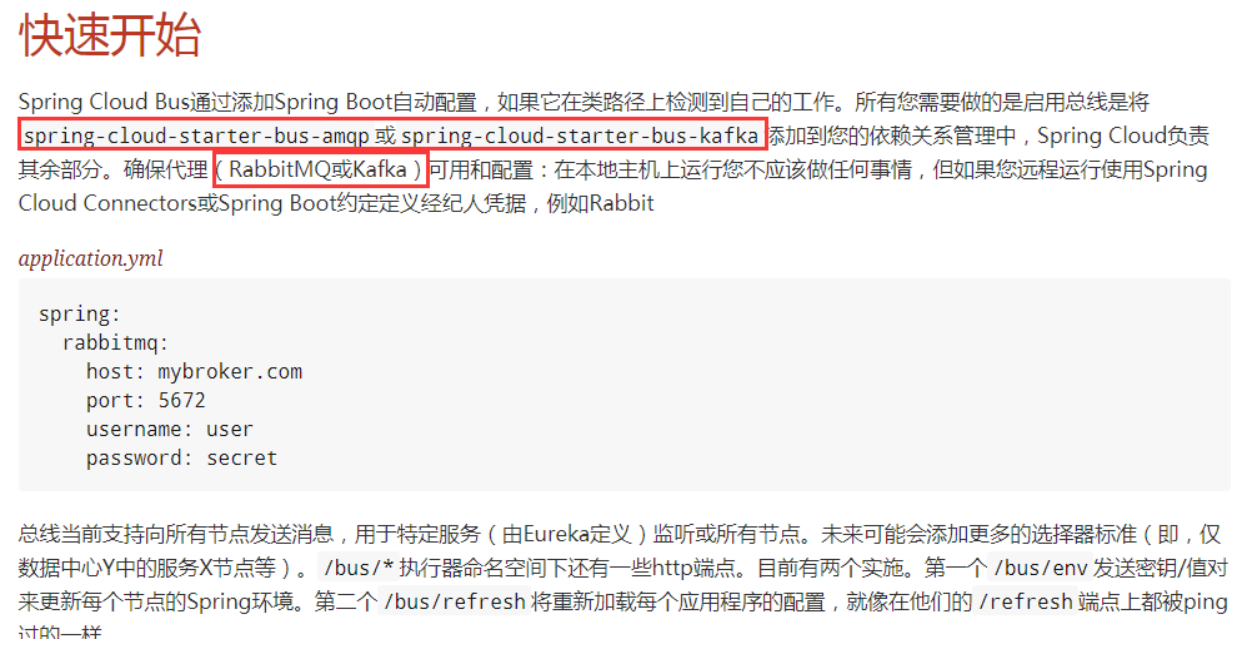
当然了，上述的destination参数都可以不提供。

二、Spring Cloud Bus 的使用

废话不多说，直接开始造，使劲造。联系才能更好的理解消息总线。

2.1、准备

官方文档中说需要结合消息代理框架使用，这也就意味着需要安装RabbitMQ或者Kafka了。如下图官方文档截图：



快速开始

Spring Cloud Bus通过添加Spring Boot自动配置，如果它在类路径上检测到自己的工作。所有您需要做的是启用总线是将 `spring-cloud-starter-bus-amqp` 或 `spring-cloud-starter-bus-kafka` 添加到您的依赖关系管理中，Spring Cloud负责其余部分。确保代理（RabbitMQ或Kafka）可用和配置：在本地主机上运行您不应该做任何事情，但如果您远程运行使用Spring Cloud Connectors或Spring Boot约定定义经纪人凭据，例如Rabbit

application.yml

```
spring:
  rabbitmq:
    host: mybroker.com
    port: 5672
    username: user
    password: secret
```

总线当前支持向所有节点发送消息，用于特定服务（由Eureka定义）监听或所有节点。未来可能会添加更多的选择器标准（即，仅数据中心Y中的服务X节点等）。 `/bus/*` 执行器命名空间下还有一些http端点。目前有两个实施。第一个 `/bus/env` 发送密钥/值对来更新每个节点的Spring环境。第二个 `/bus/refresh` 将重新加载每个应用程序的配置，就像在他们的 `/refresh` 端点上都被ping过的一样

趁此机会，简单回顾了解ActiveMQ，Kafka，RocketMQ，RabbitMQ这几个MQ吧！

2.1.1、ActiveMQ

官网地址：http://activemq.apache.org/

ActiveMQ是比较老牌的消息系统，当然了不一定是大家第一个熟知的消息系统，因为现在电商、互联网规模越来越大，不断进入程序员眼帘的大多是Kafka和RocketMQ。ActiveMQ出现的要比他们早，而且涵盖的功能也特别全，路由、备份、查询、事务、集群等等。他的美中不足是不能支撑超大规模、超高并发的互联网应用，ActiveMQ的并发承受能力在百万级别，大概500次/s的消息频率。

2.1.2、Kafka

官网地址是：http://kafka.apache.org/

Kafka是新一代的消息系统，相对于ActiveMQ来说增加了分片功能，类似于数据库分库分表，一台Broker仅负责一部分数据收发，从而使得他的伸缩性特别好，通过增加Broker就可以不断增加处理能力。一般来说，Kafka被用来处理日志流，作为流计算的接入点。在电商的订单、库存等系统里边一般不用，主要顾虑的是Kafka异步刷盘机制可能导致数据丢失。当然，对于数据丢失这一点不同的工程师也有不同的看法，认为Kafka的Master-Slave的多写机制，完全能够避免数据丢失。

2.1.3、RocketMQ

官网地址是：http://rocketmq.apache.org/

RocketMQ是阿里开源的一款消息系统，开发的初衷就是要支撑阿里庞大的电商系统。RocketMQ和Kafka有很多相似之处，由于RocketMQ开发中很大程度上参考了Kafka的实现。RocketMQ同样提供了优秀的分片机制，RocketMQ的分片比Kafka的分片有所增强，区分了绝对有序和非绝对有序两种选项。另外RocketMQ采用的是同步刷盘，一般认为不会造成数据丢失。

2.1.4、RabbitMQ

网址：http://www.rabbitmq.com/

RabbitMQ类似于ActiveMQ也是一个相对小型的消息系统，他的优势在于灵活的路由机制，可以进行自由配置。

从网上找到一个对比图，可以参考参考：

		kafka	RocketMQ	RabbitMQ
定位	设计定位	系统间的数据流管道，实时数据处理。 例如：常规的消息系统、网站活性跟踪，监控数据，志收集、处理等	非日志的可靠消息传输。 例如：订单，交易，充值，流计算，消息推送，日流式处理，binglog分发等	可靠消息传输。和RocketMQ类似。
基础对比	成熟度	日志领域成熟	成熟	成熟
	所属社区／公司	Apache	Alibaba开发，已加入到Apache下	Mozilla Public License
	社区活跃度	高	中	高
	API完备性	高	高	高
	文档完备性	高	高	高
	开发语言	Scala	Java	Erlang
	支持协议	一套自行设计的基于TCP的二进制协议	自己定义的一 套 (社区提供 JMS一不成熟)	AMQP
	客户端语言	C/C++、Python、Go、Erlang、.NET、Ruby、Node.js PHP等	Java	Java、C、 C++、 Python、 PHP、Perl 等
	持久化方式	磁盘文件	磁盘文件	内存、文件
可用性、靠性比较	部署方式	单机／集群	单机／集群	单机／集群
	集群管理	zookeeper	name server	
	选主方式	从ISR中自动选举一个leader	不支持自动选主。通过设定brokername、brokerId实现。brokername相同，brokerid=0时为maser，他为slave	最早加入集群的broker
	可用性	非常高 分布式、主从	非常高 分布式、主从	高 主从，采用镜像模式实现，数据量大时可能产生性能瓶颈
	主从切换	自动切换 N个副本，允许N-1个失效；master失效以后自动从is中选择一个主；	不支持自动切换 master失效以后不能向master发送信息，consume大概30s（默认）可以感知此事件，此后从slave消费；如果master无法恢复，异步复制时可能出现部分信息丢失	自动切换 最早加入集群的slave会成为master；因为新加入的slave不会同步master之前的数据，所以可能会出现部分数据丢失
	数据可靠性	很好 支持producer单条发送、同步刷盘、同步复制，但这场景下性能明显下降。	很好 producer单条发送，broker端支持同步刷盘、异步刷盘，同步双写，异步复制。	好 producer支持同步／异步ack。支持队列数据持久化，镜像模式中支持主从同步
	消息写入性能	非常好 每条10个字节测试：百万条/s	很好 每条10个字节测试：单机单broker约7w/s，单机3broker约12w/s	RAM约为RocketMQ的1/2， Disk的性能约为RAM性能的1/3
	性能的稳定性	队列/分区多时性能不稳定，明显下降。 消息堆积时性能稳定	队列较多、消息堆积时性能稳定	消息堆积时，性能不稳定、明显下降
	单机支持的队列数	单机超过64个队列/分区，Load会发生明显的飙高现象，队列越多，load越高，发送消息响应时间变长	单机支持最高5万个队列，Load不会发生明显变化	依赖于内存
	堆积能力	非常好 消息存储在log中，每个分区一个log文件	非常好 所有消息存储在同一个commit log中	一般 生产者、消费者正常时，性能表现稳定；消费者不消费时，性能不稳定
	复制备份	消息先写入leader的log，followers从leader中pull pull到数据以后先ack leader，然后写入log中。 ISR中维护与leader同步的列表，落后太多的follwer被删除掉	同步双写 异步复制：slave启动线程从master中拉数据	普通模式下不复制； 镜像模式下：消息先到mster，然后写到slave上。入集群之前的消息不会被复制到新的slave上。
	消息投递实时性	毫秒级 具体由consumer轮询间隔时间决定	毫秒级 支持pull、push两种模式，延时通常在毫秒级	毫秒级

功能对比	顺序消费	支持顺序消费 但是一台Broker宕机后，就会产生消息乱序	支持顺序消费 在顺序消息场景下，消费失败时消费队列将会暂停	支持顺序消费 但是如果一个消费失败，此消息的顺序会被打乱
	定时消息	不支持	开源版本仅支持定时Level	不支持
	事务消息	不支持	支持	不支持
	Broker端消息过滤	不支持	支持 通过tag过滤，类似于子topic	不支持
	消息查询	不支持	支持 根据MessageId查询 支持根据MessageKey查询消息	不支持
	消费失败重试	不支持失败重试 offset存储在consumer中，无法保证。 0.8.2版本后支持将offset存储在zk中	支持失败重试 offset存储在broker中	支持失败重试
	消息重新消费	支持通过修改offset来重新消费	支持按照时间来重新消息	
	发送端负载均衡	可自由指定	可自由指定	需要单独loadbalancer支持
	消费并行度	消费并行度和分区数一致	顺序消费：消费并行度和分区数一致 乱序消费：消费服务器的消费线程数之和	镜像模式下其实也是从master消费
	消费方式	consumer pull	consumer pull / broker push	broker push
	批量发送	支持 默认producer缓存、压缩，然后批量发送	不支持	不支持
	消息清理	指定文件保存时间，过期删除	指定文件保存时间，过期删除	可用内存少于40%（默认），触发gc，gc时找到相邻的两个文件，合并right文件到left。
	访问权限控制	无	无	类似数据库一样，需要配置用户名密码
运维	系统维护	Scala语言开发，维护成本高	java语言开发，维护成本低	Erlang语言开发，维护成本高
	部署依赖	zookeeper	nameserver	Erlang环境
	管理后台	官网不提供，第三方开源管理工具可供使用；不用重新开发	官方提供，rocketmq-console	官方提供rabbitmqadmin
	管理后台功能	**Kafka Web Conslole** Brokers列表；Kafka 集群中 Topic列表，及对应的Partition、LogSize等信息；Topic对应的Consumer Groups、Offset、Lag等信息； 生产和消费流量图、消息预览 **KafkaOffsetMonitor：** Kafka集群状态；Topic、Consumer Group列表；图形展示topic和consumer之间的关系；图形化展示consumer的Offset、Lag等信息 **Kafka Manager** 管理几个不同的集群；监控集群的状态(topics, brokers，副本分布，分区分布)；产生分区分配(Generate partition assignments)基于集群的当前态；重新分配分区	Cluster、Topic、Connection、NameServ、Messa Broker、Offset、Consumer	overview、connections、channels、exchanges、queues、admin
总结	优点	1、在高吞吐、低延迟、高可用、集群热扩展、集群容错上有非常好的表现； 2、producer端提供缓存、压缩功能，可节省性能，提高效率。 3、提供顺序消费能力 4、提供多种客户端语言 5、生态完善，在大数据处理方面有大量配套的设施。	1、在高吞吐、低延迟、高可用上有非常好的表现，消息堆积时，性能也很好。 2、api、系统设计都更加适在业务处理的场景。 3、支持多种消费方式。 4、支持broker消息过滤。 5、支持事务。 6、提供消息顺序消费能力；consumer可以水平扩展，消费能力很强。 7、集群规模在50台左右，单日处理消息上百亿；经历过大数据量的考验，比较稳定可靠。	1、在高吞吐量、高可用上较前两者有所不如。 2、支持多种客户端语言；支持amqp协议。 3、由于erlang语言的特性，性能也比较好；使用RAM模式时，性能很好。 4、管理界面较丰富，在互联网公司也有较大规模的应用；
	缺点	1、消费集群数目受到分区数目的限制。 2、单机topic多时，性能会明显降低。 3、不支持事务	1、相比于kafka，使用者较少，生态不够完善。消息堆积、吞吐率上也有所不如。 2、不支持主从自动切换，master失效后，消费者要一定的时间才能感知。 3、客户端只支持Java	1、erlang 语言难度较大。集群不支持动态扩展。 2、不支持事务、消息吞吐能力有限 3、消息堆积时，性能会明显降低

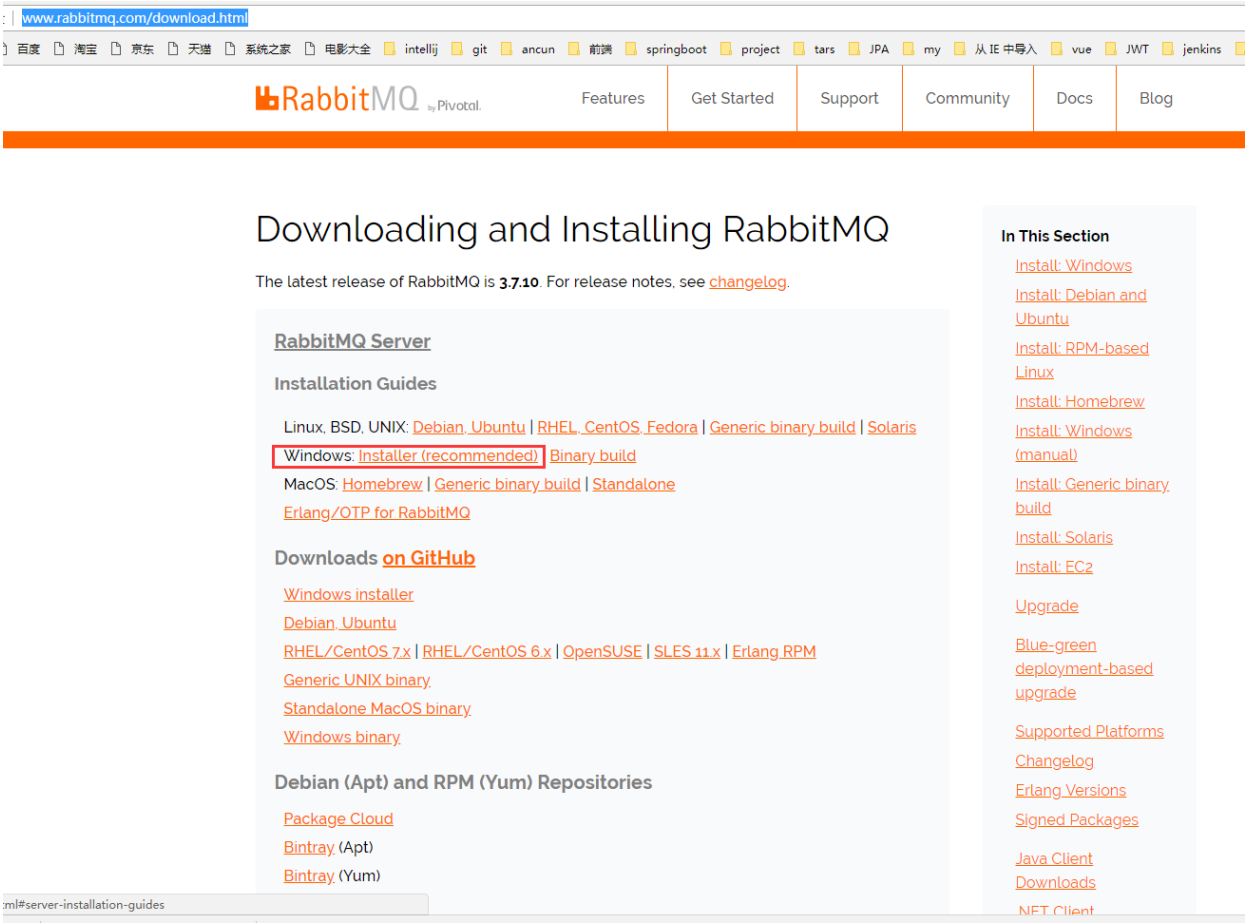
2.1.5、Redis

官网：https://redis.io/

Redis的pub/sub功能，由于Redis是内存级的系统，所以速度和单机的并发能力是上述四个消息系统不能比拟的，但是也是由于内存存储的缘故，在消息的保障上就更弱一些。据说新浪博客系统选择了Redis的pub/sub作为消息系统，不能不说艺高人胆大。

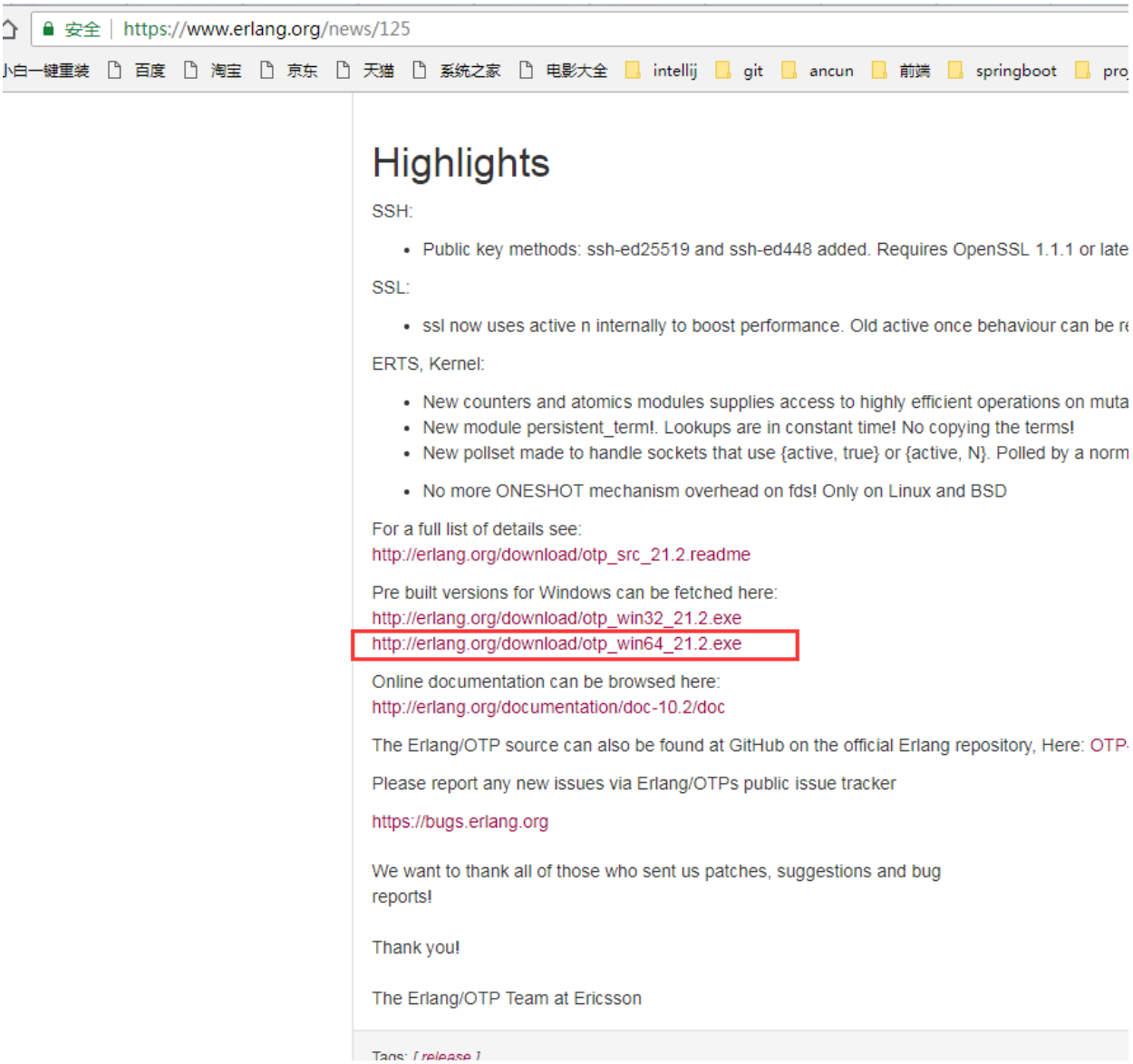
2.1.6、安装RabbitMQ

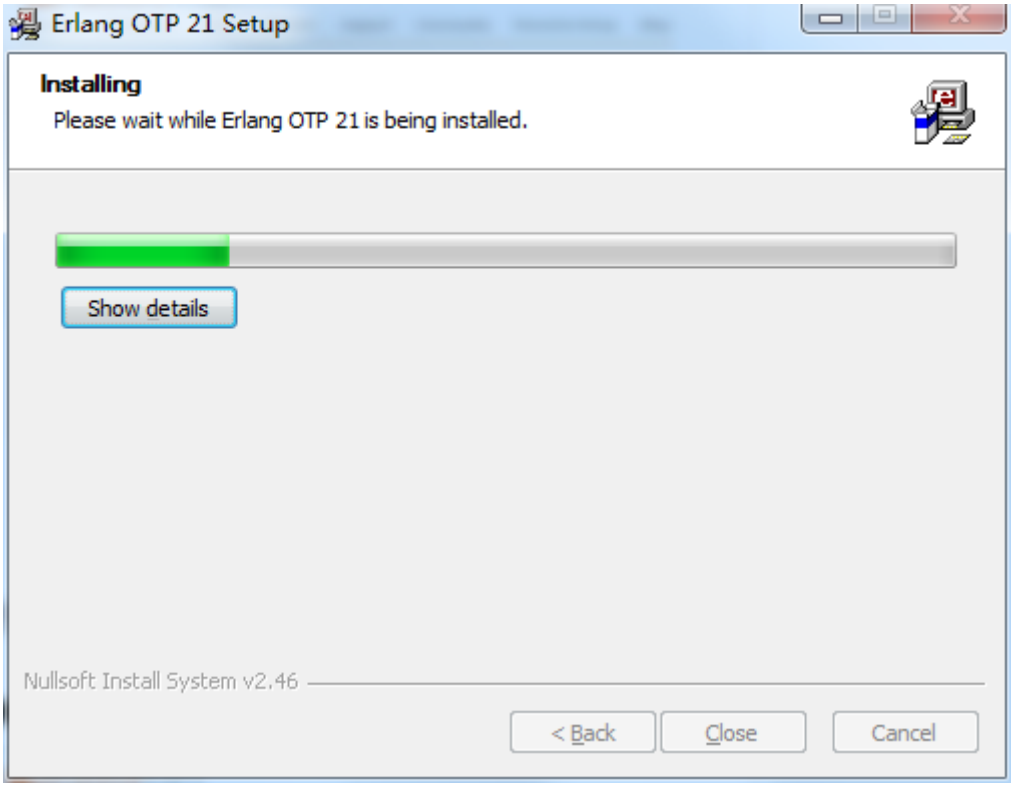
为了方便学习，也为了和官方文档统一一下，所以在这里选择安装RabbitMQ。直接点击http://www.rabbitmq.com/download.html下载RabbitMQ的安装包就好了。如下图（我选择了Windows安装包）：



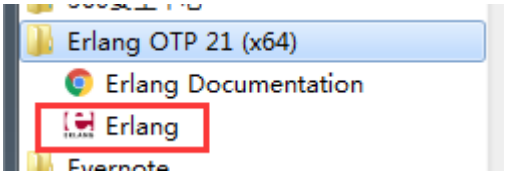
发现在安装之前还需要安装Erlang，按照提示下载安装就可以了。

Erlang下载网站http://www.erlang.org/downloads

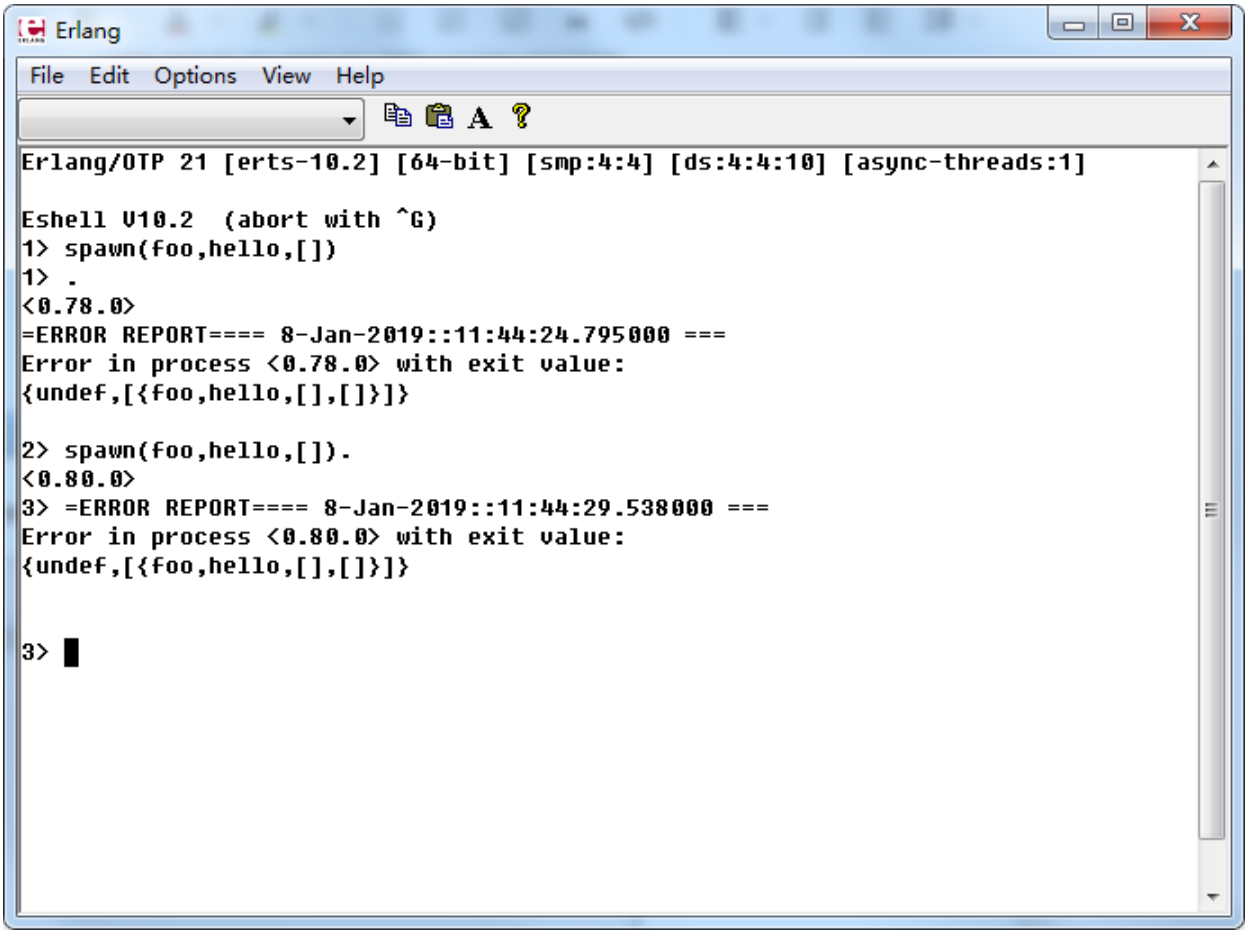

安全 | https://www.erlang.org/news/125
小白一键重装 百度 淘宝 京东 天猫 系统之家 电影大全 intellij git ancun 前端 springboot pro
Highlights
SSH:
Public key methods: ssh-ed25519 and ssh-ed448 added. Requires OpenSSL 1.1.1 or late
SSL:
ssl now uses active n internally to boost performance. Old active once behaviour can be re
ERTS, Kernel:
New counters and atomics modules supplies access to highly efficient operations on muta
New module persistent_term!. Lookups are in constant time! No copying the terms!
New pollset made to handle sockets that use {active, true} or {active, N}. Polled by a norm
No more ONESHOT mechanism overhead on fds! Only on Linux and BSD
For a full list of details see:
http://erlang.org/download/otp_src_21.2.readme
Pre built versions for Windows can be fetched here:
http://erlang.org/download/otp_win32_21.2.exe
http://erlang.org/download/otp_win64_21.2.exe
Online documentation can be browsed here:
http://erlang.org/documentation/doc-10.2/doc
The Erlang/OTP source can also be found at GitHub on the official Erlang repository, Here: OTP
Please report any new issues via Erlang/OTPs public issue tracker
https://bugs.erlang.org
We want to thank all of those who sent us patches, suggestions and bug reports!
Thank you!
The Erlang/OTP Team at Ericsson
Tags: [release]



安装完Erlang之后，打开开始菜单

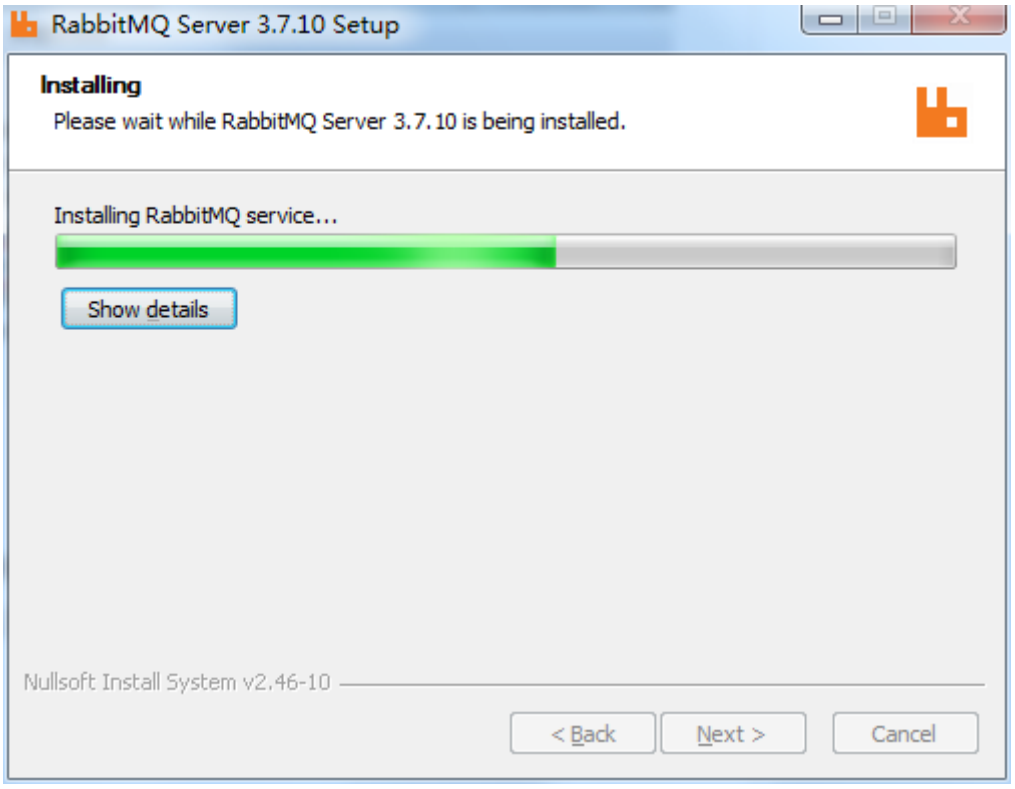


运行一下



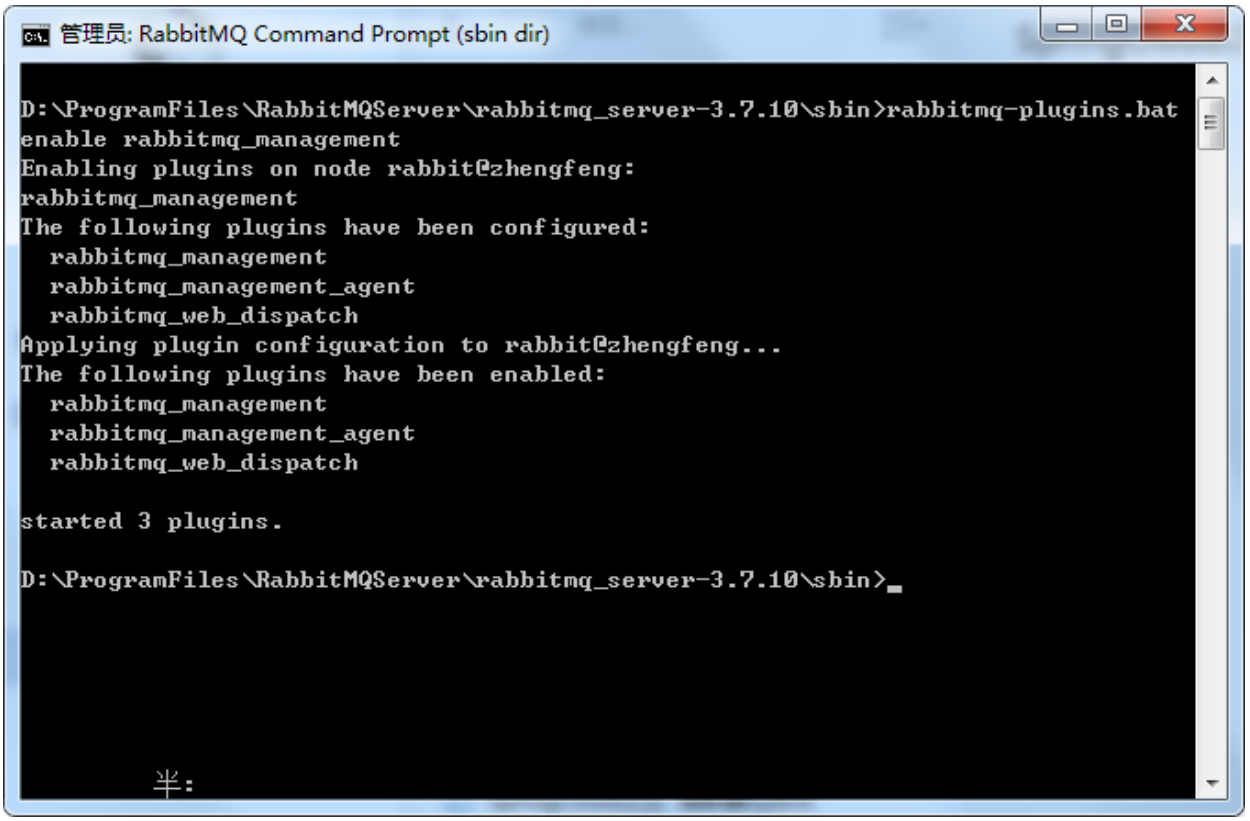
即是安装成功了！

接着安装RabbitMQ



输入命令，进行激活RabbitMQ：

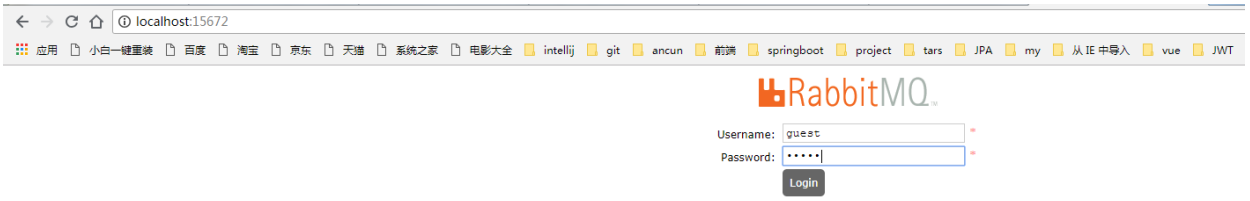
```
1 rabbitmq-plugins.bat enable rabbitmq_management
```



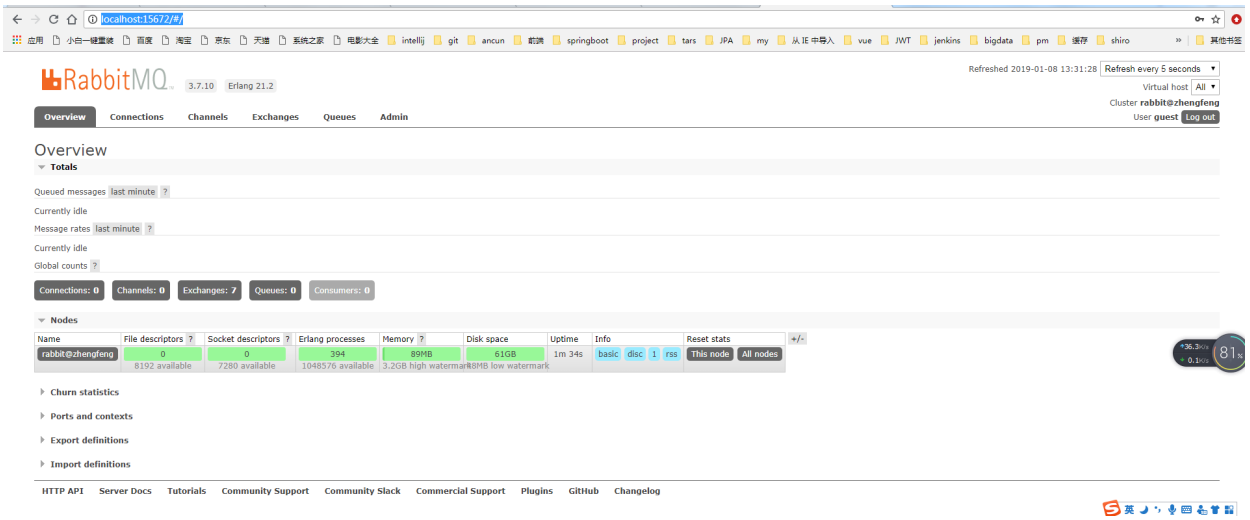
然后进行重启：先停止，再启动，也可以：

```
Net stop rabbitmq && net start rabbitmq
```

访问：http://localhost:15672，guest，guest（默认用户名，密码）



如图，RabbitMQ的首页：



至此Windows下RabbitMQ就已经安装成功啦

对于Erlang语言感兴趣的，可以自行搜索引擎去搜索学习，这里不多做扩展！

2.2、改造config-client

2.2.1、添加依赖

在pom.xml文件中加入spring-cloud-starter-bus-amqp依赖，如下：

```
<dependency>
 <groupId>org.springframework.cloud</groupId>
 <artifactId>spring-cloud-starter-bus-amqp</artifactId>
</dependency>
<dependency>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

2.2.2、修改配置文件

在配置文件application.yml 中加上RabbitMq的配置，包括RabbitMq的地址、端口，用户名、密码。并需要加上spring.cloud.bus的三个配置，具体如下：

```

spring:
 rabbitmq:
 host: localhost
 port: 5672
 username: guest
 password: guest
 cloud:
 bus:
  enabled: true
  trace:
  enabled: true

management:
 endpoints:
 web:
 exposure:
 include: bus-refresh
```

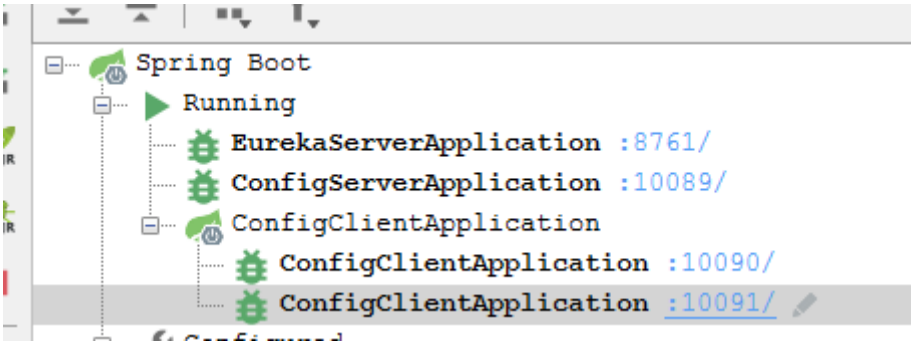
在ConfigClientApplication启动类添加上相关注解，代码如下：

```
package xyz.jiangnanke.configclient;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.cloud.context.config.annotation.RefreshScope;
import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@EnableEurekaClient
@EnableDiscoveryClient
@RefreshScope
@RestController
public class ConfigClientApplication {

 public static void main(String[] args) {
 SpringApplication.run(ConfigClientApplication.class, args);
 }

}


```

@RefreshScope(`org.springframework.cloud.context.scope.refresh`)是spring cloud提供的一种特殊的scope实现，用来实现配置、实例热加载。也可以在LoginController.java也加上这个注解。

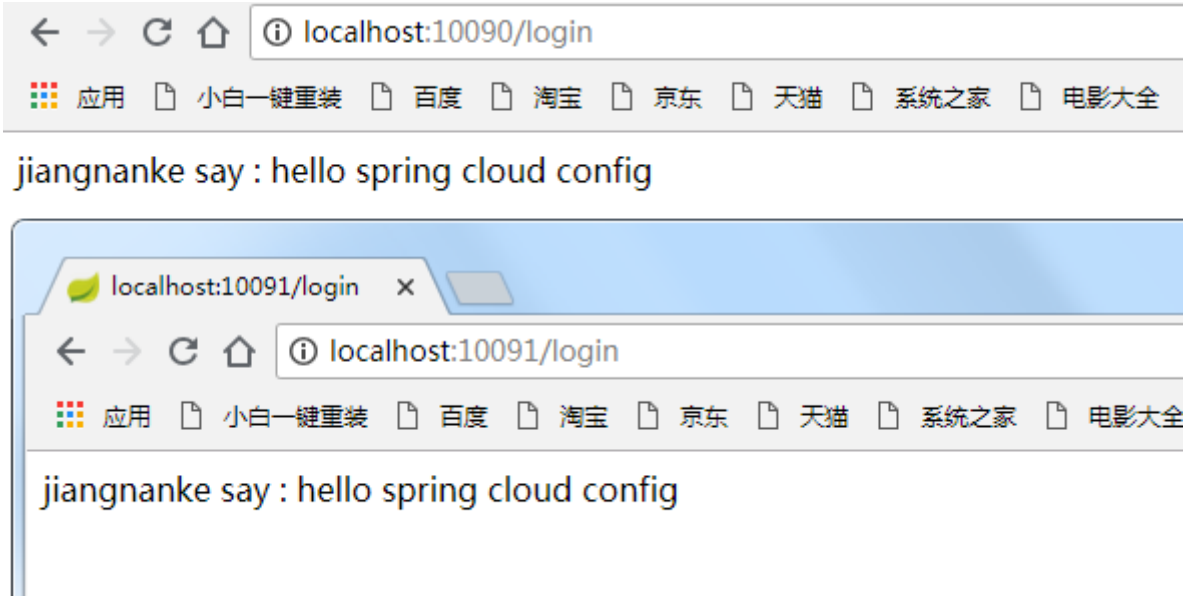
2.3、启动

依次启动eureka-server、confg-cserver，以及启动两个config-client，端口为：10090、10091。



访问http://localhost:10090/login 或者 http://localhost:10091/login

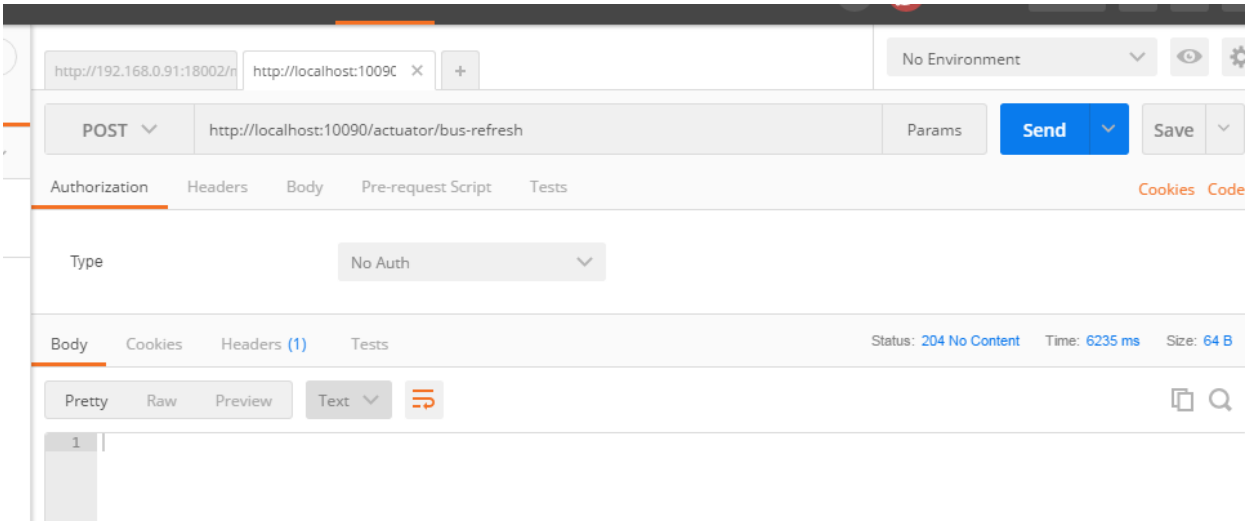
浏览器显示结果如图：



这时我们去代码仓库将 name 的值改为 “jiannanke--test” ，即改变配置文件 name 的值。

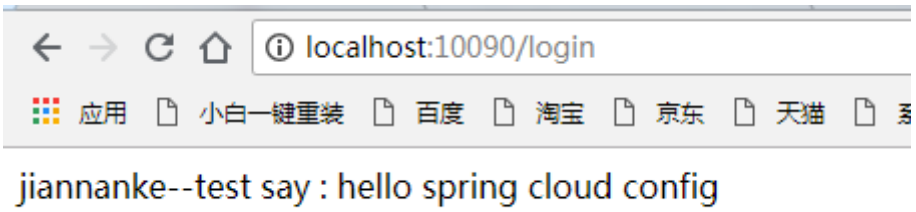
如果是传统的做法，需要重启服务，才能达到配置文件的更新。但是此时，我们只需要发送post请求：http://localhost:10090/actuator/bus-refresh，发现没

有什么返回，不过是可以有刷新的。这里使用了postman工具进行post请求发送，使用浏览器默认的是get方式，会请求无响应，导致刷新不了配置修改。



则会发现config-client会重新读取配置文件。重新访问链接，效果如图：

（刷新了之后，最好稍等一会再访问，不然很可能还是原来的值）

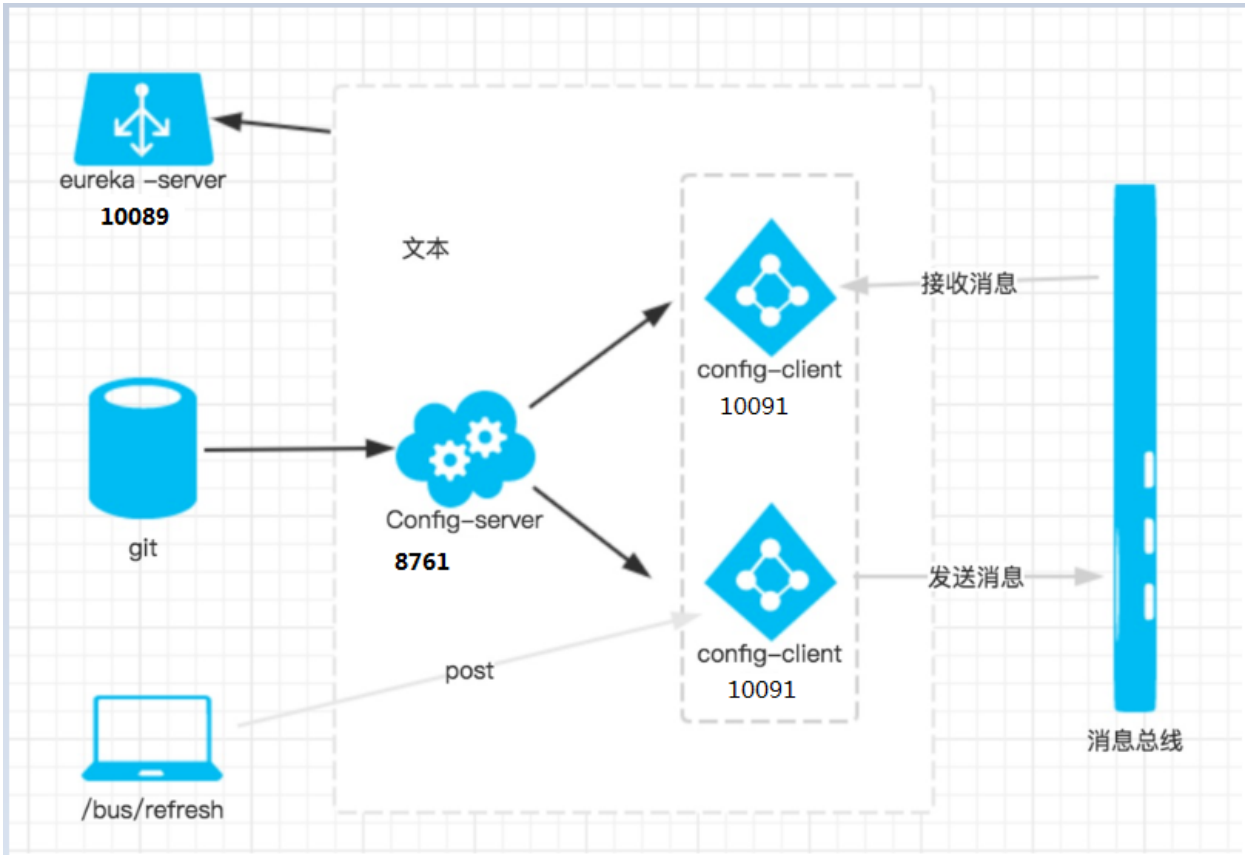


jiannanke--test say : hello spring cloud config

另外，/actuator/bus-refresh接口可以指定服务，即使用"destination"参数，比如 “/actuator/bus-refresh?destination=customers:**” 即刷新服务名为customers的所有服务。

附录：分析

当git文件更改的时候，通过pc端用post 向端口为10091的config-client发送请求/bus/refresh / ；此时10091端口会发送一个消息，由消息总线向其他服务传递，从而使整个微服务集群都达到更新配置文件。



参考资料：

https://springcloud.cc/spring-cloud-bus.html
https://springcloud.cc/
http://cloud.spring.io/spring-cloud-static/Finchley.RELEASE/single/spring-cloud.html
http://activemq.apache.org/
http://kafka.apache.org/

http://rocketmq.apache.org
http://www.rabbitmq.com/
https://www.jianshu.com/p/2838890f3284
https://blog.csdn.net/forezp/article/details/81041062
https://blog.csdn.net/BuquTianya/article/details/78698755
https://baijiahao.baidu.com/s?id=1605656085633071281&wfr=spider&for=pc
https://www.sohu.com/a/244980510_473282