和Spring Cloud Config一样，我们接下来要聊的Spring Cloud Bus也是微服务架构系统中的必备组件。Spring Cloud Bus可以将分布式系统的节点与轻量级消息代理链接，然后可以实现广播状态更改（例如配置更改）或广播其他管理指令。Spring Cloud Bus就像一个分布式执行器，用于扩展的Spring Boot应用程序，但也可以用作应用程序之间的通信通道。那么这里就涉及到了消息代理，目前流行的消息代理中间件有不少，Spring Cloud Bus支持RabbitMQ和Kafka，本文我们主要来看看RabbitMQ的基本使用。

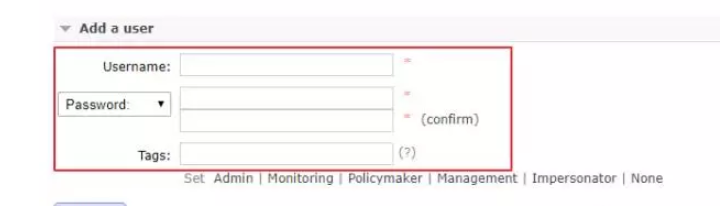
**RabbitMQ安装配置**

RabbitMQ是用Erlang语言编写的，因此安装RabbitMQ之前我们要先安装Erlang环境，首先去http://www.erlang.org/downloads地址下载erlang，下载到的是一个exe文件，直接双击安装即可，然后去http://www.rabbitmq.com/download.html地址下载RabbitMQ，下载成功之后，也是一个exe，双击安装即可(我在网上看到有人说安装目录不能有空格，我自己的软件安装目录都是没有空格的，所以没去验证这句话真假，小伙伴们注意下别掉坑里了)。RabbitMQ安装成功之后，默认会创建系统服务，将和Windows系统一起启动。

安装成功之后，我们可以使用web来管理我们的RabbitMQ，管理之前需要我们先开启web管理功能，开启方式：进入到安装目录的sbin目录下，然后执行 .\rabbitmq-plugins enable rabbitmq\_management命令

执行成功之后，打开浏览器，输入http://localhost:15672进入到web管理页面，需要先登录，默认用户名密码都是guest

登录成功之后，我们可以先尝试创建一个用户



输入用户名密码就可以创建了，tags表示用户标签，相当于角色，可选值有management、none、policymaker、monitoring和administrator，不同取值所对应的权限区别如下：

### none

1.不能访问 management plugin

### management

用户可以通过AMQP做的任何事外加：

1.列出自己可以通过AMQP登入的virtual hosts  
2.查看自己的virtual hosts中的queues, exchanges 和 bindings  
3.查看和关闭自己的channels 和 connections  
4.查看有关自己的virtual hosts的“全局”的统计信息，包含其他用户在这些virtual hosts中的活动

### policymaker

management可以做的任何事外加：

1.查看、创建和删除自己的virtual hosts所属的policies和parameters

### monitoring

management可以做的任何事外加：

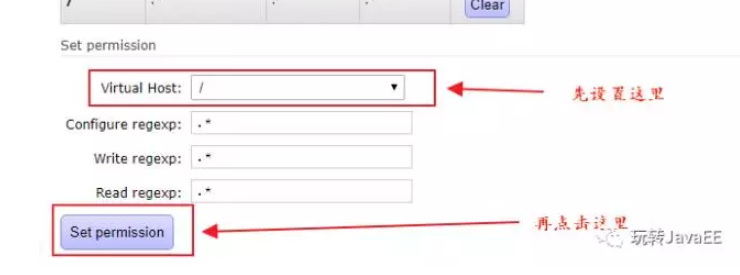
1.列出所有virtual hosts，包括他们不能登录的virtual hosts  
2.查看其他用户的connections和channels  
3.查看节点级别的数据如clustering和memory使用情况  
4.查看真正的关于所有virtual hosts的全局的统计信息

### administrator

policymaker和monitoring可以做的任何事外加:

1.创建和删除virtual hosts  
2.查看、创建和删除users  
3.查看创建和删除permissions  
4.关闭其他用户的connections

创建一个sang用户一会使用（当然不创建也可以，不创建就直接使用默认的guest用户），sang用户创建好之后，点击用户名，给用户设置virtual hosts，否则一会使用这个用户的时候会报错。



## 简单案例

创建一个普通的Spring Boot工程，然后添加如下依赖：

<dependency>

   <groupId>org.springframework.boot</groupId>

   <artifactId>spring-boot-starter-amqp</artifactId>

</dependency>

在application.properties中配置RabbitMQ的连接信息，如下：

spring.application.name=rabbitmq-hello

spring.rabbitmq.host=localhost

spring.rabbitmq.port=5672

spring.rabbitmq.username=sang

spring.rabbitmq.password=123456

server.port=2009

这里我们分别配置了RabbitMQ的地址为localhost，端口为5672（注意这里没写错，web管理端端口是15672），用户名和密码则是我们刚刚创建出来的（也可以使用默认的guest）。

发送消息我们有一个现成的封装好的对象AmqpTemplate，通过AmqpTemplate我们可以直接向某一个消息队列发送消息，消息生产者的定义方式如下：

@Component

public class Sender {

   @Autowired

   private AmqpTemplate rabbitTemplate;

   public void send() {

       String msg = "hello rabbitmq:"+new Date();

       System.out.println("Sender:"+msg);

       this.rabbitTemplate.convertAndSend("hello", msg);

   }

}

注入AmqpTemplate，然后利用AmqpTemplate向一个名为hello的消息队列中发送消息。

@Component

@RabbitListener(queues = "hello")

public class Receiver {

   @RabbitHandler

   public void process(String msg) {

       System.out.println("Receiver:"+msg);

   }

}

@RabbitListener(queues = “hello”)注解表示该消息消费者监听hello这个消息队列，@RabbitHandler注解则表示process方法是用来处理接收到的消息的，我们这里收到消息后直接打印即可。

@Configuration

public class RabbitConfig {

   @Bean

   public Queue helloQueue() {

       return new Queue("hello");

   }

}创建一个名为hello的消息队列。

创建单元测试类，用来发送消息，如下：

@RunWith(SpringJUnit4ClassRunner.class)

@SpringBootTest(classes = RabbitmqHelloApplication.class)

public class RabbitmqHelloApplicationTests {

   @Autowired

   private Sender sender;

   @Test

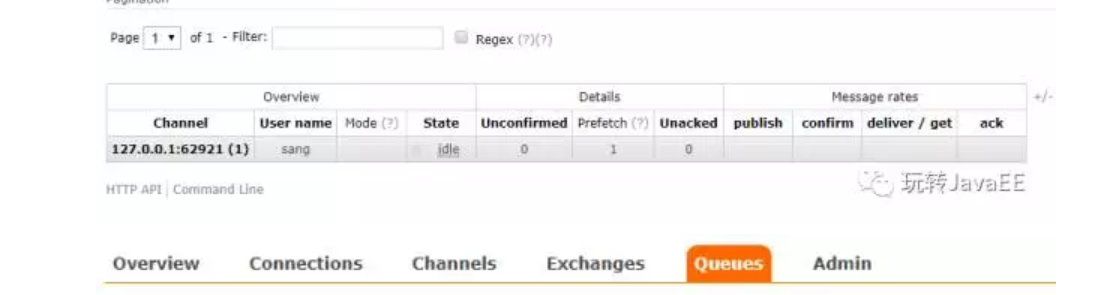
   public void contextLoads() {

       sender.send();

   }

}

上面所有的工作做完后，我们就可以启动我们的Spring Boot工程了



当我的微服务需要注册到eureka注册中心时，我需要给它添加spring-cloud-starter-eureka依赖，而当我的微服务需要使用Spring Cloud Bus时，我就给它添加spring-cloud-starter-bus-amqp依赖，我们在之前的Config-Server和Config-Client上都添加如下依赖:

<dependency>

   <groupId>org.springframework.cloud</groupId>

   <artifactId>spring-cloud-starter-bus-amqp</artifactId>

</dependency>

OK，添加好依赖之后，由于我们的Config-Server一会要连接到RabbitMQ服务器上，所以在Config-Server的application.properties中做如下配置：

spring.rabbitmq.host=localhost

spring.rabbitmq.port=5672

spring.rabbitmq.username=sang

spring.rabbitmq.password=123456

配置好之后，分别启动我们的Eureka注册中心、Config-Server和Config-Client（Config-Client启动两个实例，方便我们测试），我们在Config-Server和Config-Client的启动日志中都可以看到如下内容：

Mapped “{[/bus/refresh],methods=[POST]}” onto public void org.springframework.cloud.bus.endpoint.RefreshBusEndpo  
int.refresh(java.lang.String)

这个接口就是在我们添加了spring-cloud-starter-bus-amqp依赖之后具备的，Config-Server和Config-Client都具备这个接口。

此时我们访问http://localhost:2008/sang

然后我们通过git工具，修改配置仓库中文件的内容，修改之后提交，提交之后如果我们直接访问http://localhost:2008/sang，内容还是不变，此时我们可以利用POSTMAN或者RestClient发送一个POST请求到http://localhost:2007/bus/refresh地址，这个时候Config-Server收到通知之后，就会广播配置文件改变的消息，此时再访问http://localhost:2008/sang和http://localhost:2009/sang，就能看到改变后的配置文件了

此时我们发现两个实例http://localhost:2008/sang和http://localhost:2009/sang都改变了，如果我们我们只想通知其中一个微服务更新配置，那么在刷新请求的时候携带上一个参数就行了，比如当我的配置文件更新后，我只想端口号为2008的实例更新配置，端口号为2009的实例不更新配置，那么在配置文件修改之后，我只需发送如下请求即可：```http://localhost:2007/bus/refresh?destination=configClient:2008```，destination后面指定要更新的实例的instance-id，小伙伴们注意我这里的config-client的如下两条配置：

spring.application.name=configClient

eureka.instance.instance-id=${spring.application.name}:${server.port}

每一个实例的instance-id是{server.port}共同组成。