一、Kafka学习

1. 功能

发布与订阅消息、记录消息流、处理消息流。适用于数据吞吐量大，但对于消息重复、丢失不敏感情景。如果对消息正确性要求高但对数据吞吐量要求不严格可以采用rabbitmq。

1. 组成

生产者（producer）、topic（主题）、consumer（消费者）

消息存储按照分区存储，不同分区的消息都是按照时间顺序存储的。不同分区的数据都包含一个offset属性，用以区分该消息的在所有的消息的时间顺序。Kafka可以先把消息保存下来，等有消费者消费时从文件中取出消息，发给消费者。消息的发布有两种模式，队列模式以及发布-订阅模式。队列模式中每个消息仅可被一个消费者消费；而发布-订阅则把消息广播给所有的消费者。

1. 运行

新版的kafka内置了zookeeper，首先启动zookeeper，bin/zookeeper-server-start.sh config/zookeeper.properties；

然后启动kafka，bin/kafka-server-start.sh config/server.properties

1. 创建topic

bin/kafka-topics.sh --create --topic topicName --bootstrap-server localhost:9092

1. 接收指定topic消息

仅需给方法添加注解即可，@KafkaListener(topics = {"djTest"})，在方法中解析topic中的数据。

1. 向指定的topic发送消息

调用kafkaTemplate.send(TOPIC\_NAME, "key", msg);即可，其中msg为消息内容。TOPIC\_NAME为topic值

1. 操作kafka

通过添加kafka-clients依赖，创建AdminClient对象，即可通过adminClient对象操作kafka。创建topic，查看topic以及删除topic

二、arthas学习

Arthas为阿里开源的java诊断程序，这里仅利用其查看具体方法执行的耗时分析。具体使用为运行java -jar arthas-boot.jar，然后选中带分析的java进程。要分析具体方法耗时，采用trace demo.mathGame run,其中demo.mathGame为具体的类名具体路径，run为该类下的方法。

三、dubbo学习

首先需要安装zookeeper，启动运行zkServer.cmd即可。创建api模块，定义接口，api模块不需要引入其他依赖。

创建provider模块，引入api模块以及dubbo以及zookeeper依赖，实现api接口。在yaml文件中配置dubbo配置，包括应用名称、服务类包扫描、对外协议端口以及注册中心等。在实现类上添加dubbo的Service注解。可以创建多个provider，进行负载均衡。

创建consumer模块，同provider类似引入api模块以及dubbo依赖，并在yaml文件中配置dubbo配置，主要为应用名称、注册中心以及标志为消费者。创建controller类，其中需要引入接口添加Reference，可以在里面添加负载均衡配置。

Dubbo在spring启动的时候增加了监听ApplicationContextInitializer事件，当服务启动的时候会监听到该事件然后调用DubboBootstrap下的start方法。

四、SPI（service provider interface）

Api接口与实现绑定在一起，spi接口与实现是分离的，可以通过配置文件具体加载哪个实现类，方便扩展。Api为服务实现方与接口定义在一起，调用方引入实现方api即可调用实现方的具体实现；spi接口定义存在调用方，实现方根据接口定义给出不同的实现。Spi与api都属于接口。

五、netty学习

Netty提供异步的、[事件驱动](https://baike.baidu.com/item/%E4%BA%8B%E4%BB%B6%E9%A9%B1%E5%8A%A8/9597519" \t "_blank)的网络应用程序框架和工具，用以快速开发高性能、高可靠性的[网络服务器](https://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E6%9C%8D%E5%8A%A1%E5%99%A8/99096)和客户端程序。

Netty通过添加handler进行消息处理，handler分为InboundHandler以及outboundHandler，多个handler组成一个链表，当接收消息时按照代码编写的添加inboundhandler顺序依次执行；在发送消息时按照outboundhandler顺序逆序执行。一个handler处理完成之后需要交给下一个handler去处理需要调用ctx. fireChannelRead方法。

六、java线程池

Java的web编程通过socket连接一个客户端，服务器与客户端为1对多的关系。当新的客户端连接时javaBIO会创建一个线程处理当前的连接。因此有多少连接即会创建多少线程，如果连接数不断增加，则会出现OOM问题。可以创建线程池来限制线程的创建，通过规定线程池大小，可以允许当前活跃线程个数。线程池当中有常驻线程，临时线程以及队列。如果当前正在执行的任务个数超过常驻线程数，则将其放在队列中，如果队列也满了则创建新的线程，线程总数不可超过设置大小。如果任务个数超过线程总数大小，则拒绝请求。在队列中的任务一直在等待。只有线程中处理完其他任务之后才去队列中领取任务。

七、RocketMq学习

1、从源码编译安装包

在github上下载下源码，然后执行mvn -Prelease-all -DskipTests clean install -U命令，即可在rocketmq\distribution\target生成安装包

2、创建topic

执行命令：mqadmin updateTopic -n localhost:9876 -b localhost:10911 -t RealBizTopic

其中-n为nameserver -b broker -t为topic，即需要指定nameserver在具体的broker上创建topic

3、nameserver

Rocketmq中nameserver作为注册中心，管理所有的broker，当发布者需要发布topic时需要从nameserver获取该topic所在的broker。消费者也是一样当消费某个topic是需要从nameserver获取broker地址。Nameserver中的namesrvController为所有关于nameserver操作方法。其中包括routeInfomanager，即为存储topic与broker对应关系、broker集群信息、存活的broker列表等信息。namesrvController还包括一些配置信息、定时任务以及remoteServer。其中定时任务主要包括配置信息的更新以及检查不存活的broker；remoteServer为具体的netty创建服务处理连接请求。

八、springcloud学习

基本定义：Spring Cloud基于Spring Boot框架开发应用，为微服务开发中的架构问题提供了一整套的解决方案:如服务注册与发现、服务消费、服务容错、API网关、分布式调用追踪和分布式配置管理等。

1、服务注册到注册中心

为了让其他服务可以找到该服务并且调用，需要将服务注册到注册中心中。这里以consul为例，在创建的springboot工程中pom.xml添加spring-cloud-starter-consul-discovery依赖以及spring-boot-starter-actuator检查服务状态的依赖。然后在yml文件中配置consul地址以及服务名称等。

spring:

application:

name: student-provider

cloud:

consul:

host: localhost

port: 8500

discovery:

service-name: student-provider #注册到 Consul 的服务名称，客户端根据这个名称来进行服务调用。

2、使用openfegin进行服务调用

Spring Cloud OpenFeign 对 Feign 进行了二次封装，使得在 Spring Cloud 中使用 Feign 的时候，可以做到使用 HTTP 请求访问远程服务，就像调用本地方法一样的，开发者完全感知不到这是在调用远程访问，更感知不到在访问 HTTP 请求。Spring Cloud OpenFeign 增强了 Feign 的功能，使 Feign 有限支持 Spring MVC 的注解，如 @RequestMapping 等。OpenFeign 的 @FeignClient 注解可以解析 Spring MVC 的 @RequestMapping 注解下的接口，并通过动态代理的方式产生实现类，在实现类中做负载均衡并调用其他服务，默认集成了 Ribbon 与 Hystrix。

Openfeign通过http形式进行rpc，首先创建接口模块，添加spring-cloud-starter-openfeign依赖。接口模块定义了接口，并在接口上添加@FeignClient(“xxx”)注解，其中xxx为服务的名称。在具体方法中添加@RequestMapping注解，指定目标服务的url地址。即通过服务名称加服务url地址进行远程服务调用。

然后创建服务提供者模块，引入接口模块。创建服务类实现接口，在该类中实现具体的业务。然后需要将服务提供模块注册到注册中心以便消费者发现，这是使用的consul注册中心，因此需要添加spring-cloud-starter-consul-discovery依赖。还可添加spring-boot-starter-actuator进行服务检查。

最后创建服务消费者，引入接口模块。在具体类中注入接口，调用接口的方法，虽然调用了接口方法但是实际上调用了服务提供者模块实现的方法。服务消费者需要接入注册中心，这样才可以从注册中心根据服务名称获取服务提供者地址。添加spring-cloud-starter-consul-discovery依赖即可。还需要在启动类中添加@EnableFeignClients注解。

九、Zookeeper学习

1、Zookeeper目的

Zookeeper用来协调管理分布式的服务，具有简单、可靠、顺序以及快速优点。Zookeeper提供了类似文件系统的znode，不同的是znode可以携带少量数据，用户可以创建、修改、删除znode等操作。当znode被修改之后，Zookeeper可以通知所有监听该znode的用户。Zookeeper用途：数据发布与订阅、负载均衡、命名服务，配置管理，集群管理，分布式锁，队列管理。