1. 原生java中的缓存（JSR-107）

在原始的j2ee中规定了一些缓存的接口，用于同意缓存

CacingeProvider :定义了创建、配置、获取、管理和控制多个CacheManager。一个应用可以在运行期访问多个CachingProvider。

CacheManager：缓存管理者，定义了创建、配置、获取、管理和控制多个唯一命名的Cache，这些Cache存在于CacheManager的上下文中。一个CacheManager仅被一个CachingProvider所拥有

Cache：一个缓存，比如员工部门缓存是一个类似Map的数据结构并临时存储以Key为索引的值。一个Cache仅被一个CacheManager所拥有。

Entry：缓存中的一条数据，键值对

Expiry：缓存的有效时间，每条数据都有一个有效时间，最长见就是短信有效时间

CachingProvider

CacheManager

CacheManager

Cache

Cache

Cache

Entry<K,V>

Entry<K,V>

Entry<K,V>

Entry<K,V>

Expiry

Expiry

Expiry

Expiry

Application

CachingProvider

1. spring中的缓存
2. spring中编写了自己的缓存接口，这些接口可以支持JSR-107也可以有自己的扩展，并且spring只保存了cache 和cachemanager两个接口
3. Cache接口下Spring提供了各种xxxCache的实现；如RedisCache，EhCacheCache , ConcurrentMapCache等
4. 在不导入别的缓存容器中间件的情况之下，默认使用的是aimpleCacheConfiguration的CurrentmapManager，底层为CurrentMap。

3、缓存的一些注解以及运行流程以及注解

1. 1、CacheManager接口：主要用来管理一个或者多个cache

2、cache接口：缓存接口，定义缓存操作，并且对于市面上流行的缓存软件，有相应的接口与之对应，如RedisCache、EhCacheCache、ConcurrentMapCache等

3、@Cacheable：主要标注在一个查询的方法上，表示这个方法的结果是可缓存的，对于每次查询，Spring会检查检查**指定参数的指定的目标方法**是否已经被调用过；如果有就直接从缓存中获取方法调用后的结果，如果没有就调用方法并缓存结果后返回给用户，并将结果存入缓存。下次调用直接从缓存中获取。（已经调用过的指定参数的方法不一定再调）

4、@CacheEvict：删除缓存，用在delete方法上，他会先从数据库中删除数据，然后将缓存中该数据也删除

5、@CachePut：更新缓存：用在update方法上，这个方法一定会被调用，并且将结果放入缓存或者更新缓存

6、@EnableCaching：开启基于注解的缓存

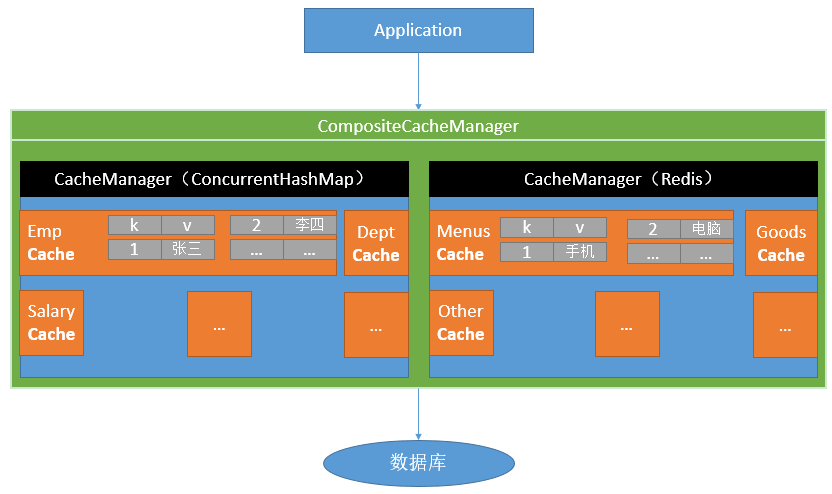
7、keyGenerator：缓存数据时key生成策略，也就是存入一条数据的时候，他的主键存储方式

8、serialize：缓存数据时value序列化策略（如以json还是键值对）

4、主要的一些参数

|  |  |  |
| --- | --- | --- |
| **@Cacheable/@CachePut/@CacheEvict 主要的参数** | | |
| value | 缓存的名称，在 spring 配置文件中定义，必须指定至少一个 | 例如： @Cacheable(value=”mycache”) 或者  @Cacheable(value={”cache1”,”cache2”} |
| key | 缓存的 key，可以为空，如果指定要按照 SpEL 表达式编写，如果不指定，则缺省按照方法的所有参数进行组合 | 例如： @Cacheable(value=”testcache”,  key=”#userName”) |
| condition | 缓存的条件，可以为空，使用 SpEL 编写，返回 true 或者 false，只有为 true 才进行缓存/清除缓存，在调用方法之前之后都能判断 | 例如： @Cacheable(value=”testcache”,  condition=”#userName.length()>2”) |
| allEntries  (**@CacheEvict**) | 是否清空所有缓存内容，缺省为 false，如果指定为 true，则方法调用后将立即清空所有缓存 | 例如： @CachEvict(value=”testcache”,  allEntries=true) |
| beforeInvocation  **(@CacheEvict)** | 是否在方法执行前就清空，缺省为 false，如果指定为 true，则在方法还没有执行的时候就清空缓存，缺省情况下，如果方法执行抛出异常，则不会清空缓存 | 例如：  @CachEvict(value=”testcache”，  beforeInvocation=true) |
| unless  **(@CachePut)**  **(@Cacheable)** | 用于否决缓存的，不像condition，该表达式只在方法执行之后判断，此时可以拿到返回值result进行判断。条件为true不会缓存，fasle才缓存 | 例如： @Cacheable(value=”testcache”,unless=”#result == null”) |

5、基本构架图



6、实操与原理

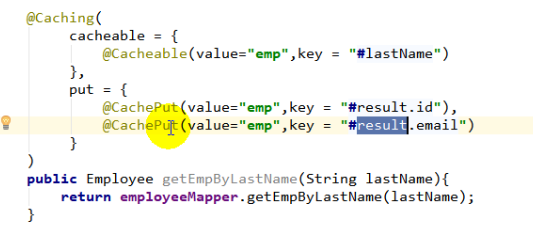
1*@Cacheable*

*/\*  
\* @Cacheable:将标注的方法进行缓存，以后再要相同的数据，直接从缓存里面获取，不再查询数据库  
\*cacheNames标注缓存名，表示他存在这个缓存里，然后每一个缓存都由@CachManager来管理  
\* key：还可以指定key，就是这个缓存存储时key是什么，不标注的话默认就是参数、也可以用spel表达式指定  
\* 如#id  
\* keyGenerator:指定主键的生成策略，与key二选一  
\* cacheManager：指定缓存管理器  
\* condition：指定在什么条件下才缓存，如condition="#id >0"  
\*unless：指定在什么条件下不缓存  
\* sync：指定同步还是异步  
\*  
\*  
\* 原理  
\* 1、自动配制类CacheAutoConfiguration  
\* 2、他会导入很多的cache自动配置，如EhCache，redis、Jcache（jsr-107），SimpleCache等  
\* 3、里面有很多condition来判断，可以在debug模式下看哪个生效  
\* 4、在我们不加任何组件的情况下，只有SimpleCacheConfiguration有效，他的作用是给容器注册了  
\* 一个cacheManager-->CurrentMapCacheManager  
\* 5|这个CurrentMapCacheManager会创建一个CurrentMapCache（交store）用于保存数据，  
\*  
\*  
\* 运行流程，以@Cacheable为例  
\* 1方法运行之前先在容器里找有没有这个cache（如这找有没有名字为emp的cache），如果没有则创建一个，  
\* （相当于创建CurrentMap容器）  
\* 2到cache里面以主键（SimpkeyGenerator）找有没有相应的内容，如果没有调用目标方法，有则直接返回  
\* 主键生成策略SimpkeyGenerator  
\* 如果方法没有参数：直接生成一个默认的SimpkeyGenerator对象  
\* 如果有一个参数：key=参数值  
\* 如果有多个参数：key=含有多个参数值键值对的SimpkeyGenerator对象  
\* 3、缓存没找到，调用目标方法  
\* 4、将结果放入currentmap  
\* 5、第二次调用以后就可以通过主键找到结果就不会去执行方法，可直接返回数据  
\* \*/*@Cacheable(cacheNames = **"emp"**)  
**public** Employee getempbyid(Integer id){  
 System.***out***.println(**"查询"**);  
 **return EmployeeMapper**.getempbyid(id);  
}  
  
2@CachePut  
  
*/\*  
\* @CachePut：即更新数据库，也更新缓存  
\* 实际执行过程  
\* 1、调用目标方法更新数据库  
\* 2、更新缓存  
\*  
\* 3、注意：对于同以个缓存中的数据，他的主键必须一样，这样他才能够操作相同的数据，才能够正确的更新  
\* 缓存中的数据，否则操作的是两条数据，因为是哪个嘛key为id，故这里指定他的key为key="#Employee.id"  
\*  
\* 4、这里的key="#Employee.id"，也可以用key="#resoult.id"代替，因为他是先执行方法更新数据，然后返回到缓存  
\* 所以resoult里存着从数据库里找到的数据，er@Cacheable不可以，因为他是先从缓存里面找，意味着在还  
\* 没查数据库的时候就要得到key  
\*  
\* \*/*@CachePut(cacheNames = **"emp"**,key=**"#Employee.id"**)  
**public** Employee updateemp(Employee Employee){  
 **EmployeeMapper**.updateemp(Employee);  
 System.***out***.println(**"员工更新"**);  
 **return** Employee;  
}  
  
3@CacheEvict  
*/\*  
\* @CacheEvict：删除缓存，默认先从数据库删除数据，再删缓存  
\* 1当标注allEntries=true的时候，会删除所有缓存  
\*  
\* \*/*@CacheEvict(value = **"emp"**)  
**public void** delete(Integer id){  
 **EmployeeMapper**.deleteemp(id);  
}

4复合标注@caching，他相当于多个标注的综合体，用于比较复杂的标注

这里的意思为

以名字查数据库然后存入缓存，再以id和email也存入缓存，相当于同一个缓存里面有三个key不同值相同的数据，当再以id或者email查的时候就不必查数据库



5 @CacheConfig（CacheNames=”emp”）

@CacheConfig可以放在一个类的头部，是缓存的公共部分，如把这句话防在类上面，相当于所有的方法的缓存均为emp

注意：缓存内部用的是Currentmap还作为cache，因此，对于多个方法都用同一个cache（value/cacheName相同）的时候，想要他们操作相同的数据，他们的key必须相同

redis的一些基本使用

引入reis后，redisAutoconfiguration起效，容器会自动注入redis的一些模板对象，redis内部也是用这两个对象来操作redis的，如果需要

也可以自动注入进来使用。

*/\*1对于stringRedisTemplate  
\* stringRedisTemplate.opsForValue(); //用于操作字符串  
 stringRedisTemplate.opsForList(); //用于操作list  
 stringRedisTemplate.opsForHash(); // 用于操作hash  
 stringRedisTemplate.opsForSet(); //用于操作set  
 stringRedisTemplate.opsForZSet(); //用于操作有序集合  
\*  
\*2对于RedisTemplate  
\*redisTemplate.opsForValue(); //用于操作字符串  
 redisTemplate.opsForList(); //用于操作list  
 redisTemplate.opsForHash(); // 用于操作hash  
 redisTemplate.opsForSet(); //用于操作set  
 redisTemplate.opsForZSet(); //用于操作有序集合  
\*  
\*二者quiet在于操作字符串还是对象  
\* \*/*

8、引入redis与使用redis

1. docker上安装redis
2. 引入redis的start
3. 为实体类加上序列化，这样才可将对象存入redis

**public class** Employee **implements** Serializable {

1. 原理

1引入redis后，容器注入的是redis的cachemanager，操作redis的时候使用的是RedisTemplate，redisTemplate的默认序列化是jdk序列化

2可以修改为json序列化，就是自己编写cachemanager，详情见12

1. 引入redis后，他的标注和simpleCache一样，所以上面那些标注仍然有效，和使用simpleCache一样使用redis就可以
2. 对于没有标注缓存的方又想使用缓存（就是随便将一个对象放入redis），可以这样

1注入Cachemanager

@Autowired  
CacheManager **CacheManager**;

2将对象放入对应的redis的cache

**public void** insertredis(Integer id){  
 Employee emps = **EmployeeMapper**.getempbyid(id);//得到对象  
 Cache emp = **CacheManager**.getCache(**"emp"**);//得到对应的cache  
 emp.put(id,emps);//将对象放入对应的cache  
 }

9、消息队列与功能

消息是为了异步通信能力和解耦能力而产生的，就是将前台客户发送的请求作为消息存入一个消息队列里面，然后由消息的消费者再慢慢处理，实现了很好的异步处理（一件事情可以不必全部做完才做另一件事），除此之外对于流量削峰，应用解耦也是很有好处

1异步处理





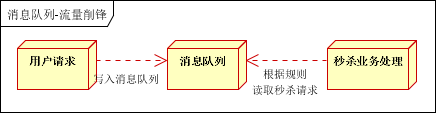
第一个图为同步处理，他们需要将所有的事情做完才能告诉用户注册完成，耗时150，而图二引入消息后，只需要将数据写入数据库和消息队列后就可以返回用户注册完成，至于后续的事情，系统会从消息队列里面找到数据再处理，这样极大的加强了体验

2应用解耦



对于两个分布式的系统，他们不必再紧密耦合，订单系统只需将订单加入消息队列，就算完成，而库存系统只需从消息队列里面取出数据然后处理即可

3流量削峰（秒杀）



比如十万个人抢一万个商品，只需将消息队列设置为一万，那么能进入队列则成功，其他不能进入队列的马上告诉秒杀失败，后续的任务可以等业务处理系统慢慢完成

消息的基本概念

1消息代理与目的地

消息代理解释市面上的一些软件如（AtiveMQ），目的地就是消息进入哪个队列，当消息发送者发送消息以后，将由消息代理接管，消息

理保证消息传递到指定目的地。

2点-点式

对于消息只有一个发送者和一个接收者，也就是说消息发送者将消息存入队列后，当有一个消息消费者得到消息后，消息动队列删除，其他消费者不能再得到消息

3发布订阅模式

发送者会将消息发给所有订阅他的人

4JMS  
基于JVM的消息队列服务api，跨平台能力较弱（ActiveMQ）

5AMQP(rabbitMQ)

网络线级协议,跨平台跨语言能利好，常用他的实现rabbitMQ，他只支持byte通信，支持五种消息交换

（1） direct exchange

（2）、fanout exchange

（3）、topic change

（4）、headers exchange

（5）、system exchange

6 Spring支持

spring-jms提供了对JMS的支持

spring-rabbit提供了对AMQP的支持

需要ConnectionFactory的实现来连接消息代理

提供JmsTemplate、RabbitTemplate来发送消息

@JmsListener（JMS）、@RabbitListener（AMQP）注解在方法上监听消息代理发布的消息

@EnableJms、@EnableRabbit开启支持

7 Springboot自动配置

JmsAutoConfiguration

RabbitAutoConfiguration

8 RabitMq

RabbitMq是基于AMQP实现的开源消息实现

9一些核心概念

Message

消息，消息是不具名的，它由消息头和消息体组成。消息体是不透明的，而消息头则由一系列的可选属性组成，这些属性包括routing-key（**路由键**）、priority（相对于其他消息的优先权）、delivery-mode（指出该消息可能需要持久性存储）等。

Publisher

消息的生产者，也是一个向交换器发布消息的客户端应用程序。

Exchange

交换器，用来接收生产者发送的消息并将这些消息路由给服务器中的队列。

Exchange有4种类型：direct(默认)，fanout, topic, 和headers，不同类型的Exchange转发消息的策略有所区别

ueue

消息队列，用来保存消息直到发送给消费者。它是消息的容器，也是消息的终点。一个消息可投入一个或多个队列。消息一直在队列里面，等待消费者连接到这个队列将其取走。

Binding

绑定，用于消息队列和交换器之间的关联。一个绑定就是基于路由键将交换器和消息队列连接起来的路由规则，所以可以将交换器理解成一个由绑定构成的路由表。

Exchange 和Queue的绑定可以是多对多的关系。

Connection

网络连接，比如一个TCP连接。

Channel

信道，多路复用连接中的一条独立的双向数据流通道。信道是建立在真实的TCP连接内的虚拟连接，AMQP 命令都是通过信道发出去的，不管是发布消息、订阅队列还是接收消息，这些动作都是通过信道完成。因为对于操作系统来说建立和销毁 TCP 都是非常昂贵的开销，所以引入了信道的概念，以复用一条 TCP 连接。

Consumer

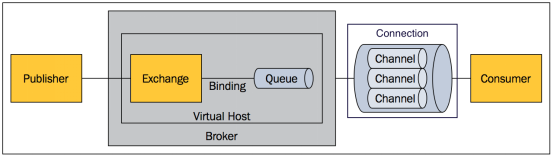
消息的消费者，表示一个从消息队列中取得消息的客户端应用程序。

Virtual Host

虚拟主机，表示一批交换器、消息队列和相关对象。虚拟主机是共享相同的身份认证和加密环境的独立服务器域。每个 vhost 本质上就是一个 mini 版的 RabbitMQ 服务器，拥有自己的队列、交换器、绑定和权限机制。vhost 是 AMQP 概念的基础，必须在连接时指定，RabbitMQ 默认的 vhost 是 / 。

Broker

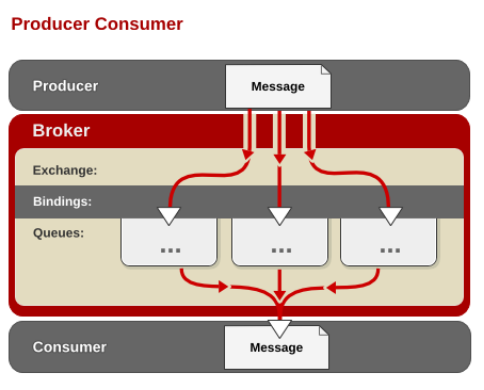
表示消息队列服务器实体，如rabbitmq（里面可包含多个vHost）



生产者与Broker只建立一条tcp连接，多路复用，当消息到达broker，brokker将数据给某一个exchange，然后exchange根据路由键和交换策略（Binding：绑定关系）将消息发送到合适的队列，消费者从队列里面拿数据也是通过一条tcp连接然后多路复用

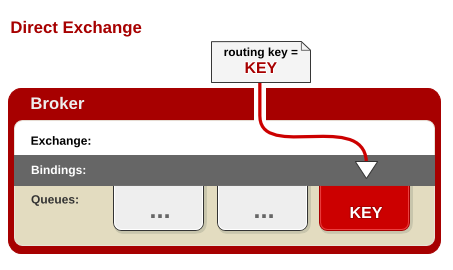
10 rabbitMq的运行机制

AMQP 中增加了 Exchange 和 Binding 的角色。生产者把消息发布到 Exchange 上，消息最终到达队列并被消费者接收，而 Binding 决定交换器的消息应该发送到那个队列。



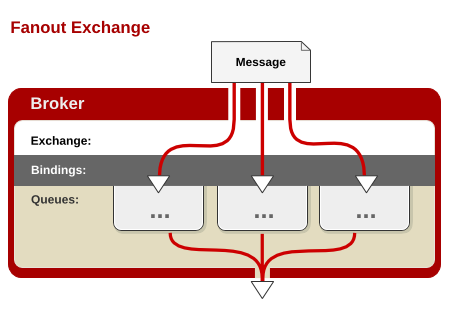
11 rabbitmq的Exchange

1direct模式

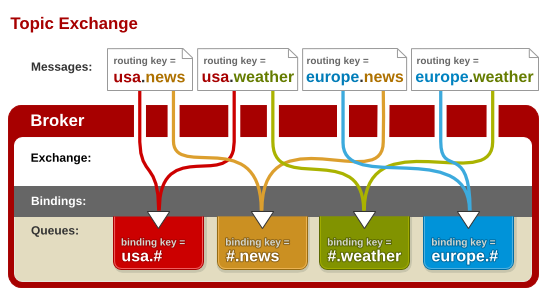


消息中的路由键（routing key）如果和 Binding 中的 binding key 一致， 交换器就将消息发到对应的队列中。路由键与队列名完全匹配，如果一个队列绑定到交换机要求路由键为“dog”，则只转发 routing key 标记为“dog”的消息，不会转发“dog.puppy”，也不会转发“dog.guard”等等。它是完全匹配、单播的模式。

2 fanout（广播式，所有订阅的都能收到）



3 topic（高级模式，模糊匹配模式）



topic 交换器通过模式匹配分配消息的路由键属性，将路由键和某个模式进行匹配，此时队列需要绑定到一个模式上。它将路由键和绑定键的字符串切分成单词，这些单词之间用点隔开。它同样也会识别两个通配符：符号“#”和符号“\*”。#匹配0个或多个单词，\*匹配一个单词。

12 docker上安装rabbitMq，记得安装带management（如rabbitmq：3-management）的这样会有自带的web管理界面

13启动rabitmq

他需要两个端口，分别给rabbitmq和他的监控页面

# docker run -d -p 5672:5672 -p 15672:15672 --name myrabbitmq df80af9ca0c9

启动后可访问他的管理界面：ip：15762

如（192.268.0.101:15672），

默认账号密码为

Username：guest

Password：guest

可以在管理界面里面添加删除交换器和队列，也可以做队列与交换器之间的绑定

springboot整合rabbitmq原理

1、自动配置类为RabbitAutoConfiguration，他为我们配置了

rabbitConnectionFactory：连接rabbitMq：而属性来自于配置文件，故要在配置文件里书写rabbitmq的地址

rabbitTemplate：用于操作rabit的添加消息和接收消息

amqpAdmin：用于对rabbit本身的添加删除交换器和队列，也可以做队列与交换器之间的绑定

2、默认读取的配置文件为RabbitProperties

16、使用rabbitmq

1、在创建项目的时候勾选rabbitmq

2、书写配置文件

**spring.rabbitmq.host**=**192.168.0.101  
spring.rabbitmq.username**=**guest  
spring.rabbitmq.password**=**guest**

1. 使用rabbitTemplaterabbitmq来发送和接收数据  
   @RunWith(SpringRunner.**class**)  
   @SpringBootTest  
   **public class** RabbitmqApplicationTests {  
    @Autowired  
    RabbitTemplate **rabbitTemplate**;  
    @Test  
    **public void** contextLoads() {  
    }  
    */\*  
    \* 单播*

**rabbitTemplate**.convertAndSend传三个参数，交换器名字、路由键、数据对象（Object型，可多样话） *\* \*/* @Test  
 **public void** addmessage2(){  
 **rabbitTemplate**.convertAndSend(**"exchange.direct"**,**"atguigu"**,**new** book(**"红落寞"**,**"曹雪芹"**));  
 }  
 */\*  
 \* 广播  
 \* \*/* @Test  
 **public void** addmessage3(){  
 **rabbitTemplate**.convertAndSend(**"exchange.fanout"**,**"atguigu"**,**new** book(**"红落寞"**,**"曹雪芹"**));  
 }

//接收消息  
 @Test  
 **public void** recivemesage(){  
 Object o = **rabbitTemplate**.receiveAndConvert(**"atguigu"**);//接收**atguigu队列上的消息**  
 System.***out***.println(o.getClass());  
 System.***out***.println(o);  
 }

}

5、rabbitmq默认使用的是jdk的序列化器和转化器，想要转为json可自己网站容器里注入一个json的转换器

@Configuration  
**public class** rabbitmq\_json {  
 @Bean  
 **public** MessageConverter messageConverter(){  
 **return new** Jackson2JsonMessageConverter();  
 }  
}

17、实际开发中如何使用rabbitmq（ @EnableRabbit+ @RabbitListener）

1、在主程序开启基于注解的rabbitmq

@EnableRabbit

2、

如一个销售系统，有下订单，和库存系统，那么可以这样

下订单系统，用 **rabbitTemplate**.convertAndSend向队列存入一个订单

@Test  
**public void** addbook(){

//向**atguigu队列发消息**

**rabbitTemplate**.convertAndSend(**"exchange.direct"**,**"atguigu"**,**new** book(**"红落寞"**,**"曹雪芹"**));  
}

库存系统可以监听相应的队列，进行处理

@RabbitListener：标注在方法上的时候，就会监听相应的队列，当监听到队列有消息的时候会进行消息获取并自动启动执行方法完成业务逻辑

@Service  
**public class** bookservice {  
 @RabbitListener(queues = **"atguigu"**) //监听**atguigu队列**  
 **public void** getmessageandbuy(book book){  
 System.***out***.println(**"收到书本消息"**);  
 System.***out***.println(book);  
 System.***out***.println(**"购买书本。。。。"**);  
 }  
  
}

当addbook向队列发消息时bookservice 就会执行得到如下结果

收到书本消息

book{name='红落寞', author='曹雪芹'}

购买书本。。。。

使用 创建交换器，队列、和绑定关系

**public void** create(){

//创建交换器  
  *amqpAdmin.declareExchange(new DirectExchange("zzx.exchange"));*

*//创建队列  
 amqpAdmin.declareQueue(new Queue("zzx.queue"));*

*//将二者绑定* **amqpAdmin**.declareBinding(**new** Binding(**"zzx.queue"**,Binding.DestinationType.***QUEUE***,**"zzx.queue"**,**"zzx"**,**null**));  
}

全文检索工具elasticsarch

elasticsarch是目前全文搜索引擎的首选。他可以快速的存储、搜索和分析海量数据，Elasticsearch是一个分布式搜索服务，提供Restful API，底层基于Lucene，采用多shard（分片）的方式保证数据安全，并且提供自动resharding的功能，github等大型的站点也是采用了ElasticSearch作为其搜索服务

总的来说elasticsearch可以存储数据（面向文档的，将一条数据存储为一个json，一个json为一个文档），分析数据，检索数据、最厉害的是做搜索

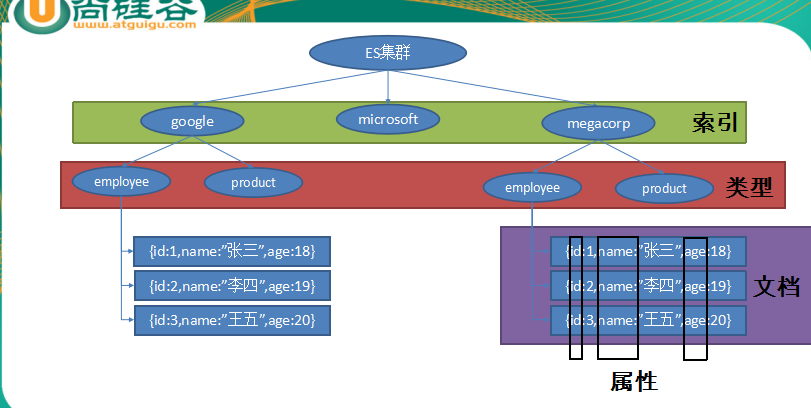
安装

1docker下载

2由于elasticsearch 默认需要2G启动内存，为了方式内存不足，使用-e命令限制其最大为256m

docker run -e ES\_JAVA\_OPTS="=Xms=256m -Xmx=256m" -d -p 9200:9200 -p 9300:9300 --name es01 37ad37f1c8a7

elasticsearch的数据存储方式（分布式存储）



Elasticsearch：相当于mysql

索引-数据库

类型-表

文档-表中的记录

属性-列

官方文档https://www.elastic.co/guide/cn/elasticsearch/guide/current/index.html，具体的检索方法和数据存储方法见文档

elasticsearch为什么能快速检索

Elasticsearch 使用一种称为 倒排索引 的结构，它适用于快速的全文搜索。

如

1. The quick brown fox jumped over the lazy dog
2. Quick brown foxes leap over lazy dogs in summer

对下面两个文本如何索引

Term Doc\_1 Doc\_2

-------------------------

Quick | | X

The | X |

brown | X | X

dog | X |

dogs | | X

fox | X |

foxes | | X

in | | X

jumped | X |

lazy | X | X

leap | | X

over | X | X

quick | X |

summer | | X

the | X |

---------------------

就是将一个个单词提取，然后建立表，然后再单词上标注是否有这个单词

22、Elasticsearch检索后会得出一个相关性的得分， 默认按照相关性得分排序，即每个文档跟查询的匹配程度，也是完全区别于传统关系型数据库的一个概念，数据库中的一条记录要么匹配要么不匹配，其实Elasticsearch和数据库相当，只是文档型数据库，更利于查询分类和分析数据

实际操作用Jest 或者 elasticsearchTemplate来进行存取查找，具体见22、23

异步任务

对于很多的任务如发送邮件，发送信息，这样的任务如果采用同步任务，那么响应的时间会很久，用户体验会很差，那么就可以使用异步任务，使用异步任务后，对于这样的任务，springboot会自动创建另一个线程去完成这个任务，那么就可以直接响应用户

使用过程

1. 在主配置类上标注@EnableAsync，告诉springboot开启异步任务

@EnableAsync  
@SpringBootApplication  
**public class** TaskSyncMailDingshiApplication {

1. 在需要异步执行的方法（一般是需要花费很多时间的方法，如发邮件）上面则标注@saync

@Async  
**public void** tssync(){  
 **try** {  
 Thread.*sleep*(3000);  
 } **catch** (InterruptedException e) {  
 e.printStackTrace();  
 }  
 System.***out***.println(**"易阻塞任务完成"**);  
 }

定时任务

每隔一段时间自动执行的任务

1. 在主配置类上面标注@EnableScheduled，开启自动配置

@EnableScheduling//开启定时任务  
@EnableAsync  
@SpringBootApplication  
**public class** TaskSyncMailDingshiApplication {

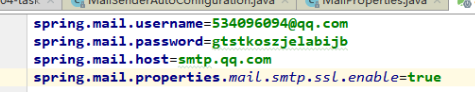
1. 在需要定时执行的方法上面标志@Scheduled，表明这是一个自动知心的方法

@Scheduled(cron = **"0/1 \* \* \* \* ? "**)  
**public void** scheduletask(){  
 System.***out***.println(**"定时任务。。。。"**);  
}

1. cron、时间指定表达式，可以正在http://cron.qqe2.com/里面自动生成

26 发送邮件

1. 引入email的start
2. 配置发送的邮箱以及一个授权码和qq的主机，以及ssl



1. 自动注入javaemailTemplate进行发邮件操作，见（26）

27、用spring的security 可以来做权限控制，

就是不同级别的人能访问不同的内容，如在图书管理系统，学习只能查书，而管理员可以看库存情况和借还情况。

1认证与授权

认证：就是验证账号密码

授权：确定这个账号能做什么，就是他的权限有什么，不同权限的人能做不同的事情

2 pring的Security做认证和授权

1 引入相关的start

2 编写配置类，这个配置类必须继承WebSecurityConfiguration，来做security的权限控制，然后再上面表注

@EnableWebSecurity

3、使用配置类进行相关的设置

@EnableWebSecurity  
**public class** mySecurityConfig **extends** WebSecurityConfigurerAdapter {  
 **protected void** configure(HttpSecurity http) **throws** Exception {  
  
 *//开启登陆请求授权功能(指定什么权限的人能做什么事)* http.authorizeRequests().antMatchers(**"/"**).permitAll()*//"/"可以通过* .antMatchers(**"/level1/\*\*"**).hasRole(**"VIP1"**)*//VIP1才能访问level1下的所有请求* .antMatchers(**"/level2/\*\*"**).hasRole(**"VIP2"**)*//* .antMatchers(**"/level3/\*\*"**).hasRole(**"VIP3"**);*//  
  
 //开启更录验证功能,如果没有权限，就会来到登录页* http.formLogin().loginPage(**"/userlogin"**);  
 *// /login就会来到登录页（security自己定义的）  
 // /出错的时候会重定向到/login?error  
 // 默认post的/login表示处理登陆（security自定义的处理规则处理）*

*//自定义loginPage后，/userlogin才是转向security处理登陆  
  
 //开启自动注销,返回首页* http.logout().logoutSuccessUrl(**"/"**);  
 *//退出用户清空sesion和cookie  
  
 //开启记住我* http.rememberMe();  
 *//会自动存cookie，时间为两周  
 //点击注销会删掉cookie* }  
  
  
 *//定义用户认证规则（指定什么人拥有什么权限）* **protected void** configure(AuthenticationManagerBuilder auth) **throws** Exception {  
 *//inMemoryAuthentication 从内存中获取，*passwordEncoder，security5后必须使用加密，BCrypt是默认的

*//*withUser，添加了账号并设置某个账号有什么权限auth.inMemoryAuthentication().passwordEncoder(**new** BCryptPasswordEncoder())  
 .withUser(**"zhangsan"**).password(**new** BCryptPasswordEncoder().encode(**"123456"**)).roles(**"VIP1"**,**"VIP2"**)  
 .and()  
 .withUser(**"lisi"**).password(**new** BCryptPasswordEncoder().encode(**"123456"**)).roles(**"VIP2"**,**"VIP3"**);  
  
 }  
}

分布式

随着网站不断变大，根据功能来拆分模块，如用户模块，订单模块，然后将不同的模块更具访问量部署到不同的机器，同时运行，而他们之间的相互调用，用dubbo或者springcloud，而他们之间还有一个注册中心zookeeper或者springcloud，注册中心里面存储着每个模块有什么功能，在什么地方怎么调用，当a模块要调用b模块的时候，先问注册中心，要的功能在什么地方，然后再去调用方法

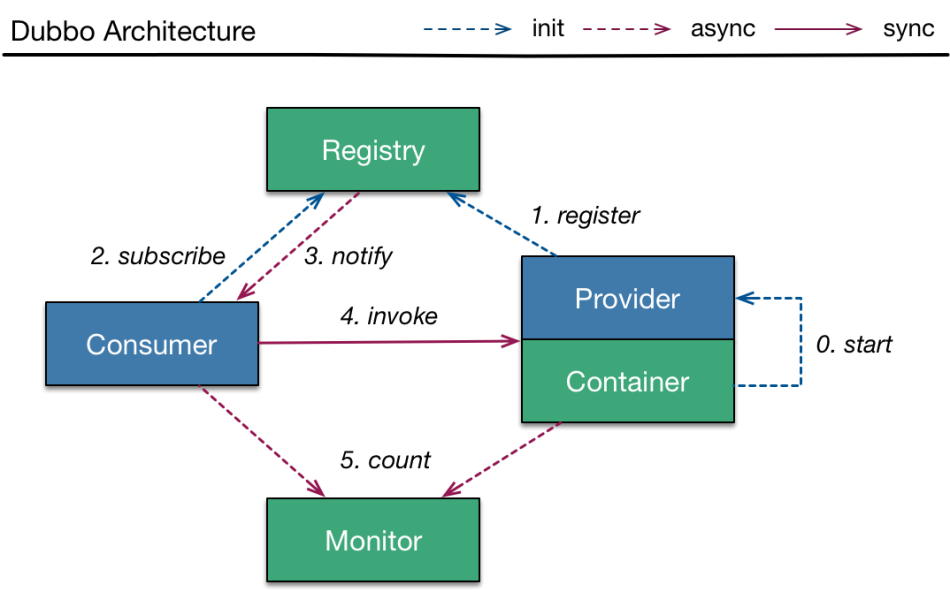
以dubbo和zookeper为例

ZooKeeper（注册中心）

ZooKeeper 是一个分布式的，开放源码的分布式应用程序协调服务。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。

Dubbo（分布式服务器）主要解决多个服务之间远程调用的问题（Rpc）

Dubbo是Alibaba开源的分布式服务框架，它最大的特点是按照分层的方式来架构，使用这种方式可以使各个层之间解耦合（或者最大限度地松耦合）。从服务模型的角度来看，Dubbo采用的是一种非常简单的模型，要么是提供方提供服务，要么是消费方消费服务，所以基于这一点可以抽象出服务提供方（Provider）和服务消费方（Consumer）两个角色。



Minitor：监控者，监控时时的数据

Container：是dubbo容器，里面有所有的分布式代码块

Provider：服务的提供者，他能够提供服务，也就是基于容器里面有什么模块来提供什么服务，并告诉注册中心，他能提供什么服务

Registry：注册中心，他管理者所有服务提供者所提供的服务的地址，及调用方法，消费者通过订阅他就可以得到所有的实体功能，也能通过它得到集体功能的地址，那么就可以去调用

Consumer：服务的消费者，他通过订阅注册中心得到服务提供者能提供什么服务，以及调用服务

整合与使用（与买票与卖票为例）

1生产者：这个module提供方法给消费者使用（提供卖票方法）

*/\*  
\* 1、引入资源dubbo和zookeeper客户端 zkclient*

*<!-- 引入dubbo-->*<**dependency**>  
 <**groupId**>com.alibaba.boot</**groupId**>  
 <**artifactId**>dubbo-spring-boot-starter</**artifactId**>  
 <**version**>0.1.0</**version**>  
</**dependency**>  
  
*<!--引入zookeeper的客户端-->  
<!-- https://mvnrepository.com/artifact/com.github.sgroschupf/zkclient -->*<**dependency**>  
 <**groupId**>com.github.sgroschupf</**groupId**>  
 <**artifactId**>zkclient</**artifactId**>  
 <**version**>0.1</**version**>  
</**dependency**> *\* 2、配置dubbo扫描包，zookeeper地址*

**dubbo.application.name**=**privoder\_movingticket  
dubbo.registry.address**=**zookeeper://192.168.43.76:2181  
dubbo.scan.base-packages**=**zzx.privoder\_movingticket.service**

*\* 3、在要发布的方法的类上加@service（alibaba的）来发布这个类里的方法*

**public interface** ticket {  
  
 **public** String getticket();  
}

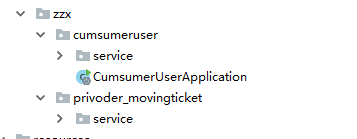
@Component  
*//扫描进入容器*@Service  
*//阿里巴巴的service，给zookeeper使用，表明将这个放啊注册到zookeeper***public class** tickerImpl **implements** ticket{  
  
 @Override  
 **public** String getticket() {  
 **return "<厉害了，我的过>"**;  
 }  
}

2消费者

*\* 1、引入dubbo，zkclient*

*<!-- 引入dubbo-->*<**dependency**>  
 <**groupId**>com.alibaba.boot</**groupId**>  
 <**artifactId**>dubbo-spring-boot-starter</**artifactId**>  
 <**version**>0.1.0</**version**>  
</**dependency**>  
  
*<!--引入zookeeper的客户端-->  
<!-- https://mvnrepository.com/artifact/com.github.sgroschupf/zkclient -->*<**dependency**>  
 <**groupId**>com.github.sgroschupf</**groupId**>  
 <**artifactId**>zkclient</**artifactId**>  
 <**version**>0.1</**version**>  
</**dependency**> *\* 2、配置资源配，zookeeper地址***dubbo.application.name**=**cumsumer-user  
dubbo.registry.address**=**zookeeper://192.168.43.76:2181**

*\* 3、把提供者的接口拿过来，实现类不要（赋值粘贴），全类名要相同*

 *\* 4、用@Reference注入对象，调用方法*

@Service  
**public class** userservice {  
  
 @Reference  
 ticket **tick**;  
  
 **public void** buy(){  
 System.***out***.println(**tick**);  
 String getticket = **tick**.getticket();  
 System.***out***.println(**"买到票了"**+getticket);  
 }  
}

springboot+springcloud实现微服务

1 SpringCloud分布式开发五大常用组件

服务发现——Netflix Eureka （注册器）

客服端负载均衡——Netflix Ribbon（负载均衡器）

断路器——Netflix Hystrix （当一个请求连续失败后，快速返回告知失败）

服务网关——Netflix Zuul （网络过滤）

分布式配置——Spring Cloud Config

2Spring Cloud 入门（创建一个空工程）

1. 创建provider

Provider是服务的提供者

1创建一个module，这个module要勾选“eureka Discovery”，引入eureka的发现与注册服务

2编写配置文件，配置服务器名，端口号等

**server:  
 port:** 8002  
**eureka:  
 instance:  
 prefer-ip-address: true** *#注册服务的时候使用ip地址* **client:  
 service-url:  
 defaultZone:** http://localhost:8761/eureka/  
**spring:  
 application:  
 name:** privoder-ticket //注意：当使用负载均衡的时候不能使用\_

3像普通工程一样编写service和controller，

@Service  
**public class** tickerservice {  
  
 **public** String getTicket(){  
 System.***out***.println(**"8002"**);  
 **return "《厉害了，我的国》"**;  
 }  
}

@RestController  
**public class** ticketCtrl {  
  
 @Autowired  
 tickerservice **tick**;  
   
 @GetMapping(**"/ticket"**)  
 **public** String getticket(){  
 String ticket = **tick**.getTicket();  
 **return** ticket;  
 }  
}

为什么编写controller？

我们知道在dubbo中不用编写controller，而主要提供服务的是service里面的方法，那么为什么要controller？

Springcloud在微服务进行通信的时候是使用轻量级的http进行通信，他的提供者的方法是通过controller暴露出去，就是将方法调用的结果用restful风格的json暴露，那么就可以通过网络去间接调用到这个方法，完成业务

1. 创建consumer

1创建一个module，这个module要勾选“eureka Discovery”，引入eureka的发现与注册服务

2编写配置文件，配置服务器名，端口号等

**server:  
 port:** 8200  
**eureka:  
 instance:  
 prefer-ip-address: true** *#注册服务的时候使用ip地址* **client:  
 service-url:  
 defaultZone:** http://localhost:8761/eureka/  
**spring:  
 application:  
 name:** cumsumer-user

3在配置类（这里直接在主程序）注入RestTemplate和开启eureka的自动发现注解（@EnableDiscoveryClient），并开启负载均恒（ @LoadBalanced）

RestTemplate：他是一个专门操作eureka 的一个对象，里面有一些方法可以远程调用生产者的方法

@EnableDiscoveryClient：开启发现远程服务功能，与RestTemplate配合使用

@LoadBalanced：标注在RestTemplate，当使用RestTemplate去远程调用生产者的服务的时候，就会自动用Springcloud的Ribbon去做负载均衡

@EnableDiscoveryClient *//开启发现功能，会自动在eurkea找服务*@SpringBootApplication  
**public class** CumsumerUserApplication {  
  
 **public static void** main(String[] args) {  
  
 SpringApplication.*run*(CumsumerUserApplication.**class**, args);  
 }  
  
 @LoadBalanced *//启用负载均衡机制，多个服务器（提供者）会交替起服务* @Bean  
 **public** RestTemplate restTemplate(){  
 **return new** RestTemplate();  
 }  
}

4利用RestTemplate在controller里来调用远程生产者的方法

@RestController  
**public class** userctrl {  
  
 @Autowired  
 RestTemplate **restTemplate**;  
  
 @GetMapping(**"/buy"**)  
 **public** String buy(){

//利用**来远程调用生产者注入的方法**  
 String forEntity = **restTemplate**.getForObject(**"http://PRIVODER-TICKET/ticket"**,String.**class**);  
 **return "买了"**+forEntity;  
 }  
}

1. 引入Eureka注册中心

*\* 1配置注册中心的相关信息*

**server:  
 port:** 8761  
**eureka:  
 instance:  
 hostname:** erueka-service *#eureka实例主机名* **client:  
 fetch-registry: false** *#不把自己注册到eureka上* **register-with-eureka: false** *#自己不从eureka获取服务注册信息* **service-url:  
 defaultZone:** http://localhost:8761/eureka/

*\* 2添加@EnableEurekaServer，开启EurekaServer*

@EnableEurekaServer  
@SpringBootApplication  
**public class** EruekaServiceApplication {

热部署

在我们开发的时候当我们改变一些代码的时候，前台能够立即也做出改变，这样就嫩方便我们开发，这个功能由spring的一个启动器devTools可以做到，只要引入就可以了

1. 加入启动器

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-devtools</**artifactId**>  
 <**scope**>runtime</**scope**>  
</**dependency**>

2当我们改变某些代码的时候只需要Ctrl+F9就会重新编译，前台（页面和逻辑都可以）也会做出改变，对于开发很好用

监控管理

监控管理可以帮我们很好的监控整个程序的健康状况，如注入了什么bean，各个映射情况等等

通过引入spring-boot-starter-actuator，可以使用Spring Boot为我们提供的准生产环境下的应用监控和管理功能。我们可以通过HTTP，JMX，SSH协议来进行操作，自动得到审计、健康及指标信息等

1. 引入spring-boot-starter-actuator，程序就会自动加入很多映射，这些映射的目的地就是程序的监控信息
2. 通过http请求查看程序的指标

监控信息与地址表，在springboot2.0之后只暴露几个监控点，需要全部暴露需要自己配置一下，如下

|  |  |
| --- | --- |
| **端点名** | **描述** |
| *autoconfig* | 所有自动配置信息 |
| auditevents | 审计事件 |
| beans | 所有Bean的信息 |
| configprops | 所有配置属性 |
| dump | 线程状态信息 |
| env | 当前环境信息 |
| health | 应用健康状况 |
| info | 当前应用信息 |
| metrics | 应用的各项指标 |
| mappings | 应用@RequestMapping映射路径 |
| shutdown | 关闭当前应用（默认关闭） |
| trace | 追踪信息（最新的http请求） |

Springboot2.0 之后暴露全部监控的配置

**management:** *#actuator* **server:  
 port:** 8081  
 **endpoints:  
 web:  
 base-path:** /  
 **exposure:  
 include: "\*"**