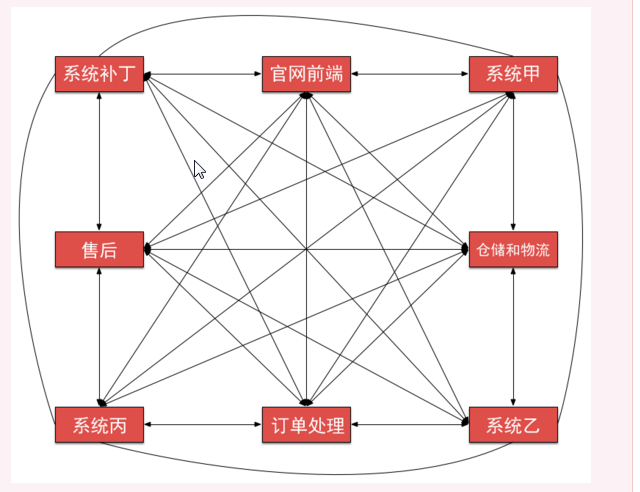
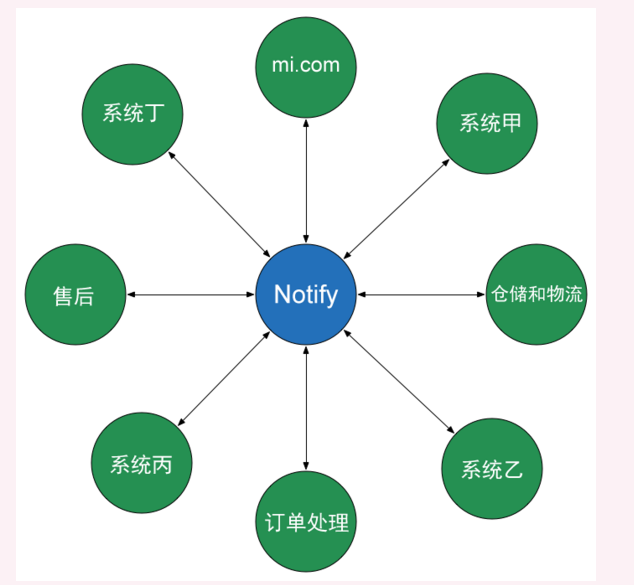
# RocketMq使用

## 消息中间件

消息中间件仅有一个功能：接收Producer发送的消息，发往Consumer。这样带来了一个好处，将Producer和Consumer**解耦**。在分布式系统中，如果不用消息中间件，各个分布式模块的连接关系将非常复杂，需要维护的接口越来越多，比如小米网一段时间的架构如下：



在小米使用异步消息服务系统（Notify）后，架构变成如下：



每个子系统仅仅需要面向消息中间件编程。

为了提高分布式系统的可用性和灵活性，消息中间件一般还具有**异步**的特性：Producer不用等待Consumer获取再返回，将消息发送给消息中间件即可。相对于同步，异步减少了等待时间，效率和系统吞吐量大大提升，异步也有利于系统的并发特性和解耦和。

## RocketMq架构

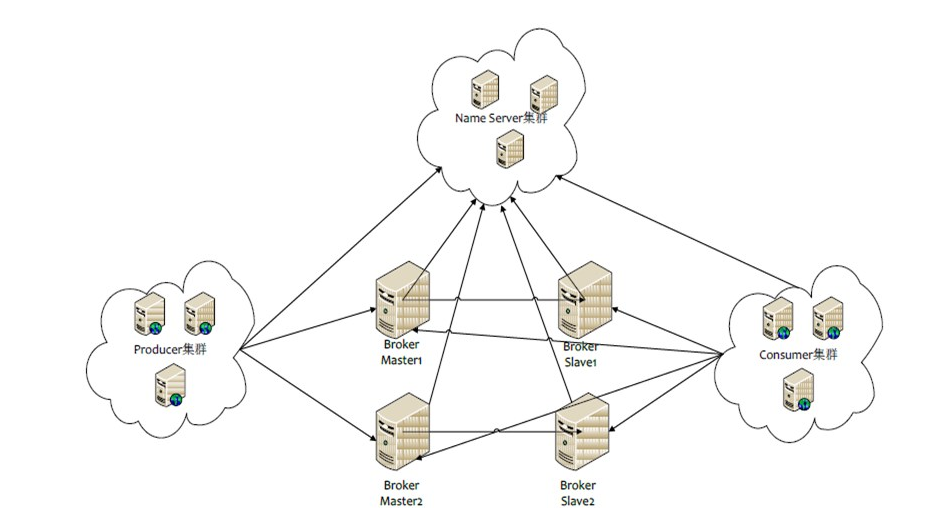
RocketMq是基于发布/订阅模型，Producer发布特定topic的消息，Consumer订阅感兴趣的topic，消息中间件负责分类转发。

由于是分布式环境下，需要进行域名服务，也就是解决以下两个问题：Producer将消息发往哪个消息中间件节点，Consumer从哪个消息中间件节点获取消息。

域名服务由RocketMq的NameServer节点完成。

分布式环境下，消息中间件需要处理海量的处理，消息转发需要具有可扩展性，来进行负载均衡和保障可靠性。消息转发的具体功能由RocketMq的Broker节点完成，Broker节点和Mysql节点类型，可以分为Master节点和Slave节点。

下图：



其中，NameServer和Broker都可以集群部署。

RocketMq相关概念如下

|  |  |
| --- | --- |
| Producer | 消息生产者，负责产生消息，一般由业务系统负责产生消息 |
| Consumer | 消息消费者，负责消费消息，一般是后台系统负责异步消费 |
| Push Consumer | Consumer的一种，应用通常向Consumer对象注册一个Listener接口，一旦收到消息，Consumer对象立刻回调Listener接口方法 |
| Pull Consumer | Consumer的一种，应用通常主动调用Consumer的拉消息方法从Broker拉消息，主动权由应用控制 |
| Producer Group | 一类Producer的集合名称，这类Producer通常发送一类消息，且发送逻辑一致 |
| Consumer Group | 一类Consumer的集合名称，这类Consumer通常消费一类消息，且消费逻辑一致 |
| Broker | 消息中转角色，负责存储消息，转发消息，一般也称为Server。 |

## RocketMq部署和使用

部署主要是部署NameServer和Brokre，相关网址如下：

<http://blog.csdn.net/tianwei7518/article/category/2842681>

使用RocketMq需要编写Consumer和Producer，一个简单的例子如下

public class Consumer {

public static void main(String[] args) throws InterruptedException, MQClientException {

DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("QuickStartConsumer");

consumer.setNamesrvAddr("127.0.0.1:9876");

consumer.setInstanceName("QuickStartConsumer");

consumer.subscribe("QuickStart", "\*");

consumer.registerMessageListener(new MessageListenerConcurrently() {

@Override

public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,

ConsumeConcurrentlyContext context) {

System.out.println(Thread.currentThread().getName() + " Receive New Messages: " + msgs);

return ConsumeConcurrentlyStatus.CONSUME\_SUCCESS;

}

});

consumer.start();

System.out.println("Consumer Started.");

}

}

public class Producer {

public static void main(String[] args) throws MQClientException, InterruptedException {

DefaultMQProducer producer = new DefaultMQProducer("QuickStartProducer");

producer.setNamesrvAddr("127.0.0.1:9876");

producer.setInstanceName("QuickStartProducer");

producer.start();

for (int i = 0; i < 1000; i++) {

try {

Message msg = new Message("QuickStart",// topic

"TagA",// tag

("Hello RocketMQ ,QuickStart" + i).getBytes()// body

);

SendResult sendResult = producer.send(msg);

System.out.println(sendResult);

}

catch (Exception e) {

e.printStackTrace();

Thread.sleep(1000);

}

}

producer.shutdown();

}

}

这部分安装和部署，需要进行测试，并搭建平台，便于测试。

# 比赛中RocketMq如何使用

赛题：脱敏后的双11交易数据存储在RocketMQ中，选手利用JStorm从RocketMQ拉取数据，按照要求实时计算出结果，并将结果写入Tair。

我们需要Consumer代码，来从RockerMq中获取数据。

如何获取数据，和在JStrorm中的计算模型相关。总的来说，应该是根据赛题，采用某种能减少计算次数（算两类的数据，能否减少复用一些计算），并行计算的方式。

具体方式，等5月下旬，数据出来了再进行讨论。

# 性能优化

1：消息查询方式，by id还是by type

2：尽量从RocketMQ的内存读，而不是磁盘

3：处理消息堆积

4：是否使用消息过滤

5：长连接

6：是否建立多个Consumer