https://github.com/killme2008/Metamorphosis/wiki/%E7%AE%80%E5%8D%95%E4%BE%8B%E5%AD%90

消息中间件中有两个角色：消息生产者和消息消费者。Meta里同样有这两个概念，消息生产者负责创建消息并发送到meta服务器，meta服务器会将消息持久化到磁盘，消息消费者从meta服务器拉取消息并提交给应用消费。我们假设你已经部署了你的meta服务器，参见如何开始。

会话工厂配置好zookeeper，创建生产者和消费者

Java客户端例子

推荐你使用maven，引用meta java client非常简单：

```
<dependency>
    <groupId>com.taobao.metamorphosis</groupId>
    <artifactId>metamorphosis-client</artifactId>
    <version>1.4.6.2</version>
</dependency>
```

创建一个示例工程，或者直接将metamorphosis-example clone出来到你本地机器。

```
git clone git://github.com/killme2008/metamorphosis.git
```

导入metamorphosis-example工程。

请注意，1.4.3及以上版本的java客户端只能连接1.4.3及以上版本的MetaQ服务器，而1.4.3之前的老客户端则没有限制。主要是因为1.4.3引入了发布和订阅topic的分离，1.4.3的新客户端只能查找到新版本的broker

消息会话工厂类

在使用消息生产者和消费者之前，我们需要创建它们，这就需要用到消息会话工厂类——MessageSessionFactory，由这个工厂帮你创建生产者或者消费者。除了这些，MessageSessionFactory还默默无闻地在后面帮你做很多事情，包括：

1.服务的查找和发现，通过diamond和zookeeper帮你查找日常的meta服务器地址列表

2.连接的创建和销毁，自动创建和销毁到meta服务器的连接，并做连接复用，也就是到同一台meta的服务器在一个工厂内只维持一个连接。

3.消息消费者的消息存储和恢复，后续我们会谈到这一点。
4.协调和管理各种资源，包括创建的生产者和消费者的。
因此，我们首先需要创建一个会话工厂类，MessageSessionFactory仅是一个接口，它的实现类是MetaMessageSessionFactory：

```
MessageSessionFactory sessionFactory = new MetaMessageSessionFactory(new
MetaClientConfig());
```

请注意,MessageSessionFactory应当尽量复用，也就是作为应用中的单例来使用，简单的做法是交给spring之类的容器帮你托管。

消息生产者

```
package com.taobao.metamorphosis.example;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import com.taobao.metamorphosis.Message;
import com.taobao.metamorphosis.client.MessageSessionFactory;
import com.taobao.metamorphosis.client.MetaClientConfig;
import com.taobao.metamorphosis.client.MetaMessageSessionFactory;
import com.taobao.metamorphosis.client.producer.MessageProducer;
import com.taobao.metamorphosis.client.producer.SendResult;
import com.taobao.metamorphosis.utils.ZkUtils.ZKConfig;

public class Producer {
    public static void main(String[] args) throws Exception {
        final MetaClientConfig metaClientConfig = new MetaClientConfig();
        final ZKConfig zkConfig = new ZKConfig();
        //设置zookeeper地址
        zkConfig.zkConnect = "127.0.0.1:2181";
        metaClientConfig.setZkConfig(zkConfig);
        // MessageSessionFactory,强烈建议使用单例
        MessageSessionFactory sessionFactory =
            new MetaMessageSessionFactory(metaClientConfig);
        // create producer,强烈建议使用单例
        MessageProducer producer = sessionFactory.createProducer();
        // publish topic
        final String topic = "test";
        producer.publish(topic);
```

```
        BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
        String line = null;
        while ((line = reader.readLine()) != null) {
            // send message
            SendResult sendResult =
                producer.sendMessage(new Message(topic,
line.getBytes()));
            // check result
            if (!sendResult.isSuccess()) {
                System.err.println("Send message failed,error message:" +
sendResult.getErrorMessage());
            }
            else {
                System.out.println("Send message successfully,sent to " +
sendResult.getPartition());
            }
        }
    }
}
```

消息生产者的接口是MessageProducer，你可以通过它来发送消息。创建生产者很简单，通过MessageSessionFactory的createProducer方法即可以创建一个生产者。在Meta里，每个消息对象都是Message类的实例，Message表示一个消息对象，它包含这么几个属性：

- id: Long型的消息id,消息的唯一id，系统自动产生，用户无法设置，在发送成功后由服务器返回，发送失败则为0。
- topic: 消息的主题，订阅者订阅该主题即可接收发送到该主题下的消息，生产者通过指定发布的topic查找到需要连接的服务器地址，必须。
- data: 消息的有效载荷，二进制数据，也就是消息内容，meta永远不会修改消息内容，你发送出去是什么样子，接收到就是什么样子。消息内容通常限制在1M以内，我的建议是最好不要发送超过上百K的消息，必须。数据是否压缩也完全取决于用户。
- attribute: 消息属性，一个字符串，可选。发送者可设置消息属性来让消费者过滤。

细心的朋友可能注意到，我们在sendMessage之前还调用了MessageProducer的publish(topic)方法

```
producer.publish(topic);
```

这一步在发送消息前是必须的，你必须发布你将要发送消息的topic，这是为了让会话工厂帮你去查找接收这些topic的meta服务器地址并初始化连接。这个步骤针对每个topic只需要做一次，多次调用无影响。
总结下这个例子，从标准输入读入你输入的数据，并将数据封装成一个Message对象，发送到topic为test的服务器上。
请注意，MessageProducer是线程安全的，完全可重复使用，因此最好在应用中作为单例来使用，一次创建，到处使用，配置为spring里的singleton bean。MessageProducer创建的代价昂贵，每次都需要通过zk查找服务器并创建tcp长连接。

消息消费者

发送消息后，消费者可以接收消息了，下面的代码创建消费者并订阅test这个主题，等待消息送达并打印消息内容

```
package com.taobao.metamorphosis.example;

import java.util.concurrent.Executor;
import com.taobao.metamorphosis.Message;
import com.taobao.metamorphosis.client.MessageSessionFactory;
import com.taobao.metamorphosis.client.MetaClientConfig;
import com.taobao.metamorphosis.client.MetaMessageSessionFactory;
import com.taobao.metamorphosis.client.consumer.ConsumerConfig;
import com.taobao.metamorphosis.client.consumer.MessageConsumer;
import com.taobao.metamorphosis.client.consumer.MessageListener;
import com.taobao.metamorphosis.utils.ZkUtils.ZKConfig;

public class AsyncConsumer {
    public static void main(String[] args) throws Exception {
        final MetaClientConfig metaClientConfig = new MetaClientConfig();
        final ZKConfig zkConfig = new ZKConfig();
        //设置zookeeper地址
        zkConfig.zkConnect = "127.0.0.1:2181";
        metaClientConfig.setZkConfig(zkConfig);
        // New session factory,强烈建议使用单例
        MessageSessionFactory sessionFactory
            = new MetaMessageSessionFactory(metaClientConfig);
        // subscribed topic
```

```
        final String topic = "test";
        // consumer group
        final String group = "meta-example";
        // create consumer,强烈建议使用单例
        MessageConsumer consumer
            = sessionFactory.createConsumer(new ConsumerConfig(group));
        // subscribe topic
        consumer.subscribe(topic, 1024 * 1024, new MessageListener() {

            public void recieveMessages(Message message) {
                System.out.println("Receive message " + new
String(message.getData()));
            }
            public Executor getExecutor() {
                // Thread pool to process messages,maybe null.
                return null;
            }
        });
        // complete subscribe
        consumer.completeSubscribe();
    }
}
```

通过createConsumer方法来创建MessageConsumer，注意到我们传入一个ConsumerConfig参数，这是消费者的配置对象。每个消息者都必须有一个ConsumerConfig配置对象，我们这里只设置了group属性，这是消费者的分组名称。Meta的Producer、Consumer和Broker都可以为集群。消费者可以组成一个集群共同消费同一个topic，发往这个topic的消息将按照一定的负载均衡规则发送给集群里的一台机器。同一个消费者集群必须拥有同一个分组名称，也就是同一个group。我们这里将分组名称设置为meta-example。订阅消息通过subscribe方法，这个方法接受三个参数

- topic，订阅的主题
- maxSize，因为meta是一个消费者主动拉取的模型，这个参数规定每次拉取的最大数据量，单位为字节，这里设置为1M，默认最大为1M。
- MessageListener，消息监听器，负责消息消息。

MessageListener的接口方法如下:

```
public interface MessageListener {
    /**
     * 接收到消息列表，只有message不为空并且不为null的情况下会触发此方法
```

```
     * @param message
     */
    public void recieveMessages(Message message);


    /** 处理消息的线程池 */
    public Executor getExecutor();
}
```

消息的消费过程可以是一个并发处理的过程，getExecutor返回你想设置的线程池，每次消费都会在这个线程池里进行。recieveMessage方法用于实际的消息消费处理，message参数即为消费者收到的消息，它必不为null。我们这里简单地打印收到的消息内容就完成消费。如果在消费过程中抛出任何异常，该条消息将会在一定间隔后重新尝试提交给MessageListener消费。在多次消费失败的情况下，该消息将会存储到消费者应用的本次磁盘，并在后台自动恢复重试消费。

细心的你一定还注意到，在调用subscribe之后，我们还调用了completeSubscribe方法来完成订阅过程。请注意，subscribe仅是将订阅信息保存在本地，并没有实际跟meta服务器交互，要使得订阅关系生效必须调用一次completeSubscribe，completeSubscribe仅能被调用一次，多次调用将抛出异常。 为什么需要completeSubscribe方法呢，原因有二：

- 首先，subscribe方法可以被调用多次，也就是一个消费者可以消费多种topic
- 其次，如果每次调用subscribe都跟zk和meta服务器交互一次，代价太高

因此completeSubscribe一次性将所有订阅的topic生效，并处理跟zk和meta服务器交互的所有过程。

同样,MessageConsumer也是线程安全的，创建的代价不低，因此也应该尽量复用。