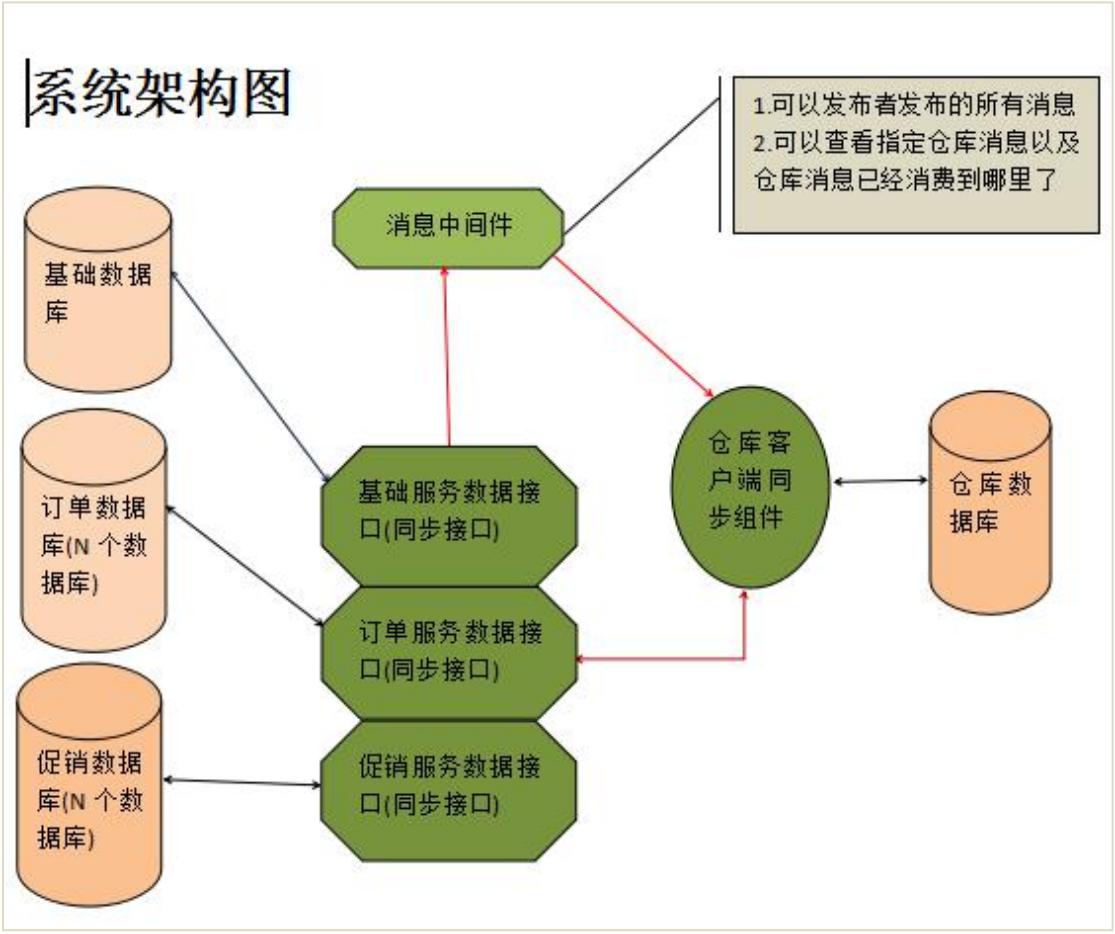
仓库系统数据同步方案

一、基本思想

同步方案采取消息机制，领域层触发领域事件，对发生的存储(Created)，更新(Changed)，删除(Deleted)，当然根据具体业务需求可以分析出其他事件（比如：需要单独跟踪接货等等）事件进行记录（相当于发布者 Publisher），异构系统端(相当 Subscriber)采取消费消息的方式来进行数据同步。系统尽量考虑到到通用性以及系统与系统之间的松耦合性。

二、系统架构图



三、系统实现

3.0 消息中心

消息中心用于接收消息发布者发布的消息以及推送消息给特定仓库。消息中能够管理所有的已经发布的消息。或者可以查看所有仓库消息的消费情况。消息中心的消息在刷新到存储介质后，只会保留 7 天，即：当消息发布者发布消息 7 天后，此条消息将会从消息队列里删除（删除的消息会被保存到另外存储介质，备查。）

3.1 消息的分类

消息分全局消息(Global Message)与特定消息 2 种，全局消息为所有仓库都需要接受的消息，比如：基础资源库里的分类，商品资料等等。特定消息(Specified Message)即只有某个仓库才能够接受到的消息。

3.2 消息的存储与消息传输对象定义

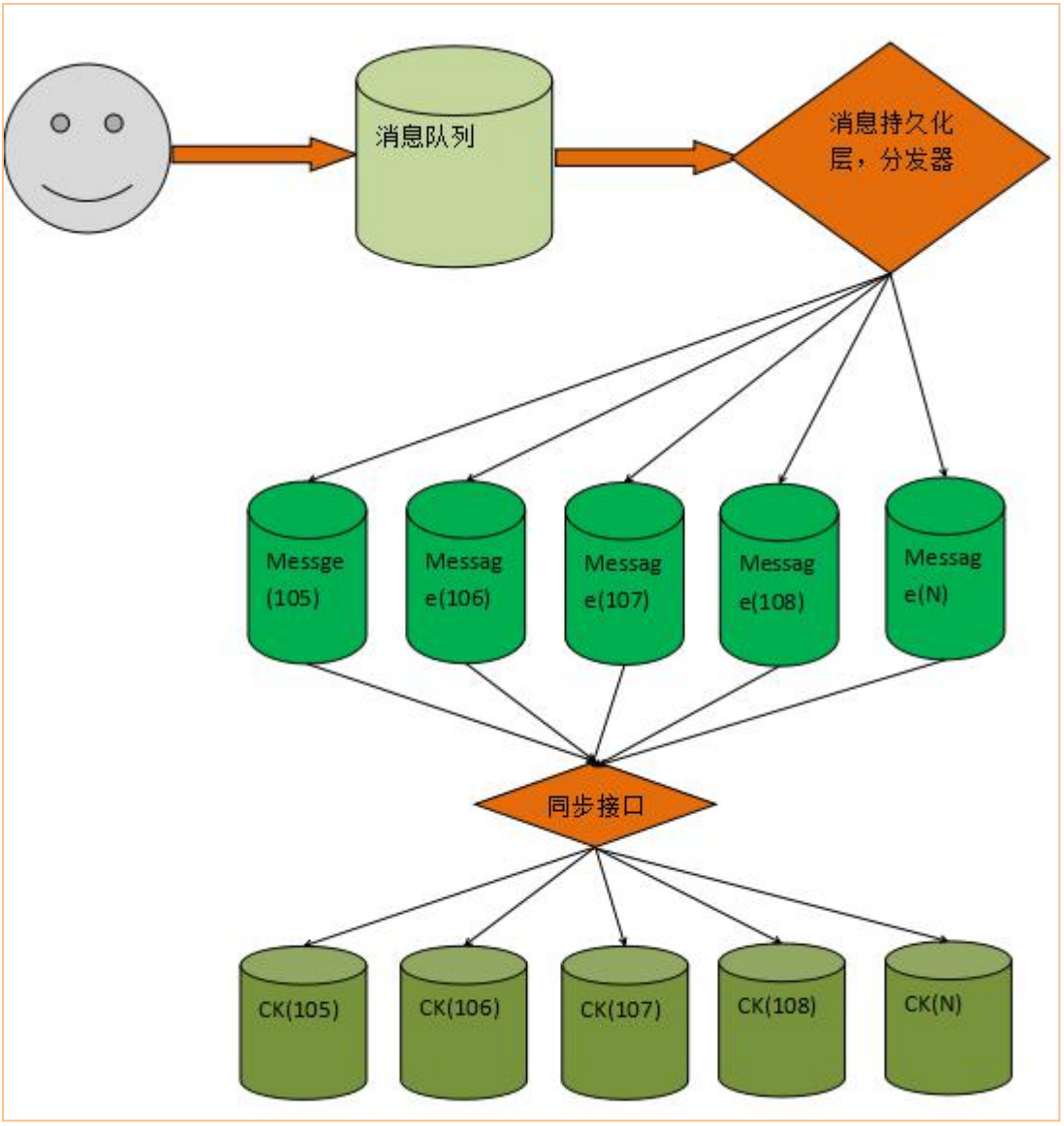
3.2.1 消息体对象

```
/// <summary>
/// 消息传输对象
/// </summary>
public class Message : IEquatable<Message>
{
    /// <summary>
    /// 消息唯一 ID，有序
    /// </summary>
    public string Id { get; set; }

    /// <summary>
    /// 消息主题，比如：Order_Created,Order_Changed,Order_Delete 等
    /// </summary>
    public string Topic { get; set; }

    /// <summary>
    /// 消息摘要,采取 JSON 保存方式，
```

```
        比如：{OrderId:"10001",Created:"2016-1-1 00:00:00"}
    /// </summary>
    public string Body { get; set; }

    /// <summary>
    /// 对应的仓库编号，注意：如果值为 0 代表为全局消息，
       所有仓库都必须要消费的，>0 为特定仓库才能收到的消息
    /// </summary>
    public int WID { get; set; }

    /// <summary>
    /// 消息发布时间
    /// </summary>
    public DateTime Created { get; set; }

    /// <summary>
    ///
    /// </summary>
    /// <param name="other"></param>
    /// <returns></returns>
    public bool Equals(Message other)
    {
        if (null == other)
            return false;
        return this.Id.Equals(other.Id);
    }
}
```

为了保证后续消息存储和数据分布式迁移，我们将消息 ***Message.Id*** *设计为生成为全局有序* ***GUID。***

3.2.2 系统存储架构图



3.3 消息发布者(Publisher)接口

同步组件系统会提供一个简单封装的消息发布接口。业务层（或者接口层）只要使用提供的封装来发布消息即可。接口很简单，只会定义一系列的发布事件，代码暂时演示如下：

```
EventBus.OrderCreated(string orderId,DateTime created);
EventBus.OrderChanged(string orderId,DateTime created);
EventBus.OrderDeleted(tring orderId,DateTime created);

.........................
```

3.4 客户端同步工具(Subscriber)的工作方式

客户端同步组件暂时采取轮询的方式，即定时比如：1 分钟从消息中间件里拉取 100 条消息，进行消费。然后通过接口的 Topic 主题来确定触发哪个事件，然后调用哪个同步接口。返回数据后再进行加工，直接将数据插入，更新等仓库数据库。由于消息消费必须要串行化进行，即：不能先消费后产生的消息，因此为了拉取效率和执行速度。我们会在消息中间件里记录下仓库当前拉取消息所在的消息索引号。以便于下次直接过滤掉已经消费的消息。

调用代码如下：

```
    /// <summary>
       /// 消息发布演示示例代码
       /// </summary>
       /// <param name="args"></param>
       static void Main(string[] args)
       {

           //消费端调用示例
           var consumerClient = new
MessageConsumerClient("http://localhost:88/Api",
               //客户端自己实现的序列化与反序列化对象
               new DefaultJosnSerializer(),
               //获取仓库编号委托，需要客户端自己实现具体的获取方式
               () => 105);
           //consumerClient.OnMessageRequestSuccess +=
ConsumerClient_OnMessageRequestSuccess;
           //consumerClient.OnMessageRequestError +=
ConsumerClient_OnMessageRequestError;
           //consumerClient.OnMessageConsumeComplete +=
ConsumerClient_OnMessageConsumeComplete;
           consumerClient.OnCategoriesChanged +=
ConsumerClient_OnCategoriesChanged;
           consumerClient.OnBuyOrderCreated += ConsumerClient_OnBuyOrderCreated;
           consumerClient.OnSaleOrderCreated +=
ConsumerClient_OnSaleOrderCreated;
```

```
            consumerClient.OnSaleOrderChanged +=
ConsumerClient_OnSaleOrderChanged;
            consumerClient.OnProductCreated += ConsumerClient_OnProductCreated;
            consumerClient.Start("160704133235041047000007210", 1000);

            Console.ReadLine();

            //初始化消息发布者客户端
            var publisherClient = new
MessagePublisherClient("http://localhost:88/Api",
                //客户端自己实现的序列化与反序列化对象
                new DefaultJosnSerializer(),
                //获取仓库编号委托，需要客户端自己实现具体的获取方式
                () => 105);

            //测试循环触发领域事件，发布消息
            Parallel.For(0, 1000, i =>
            {
                var r = publisherClient.CategoriesCreated(103);
                Console.WriteLine(r.IsSuccess + "," + r.MessageId + "," + r.Message);
                publisherClient.BuyOrderCreated("100000000");
                Console.WriteLine(r.IsSuccess + "," + r.MessageId + "," + r.Message);
                r = publisherClient.SaleOrderChanged("100000000000000");
                Console.WriteLine(r.IsSuccess + "," + r.MessageId + "," + r.Message);
                r = publisherClient.SaleOrderCreated("100000000111111",
DateTime.Now);
                Console.WriteLine(r.IsSuccess + "," + r.MessageId + "," + r.Message);
                r = publisherClient.CategoriesChanged(102);
                Console.WriteLine(r.IsSuccess + "," + r.MessageId + "," + r.Message);
                r = publisherClient.ProductCreated(10000000002);
                Console.WriteLine(r.IsSuccess + "," + r.MessageId + "," + r.Message);
            });

            Console.ReadLine();
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="arg1"></param>
        /// <param name="arg2"></param>
        private static void ConsumerClient_OnProductCreated(object arg1,
ProductCreatedEventAgrs arg2)
        {
```

```
            Console.WriteLine(arg2.Message.Id + "-->" + arg2.Message.Topic);
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="arg1"></param>
        /// <param name="arg2"></param>
        private static void ConsumerClient_OnMessageConsumeComplete(object arg1,
ResponseResult<dynamic> arg2)
        {

            Console.WriteLine("消费消息完成，总共消费消息数：" + arg2.Data.Count);
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="arg1"></param>
        /// <param name="arg2"></param>
        private static void ConsumerClient_OnMessageRequestSuccess(object arg1,
ResponseResult<dynamic> arg2)
        {
            Console.WriteLine(new DefaultJosnSerializer().Serialize(arg2));
            System.Threading.Thread.Sleep(2000);
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="arg1"></param>
        /// <param name="arg2"></param>
        private static void ConsumerClient_OnMessageRequestError(object arg1,
ResponseResult<dynamic> arg2)
        {
            Console.WriteLine("出现错误了：" + arg2.Info);
        }
```

3.4.1 消息分组处理

消息有时候发生具有连续性，比如：分类 A，在触发创建事件后会生成 Created 消息。然后又联系 2 次修改了，那么消息中心会存在 2 条 Changed 消息。我们只要处理最后一个 Changed 事件消息即可。这样

我们我们虽然接收到 3 条消息，但是只要处理一条消息，相当于将消息压缩。可以提高消息处理速度。

3.4.2 消息重试机制

当接收到消息后，消息中心实际已经记录了当前仓库消费消息的最后索引。因此当出现调用同步接口获取数据或者同步出现错误的时候，我们需要考虑到重试机制。暂时采取重试 10 次（可以调整），如果还是不成功，需要将消费错误的消息记录到本地。当前批全部消费完，还有错误。强制保存消息到客户端磁盘。并且给出声音或者其他提示。进行人工干预。

3.5 需要开发的组件任务

1. 基础数据获取接口
2. 订单数据获取接口
3. 促销中心数据获取接口
4. 消息中间件
5. 仓库客户端同步工具