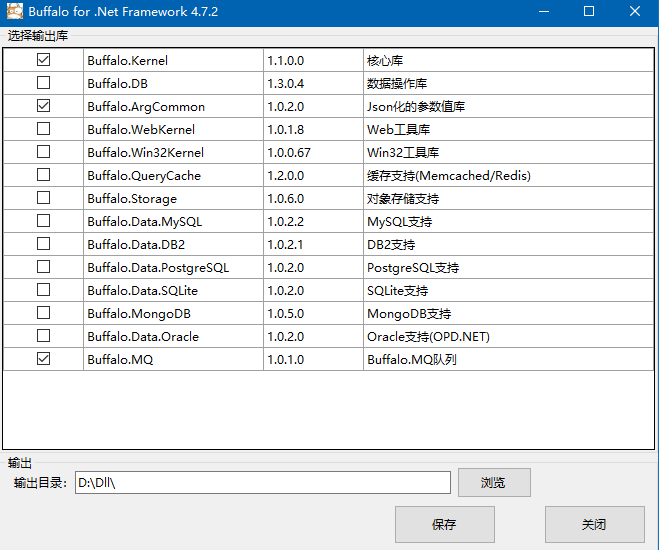
# Buffalo.MQ说明

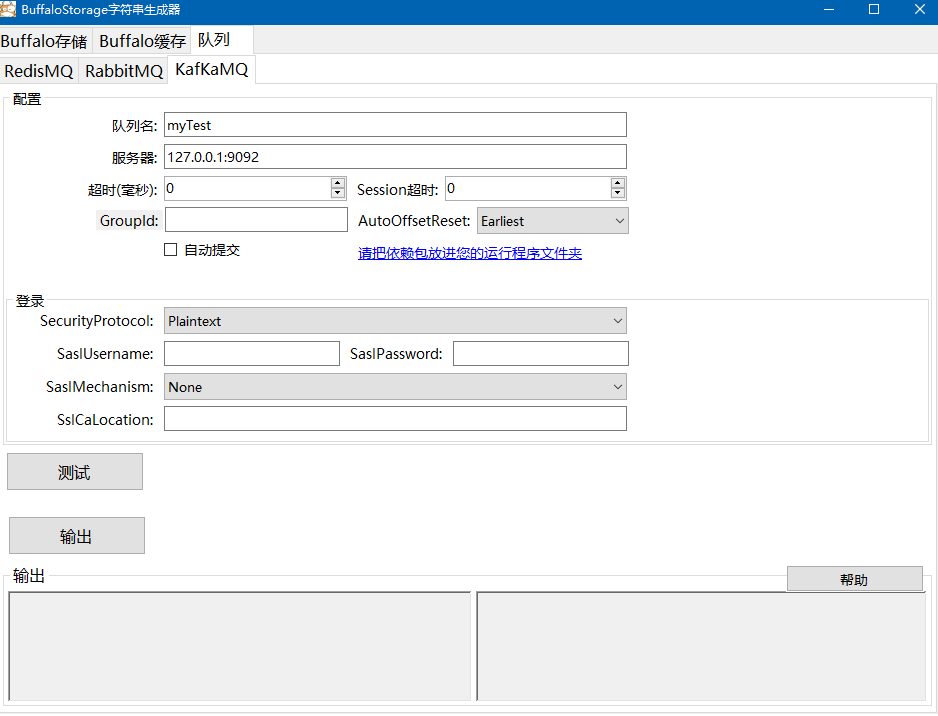
## 1.介绍

Buffalo.QueryCache是一个支持多种缓存调用的缓存封装包，通过此包可以方便使用和切换多种缓存

项目依赖文件输出可以在 Buffalo安装工具🡪选择对应的.net版本🡪输出 获取



连接字符串生成器可以在Buffalo安装工具🡪帮助🡪连接字符串🡪选择 队列 选项卡 然后选择对应的缓存选项卡，填入相关信息就可以输出对应的连接字符串



### 1.1错误处理

如果出现dll错误，请在config加入

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity name="System.Memory" publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-4.0.1.1" newVersion="4.0.1.1" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Runtime.CompilerServices.Unsafe" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-4.0.6.0" newVersion="4.0.6.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Numerics.Vectors" publicKeyToken="b03f5f7f11d50a3a" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-4.1.4.0" newVersion="4.1.4.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Buffers" publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-4.0.3.0" newVersion="4.0.3.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.IO.Pipelines" publicKeyToken="cc7b13ffcd2ddd51" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-4.0.2.1" newVersion="4.0.2.1" />

</dependentAssembly>

</assemblyBinding>

</runtime>

## 2.使用

### 2.1 初始化

把配置生成的字符串和MQ类型利用

MQUnit.SetMQInfo添加到配置表

string name = "myTest";

string connectString = "server=127.0.0.1%3a9092;groupId=1122;saslMechanism=0;securityProtocol=0;offsetType=1;autoCommit=0;";

string type = "kafkamq"

MQUnit.SetMQInfo(name, type, connectString);

### 2.2生产信息

MQConnection conn = MQUnit.GetMQConnection("myTest");

string value = DateTime.Now.ToString();

byte[] sendByte = System.Text.Encoding.UTF8.GetBytes(value);

conn.Send("app.Test", sendByte);

使用MQUnit.GetMQConnection获取生产者，通过conn.Send(string key,byte[] value)发送信息，

Key：路由键，消费者需要通过这个key来获取信息，不同消费者就依靠监听不同的键来获取自己的消息

Value：要发送的信息

### 2.3 消费信息

MQListener listener= MQUnit.GetMQListener("myTest");

listener.OnMQReceived += Listener\_OnMQReceived;//获取到数据的事件

listener.OnMQException += Listener\_OnMQException;//报异常的事件

listener.StartListend(new string[] { "app.Test" });//开启监听

if (!listener.WaitStart(5000))//等待监听开始，可以不写

{

Debug.WriteLine("等待开启超时");

}

private void Listener\_OnMQException(MQListener sender, Exception ex)

{

Debug.WriteLine(ex.Message);

}

private void Listener\_OnMQReceived(MQListener sender, string exchange, string routingKey, byte[] body, int partition, long offset)

{

string text = System.Text.Encoding.UTF8.GetString(body);

}

使用MQUnit. GetMQListener获取消费者，通过OnMQReceived事件获取信息，StartListend(IEnumerable<string> listenKeys) 开启监听，并监听listenKeys的键

### 2.4开启事务

using (MQTransaction tran = conn.StartTransaction())

{

string value = Guid.NewGuid().ToString();

conn.Send("app.Test", System.Text.Encoding.UTF8.GetBytes(value));

value = Guid.NewGuid().ToString();

conn.Send("app.Test", System.Text.Encoding.UTF8.GetBytes(value));

tran.Commit();

}

事务使用StartTransation

开启，然后需要用using包着需要事务的动作，在tran之前退出了方法，没执行Commit的话，事务将会自动回滚

### 2.5 批量发送

using (MQBatchAction ba = conn.StartBatchAction())

{

value = Guid.NewGuid().ToString();

conn.Send("app.Test", System.Text.Encoding.UTF8.GetBytes(value));

value = Guid.NewGuid().ToString();

conn.Send("app.Test", System.Text.Encoding.UTF8.GetBytes(value));

}

批量发送是等待using结束时候才一起发送，提升发送性能