外部メッセージング・プラットフォーム向け API の使用法

Version 2024.1
2024-06-03

InterSystems Corporation　1 Memorial Drive　Cambridge MA 02142　www.intersystems.com

外部メッセージング・プラットフォーム向け API の使用法
InterSystems IRIS Data Platform　Version 2024.1　2024-06-03

インターシステムズ社の製品に関するサポートやご質問は、以下にお問い合わせください：

InterSystems Worldwide Response Center (WRC)
Tel:　+1-617-621-0700
Tel:　+44 (0) 844 854 2917
Email:　support@InterSystems.com

目次

1

メッセージング API の使用法

InterSystems IRIS は、JMS、Kafka、RabbitMQ、Amazon Simple Notification Service (SNS)、Amazon Simple Queue Service (SQS) の各メッセージング・プラットフォームと直接通信するために使用できるメッセージング API を提供します。これらの API クラスは %External.Messaging パッケージに用意されています。

どのプラットフォームでも、大半のコード・フローは同一です。このページでは、この共通のフローについて説明し、共通のクラスとメソッドに関する詳細情報を提供します。特定のメッセージ・プラットフォームについて説明している技術文書もあります。上記の各リンクを参照してください。

(インターシステムズでは、このページで取り上げている各 API のほか、相互運用プロダクションで使用する特殊クラスも用意しています。これらのクラスを使用すると、上記と同様のメッセージング・プラットフォームと直接通信できます。詳細は、上記のリンクを参照してください。)

1.1 メッセージング・プラットフォームとの接続

メッセージを送受信するには、コードで以下の各処理を実行する必要があります。

1. プラットフォームとの接続に関する情報を収めた設定オブジェクトを作成します。このオブジェクトは、いずれかの設定クラスのインスタンスです。インターシステムズでは、プラットフォーム固有の各種設定クラスを提供しています。これらはすべて %External.Messaging.Settings のサブクラスです。適用できるクラスのインスタンスを作成し、必要に応じてそのプロパティを設定します。

   以下に Kafka での例を挙げます。

   ObjectScript

   ```
   Set settings = ##class(%External.Messaging.KafkaSettings).%New()
   Set settings.username = "amandasmith"
   Set settings.password = "234sdsge"
   Set settings.servers = "100.0.70.179:9092, 100.0.70.089:7070"
   Set settings.clientId = "BazcoApp"
   // If Consumer, specify a consumer group
   Set settings.groupId = "G1"
   ```

2. 同様にプラットフォーム固有のメッセージング・クライアント・オブジェクトを作成します。このオブジェクトは、すべてが %External.Messaging.Client のサブクラスであるクライアント・クラスのいずれかのインスタンスです。

   メッセージング・クライアント・オブジェクトを作成するには、%External.Messaging.Client の CreateClient() メソッドを呼び出して、設定オブジェクトを 1 番目の引数として渡します。例えば以下のようにします。

ObjectScript

```
Set client = ##class(%External.Messaging.Client).CreateClient(settings, .tSC)
// if tSC is an error, handle error scenario
```

このメソッドからは、参照渡しでステータス・コードが 2 番目の引数として返されます。コードでは、このステータスを確認したうえで処理を継続する必要があります。

CreateClient() からは、該当するクライアント・クラスのインスタンスが返されます。例えば、settings 変数が %External.Messaging.KafkaSettings のインスタンスであれば、返されるオブジェクトは %External.Messaging.KafkaClient のインスタンスです。

同様に、settings が %External.Messaging.SNSSettings のインスタンスであれば、返されるオブジェクトは %External.Messaging.SNSClient のインスタンスです。

1.2 トピックまたはキューの作成

メッセージを送信するには、使用しているプラットフォームとの関連性に応じてトピックまたはキューが必要になることがあります。

トピックまたはキューのプラットフォーム固有の設定は文字列として共通メソッドに渡されますが、これらの設定はオブジェクトを使用して定義されています。したがって、この設定を渡すときにはそのオブジェクトの ToJSON() メソッドを呼び出す必要があります。以下のコードは、共通メソッドを使用して新しい Kafka トピックを作成します。

ObjectScript

```
Set topic = "quick-start-events"
Set queueSettings = ##class(%External.Messaging.KafkaTopicSettings).%New()
Set queueSettings.numberOfPartitions = 1, queueSettings.replicationFactor = 1
Set tSC = client.CreateQueueOrTopic(topic, queueSettings.ToJSON())
```

一部のメッセージング・プラットフォームには、高度な機能を含む可能性のあるトピック/キューを作成するための独自のメソッドがあることに注意してください。

1.3 メッセージの送信

メッセージ・クライアント・オブジェクトおよび関連するトピック・オブジェクトまたはキュー・オブジェクトをすべて用意していれば、そのクライアント・オブジェクトのメソッドを次のように呼び出し、メッセージを送信します。

1. メッセージ・オブジェクトを作成します。これはメッセージ・クラスのインスタンスです。インターシステムズでは、プラットフォーム固有の各種メッセージ・クラスを提供しています。これらはすべて %External.Messaging.Message のサブクラスです。

   以下に Kafka での例を挙げます。

   ObjectScript

   ```
   Set msg = ##class(%External.Messaging.Messaging).%New()
   Set msg.topic = "quick-start-events"
   Set msg.value = "body of the message"
   Set msg.key = "somekey"
   ```

2. クライアント・オブジェクトの SendMessage() メソッドを呼び出し、メッセージ・オブジェクトを引数として渡します。例えば以下のようにします。

ObjectScript

```
Set tSC = client.SendMessage(msg)
// if tSC is an error, handle error scenario
```

このメソッドからはステータス・コードが返されます。このステータス・コードをコードで確認したうえで処理を続行する必要があります。

3. アプリケーションでトピックやキューを必要としなくなった時点で、クライアント・オブジェクトの DeleteTopicOrQueue() メソッドをコードで使用してトピックまたはキューを安全に削除できます。

ObjectScript

```
Do client.DeleteQueueOrTopic(topic)
```

一部のメッセージング・プラットフォームには、高度な機能を含む可能性のあるトピック/キューを削除するための独自のメソッドがあることに注意してください。

1.4 メッセージの受信

メッセージを受信できるメッセージング・プラットフォームでは、クライアント・オブジェクトの ReceiveMessage() メソッドを使用してメッセージを受信します。このメソッドの 1 番目の引数はトピックまたはキューの名前です。2 番目の引数は %ListOfObjects のインスタンスです。このメソッドからは、この 2 番目の引数を使用して参照渡しでメッセージが返されます。

ReceiveMessage() では、settings の JSON 形式文字列もオプションの 3 番目の引数として受け入れます。使用できる設定はプラットフォームによって異なり、プラットフォームごとに ReceiveSettings クラスのプロパティとして定義されています。このような設定を作成するには、ReceiveSettings クラスの新しいインスタンスを作成し、必要に応じてそのプロパティを設定します。つづいて、ReceiveSettings オブジェクトで継承した ToJSON() メソッドを使用して、目的の設定を ReceiveMessage() メソッドに指定します。

ReceiveMessage() メソッドからはステータス・コードが返されます。このステータス・コードをプログラムで確認したうえで処理を続行する必要があります。

以下の例では、JMS キュー (queue) から JMS クライアント (client) がメッセージを受信します。メッセージ取得のタイムアウトとして 200 ミリ秒を設定しています。

ObjectScript

```
Set rset = ##class(%External.Messaging.JMSReceiveSettings).%New()
Set rset.receiveTimeout = 200

#dim messages As %ListOfObjects
Set tSC = client.ReceiveMessage(queue, .messages, rset.ToJSON())
// if tSC is an error, handle error scenario
```

%ListOfObjects のインスタンスである messages に、トピックまたはキューから取得したメッセージが保存されます。

1.5 トピックまたはキューの削除

アプリケーションでトピックやキューを必要としなくなった時点で、クライアント・オブジェクトの DeleteTopicOrQueue() メソッドをコードで使用してトピックまたはキューを安全に削除できます。

ObjectScript

```
Do client.DeleteQueueOrTopic(topic)
```

一部のメッセージング・プラットフォームには、高度な機能を含む可能性のあるトピック/キューを削除するための独自のメソッドがあることに注意してください。

1.6 メッセージ・プラットフォームからの切断

メッセージ・プラットフォームとの通信を終了した時点で、コードからクライアント・オブジェクトの Close() メソッドを呼び出します。例えば以下のようにします。

ObjectScript

```
Do:client'="" client.Close()
```

2

JMS メッセージング API の使用法

インターシステムズでは、Java Message Service (JMS) によるメッセージの作成と利用に使用できる API を用意しています。コードは、クライアントを作成した後、メッセージの送信または受信のアクションを実行する、そのクライアントのメソッドを呼び出すことにより、メッセージのプロデューサまたはコンシューマとして機能します。InterSystems IRIS も、JMS のキューとトピックを作成するメソッドと削除するメソッドを提供しています。

JMS API は、他のメッセージング・プラットフォームと共有する共通のメッセージング・クラスを基本としています。このページでは、このような共通のクラスで確立されているワークフローに対し、プラットフォーム固有で発生するバリエーションについて説明します。

インターシステムズでは、ここで説明している API のほか、相互運用プロダクションの過程で JMS へのメッセージ送信と JMS からのメッセージ取得に使用できる専用のクラスを用意しています。

2.1 JMS への接続

JMS アプリケーションとの接続を作成するには以下の手順に従います。

1. 設定オブジェクトを作成します。そのためには、%External.Messaging.JMSSettings のインスタンスを作成し、必要に応じてそのプロパティを設定します。

   ObjectScript

   ```
   Set settings = ##class(%External.Messaging.JMSSettings).%New()
   Set settings.url = "messaging.bazco.com"
   Set settings.connectionFactoryName = "connectionFactory"
   Set settings.initialContextFactoryName = "eventContextFactory"
   Set settings.username = "briannawaller"
   Set settings.password = "824yvpi"
   ```

   使用できるすべてのプロパティは、"JMSSettings クラス・リファレンス" のページを参照してください。

2. メッセージング・クライアント・オブジェクトを作成します。そのためには、汎用 %External.Messaging.Client クラスの CreateClient() メソッドを呼び出し、設定オブジェクトを 1 番目の引数として渡します。例えば以下のようにします。

   ObjectScript

   ```
   Set client = ##class(%External.Messaging.Client).CreateClient(settings, .tSC)
   If $$$ISERR(tSC) {
         //handle error scenario
   }
   ```

   このメソッドからは、参照渡しでステータス・コードが 2 番目の引数として返されます。コードでは、このステータスを確認したうえで処理を継続する必要があります。

settings オブジェクトは %External.Messaging.JMSSettings のインスタンスなので、返されるオブジェクト (client) は %External.Messaging.JMSClient のインスタンスです。

2.2 JMS プロデューサ

InterSystems IRIS は、メッセージを作成する API メソッドを呼び出し、作成したメッセージを送信することにより、JMS アプリケーション内部でプロデューサとして機能できます。メッセージをルーティングするキューまたはトピックを作成する場合は、"キューとトピックの操作" を参照してください。以下のフローでは、クライアント・オブジェクトを使用し、プロデューサとして JMS アプリケーション を操作しています。

メッセージ・プロパティの設定 (オプション)

JMS 仕様では、指定のメッセージ・ヘッダとカスタムのメッセージ・プロパティの両方を使用して、メッセージにさまざまなメタデータを添付できます。JMS メッセージのプロパティを作成して設定するには、JMS メッセージ・プロパティ・オブジェクトの %ListOfObjects コレクションを作成する必要があります。次のセクションで説明する JMS メッセージ・オブジェクトでは、このコレクション (一覧) をオプションのプロパティとして受け入れます。メッセージ・プロパティごとに以下を定義します。

1. %External.Messaging.JMSMessageProperty オブジェクトの新しいインスタンスを作成します。
2. 次のように、このメッセージ・プロパティ・オブジェクトのプロパティを設定します。
   - key :メッセージ・プロパティ・キー
   - type :メッセージ・プロパティ値のデータ型
   - value :メッセージ・プロパティ値の文字列表現
3. メッセージ・プロパティ・オブジェクトの一覧に、このメッセージ・プロパティ・オブジェクトを追加します。

メッセージ・プロパティの詳細は、JMS ドキュメントを参照してください。以下の例では、カスタムのタイム・スタンプ・プロパティを作成しています。

ObjectScript

```
Set propList = ##class(%ListOfObjects).%New()

Set key = "odbcUtcTimestamp"
Set type = "String"
Set value = $zdatetime($horolog, 3, 8)

Set msgProp1 = ##class(%External.Messaging.JMSMessageProperty).%New()
Set msgProp1.key = key
Set msgProp1.type = type
Set msg.value = value

Set tSC = propList.Insert(msgProp1)
```

メッセージの作成

送信するメッセージを作成するには、%External.Messaging.JMSMessage オブジェクトの新しいインスタンスを作成します。つづいて、そのメッセージ・オブジェクトのプロパティを定義します。メッセージの destination の名前と type ("Text" または "Bytes") を指定する必要があります。メッセージ・タイプに応じ、textBody プロパティまたは bytesBody プロパティを使用してメッセージ本文を設定します。前のセクションの説明に従ってメッセージ・プロパティ・オブジェクトの一覧を作成している場合は、その一覧を properties プロパティとして指定します。

ObjectScript

```
Set destination = "quick-start-events-queue"
Set type = "Text"
Set textBody = "MyMessage"

Set msg = ##class(%External.Messaging.JMSMessage).%New()
Set msg.destination = destination
Set msg.type = type
Set msg.textBody = textBody
Set msg.properties = propList
```

メッセージの送信

作成したメッセージは、JMS クライアント・オブジェクトの SendMessage() メソッドを実行することにより、送信先のキューまたはトピックに送信できます。例えば以下のようにします。

ObjectScript

```
set tSC = client.SendMessage(msg)
if $$$ISERR(tSC) {
      //handle error scenario
}
```

2.3 JMS コンシューマ

InterSystems IRIS は、トピックのメッセージを取得する API メソッドを呼び出すことにより、JMS アプリケーション内部でコンシューマとして機能できます。以下のフローでは、クライアントを使用し、コンシューマとして JMS アプリケーションを操作しています。

メッセージ取得のための設定の構成 (オプション)

JMS クライアント・オブジェクトは、ReceiveMessage() メソッドを使用して JMS メッセージのコンシューマとして機能できます。このメソッドでは、JSON 形式の文字列をオプションの引数として指定することで、メッセージ取得操作の設定を指定できます。そのためには、%External.Messaging.JMSReceiveSettings クラスの新しいインスタンスを作成し、必要に応じてそのプロパティを設定します。利用できるプロパティは以下のとおりです。

- receiveTimeout : 取得操作がタイムアウトするまでの時間 (ミリ秒) を指定する整数。設定しない場合の既定値は 100 です。
- subscriber (オプション) : クライアントを特定するサブスクライバ名を記述した文字列

例えば以下のようにします。

ObjectScript

```
Set rset = ##class(%External.Messaging.JMSReceiveSettings).%New()
Set rset.receiveTimeout = 200
```

メッセージの取得

メッセージを取得するには、JMS クライアント・オブジェクトが継承した ReceiveMessage() メソッドを呼び出します。このメソッドは、キューまたはトピックの名前を引数として取り、参照渡しで %ListOfObjects としてメッセージを返します。前のセクションで説明したメッセージ取得設定を指定している場合は、設定オブジェクトの ToJSON() メソッドを使用して、その設定を 3 番目の引数として指定します。

ObjectScript

```
#dim messages As %ListOfObjects
Set tSC = client.ReceiveMessage(queue, .messages, rset.ToJSON())
```

2.4 キューとトピックの操作

InterSystems IRIS には、JMS メッセージの送信先を管理する API が用意されています。ポイントツーポイントのメッセージングでは、キューを作成または削除できます。パブリッシャツーパブリッシャのメッセージングでは、トピックを作成または削除できます。

2.4.1 キューの作成または削除

キューの作成

JMS クライアント・オブジェクトは、キューを作成するための CreateQueue() メソッドを備えています。CreateQueue() は、キューの名前を引数として受け入れます。

ObjectScript

```
Set queueName = "quick-start-events"
Set tSC = client.CreateQueue(queueName)
```

また、すべてのメッセージング・プラットフォームに共通のメソッドを使用して、キューを作成することもできます。詳細は、"%External.Messaging.Client.CreateQueueOrTopic()" を参照してください。

キューの削除

この API でサポートされているすべてのメッセージング・プラットフォームのクライアント・オブジェクトに共通のメソッドを使用して、JMS キューを削除できます。詳細は、"%External.Messaging.Client.DeleteQueueOrTopic()" を参照してください。

ObjectScript

```
Set tSC = client.DeleteQueueOrTopic(queueName)
```

2.4.2 トピックの作成または削除

トピックの作成

JMS クライアント・オブジェクトは、トピックを作成するための CreateTopic() メソッドを備えています。CreateTopic() は、トピックの名前を引数として受け入れます。

ObjectScript

```
Set topicName = "alerts_urgent"
Set tSC = client.CreateTopic(topicName)
```

また、すべてのメッセージング・プラットフォームに共通のメソッドを使用してトピックを作成することもできます。詳細は、"%External.Messaging.Client.CreateQueueOrTopic()" を参照してください。

トピックの削除

この API でサポートされているすべてのメッセージング・プラットフォームのクライアント・オブジェクトに共通のメソッドを使用して、JMS トピックを削除できます。詳細は、"%External.Messaging.Client.DeleteQueueOrTopic()" を参照してください。

ObjectScript

```
Set tSC = client.DeleteQueueOrTopic(topicName)
```

2.5 クライアントの終了

JMS アプリケーションとの通信を完了した InterSystems IRIS アプリケーションは、クライアント・オブジェクトの Close() メソッドを使用してクライアントを閉じる必要があります。例えば以下のようにします。

ObjectScript

```
Do:client'="" client.Close()
```

3

Kafka メッセージング API の使用法

インターシステムズでは、Kafka メッセージの作成と利用に使用できる API を用意しています。コードは、クライアントを作成した後、メッセージの送信または受信のアクションを実行する、そのクライアントのメソッドを呼び出すことにより、メッセージのプロデューサまたはコンシューマとして機能します。InterSystems IRIS も、Kafka のトピックを作成するメソッドと削除するメソッドを提供しています。

Kafka API は、他のメッセージング・プラットフォームと共有する共通のメッセージング・クラスを基本としています。このページでは、このような共通のクラスで確立されているフローに対し、プラットフォーム固有で発生するバリエーションについて説明します。

インターシステムズでは、ここで説明する API のほか、相互運用プロダクションに使用できる専用のクラスを用意しています。このようなクラスでは、プロダクションで Kafka へメッセージを送信でき、また Kafka からメッセージを取得できます。

3.1 Kafka との接続

Kafka との接続を作成するには以下の手順に従います。

1. 設定オブジェクトを作成します。そのためには、%External.Messaging.KafkaSettings のインスタンスを作成し、必要に応じてそのプロパティを設定します。以下に挙げるほとんどのプロパティは、プロデューサとコンシューマの両方に適用できますが、groupId 設定は例外で、コンシューマ・グループへのコンシューマの割り当てにのみ使用します。
   - username と password でクライアントの Kafka 認証情報を定義します。
   - servers は、Kafka ブローカ・クラスタ内のサーバを指定する、コンマ区切りの IP address:port エントリのリストを定義します。
   - 必要に応じて、Kafka プロデューサまたは Kafka コンシューマのクライアント ID を clientId で定義します。
   - groupId で、コンシューマのコンシューマ・グループ ID を定義します。
   - securityprotocol は、Kafka ブローカ・クラスタへの接続を保護するセキュリティ・プロトコルを指定します。現在、このプロパティは以下の 2 つの値をサポートしています。
     - SASL_PLAINTEXT。これは、暗号化されていないチャンネルでクライアントの SASL 認証を実行します。
     - SASL_SSL。指定したトラストストアとキーストアの情報を使用して、SASL 認証が行われる SSL/TLS 接続を確立します。
   - saslmechanism は、クライアントの認証に使用される SASL 認証メカニズムを指定します。現在、PLAIN のみがサポートされています。

- truststorelocation (オプション) は、Kafka ブローカ・クラスタからの証明書を検証し、SSL/TLS 接続を確立するために必要な認証局の証明書を含むトラストストアへのファイル・システム・パスを指定します。
- truststorepassword (オプション) は、truststorelocation で指定された場所にあるトラストストアへのアクセスを提供するパスワードを定義します。
- keystorelocation (オプション) は、Kafka ブローカ・クラスタとの SSL/TLS 接続を確立するために必要なキーを含むキーストアへのファイル・システム・パスを指定します。
- keystorepassword (オプション) は、keystorelocation で指定された場所にあるキーストアへのアクセスを提供するパスワードを定義します。
- keypassword (オプション) は、keystorelocation で指定された場所にあるキーストア内の秘密鍵へのアクセスを提供するパスワードを定義します。

例えば以下のようにします。

ObjectScript

```
Set settings = ##class(%External.Messaging.KafkaSettings).%New()
Set settings.username = "amandasmith"
Set settings.password = "234sdsge"
Set settings.servers = "100.0.70.179:9092, 100.0.70.089:7070"
Set settings.clientId = "BazcoApp"
// If Consumer, specify a consumer group
Set settings.groupId = "G1"
Set settings.securityprotocol="SASL_SSL"
Set settings.saslmechanism="PLAIN"
Set settings.truststorelocation="/etc/kafkatls/trustcerts.jks"
Set settings.truststorepassword="F00B&r!"
```

2. メッセージング・クライアント・オブジェクトを作成します。そのためには、%External.Messaging.Client の CreateClient() メソッドを呼び出し、設定オブジェクトを 1 番目の引数として渡します。例えば以下のようにします。

ObjectScript

```
Set client = ##class(%External.Messaging.Client).CreateClient(settings, .tSC)
// if tSC is an error, handle error scenario
```

このメソッドからは、参照渡しでステータス・コードが 2 番目の引数として返されます。コードでは、このステータスを確認したうえで処理を継続する必要があります。

settings オブジェクトは %External.Messaging.KafkaSettings のインスタンスであるため、メソッドは client に対する %External.Messaging.KafkaClient のインスタンスを返します。

3.2 Kafka プロデューサ

InterSystems IRIS は、メッセージを作成する API メソッドを呼び出し、作成したメッセージを送信することにより、Kafka プロデューサとして機能できます。メッセージの送信先とするトピックをアプリケーションで作成する必要がある場合は、"トピックの操作" を参照してください。以下のフローでは、クライアントを使用し、プロデューサとして Kafka を操作しています。

プロデューサとしてのクライアントの構成 (オプション)

Kafka クライアントの作成後、メッセージを送信する前に、標準の Apache ProducerConfig 構成オプションをアプリケーションで使用して、プロデューサをカスタマイズできます。既定では、クライアントは標準のプロデューサとして構成されますが、Apache オプションを JSON 文字列として UpdateProducerConfig() メソッドに渡すことで、この既定の構成を変更できます。サポートされているオプションの一覧は、Apache Kafka のドキュメントを参照してください。例えば、次のコードは、Apache 構成オプションを使用して、Kafka クライアントを構成します。

ObjectScript

```
 Set client = ##class(%External.Messaging.Client).CreateClient(kafkaSettings, .tSC)

 Set producerSettings =
"{""key.serializer"":""org.apache.kafka.common.serialization.StringSerializer""}"
 Set tSC = client.UpdateProducerConfig(producerSettings)
```

メッセージの作成

Kafka に送信するメッセージには、**topic** と **value** を記述する必要があります。また、値のタグとして機能する **key** を必要に応じて記述できます。トピックに送信するメッセージを作成するには、新しい Kafka メッセージ・オブジェクトを作成し、そこで上記のプロパティを定義します。例えば以下のようにします。

ObjectScript

```
 Set topic = "quick-start-events"
 Set value = "MyMessage", key = "OptionalTag"

 Set msg = ##class(%External.Messaging.KafkaMessage).%New()
 Set msg.topic = topic
 Set msg.value = value
 Set msg.key = key
```

InterSystems IRIS の %String で指定されている値を超える長さのメッセージを交換できる Kafka 構成をサポートするには、**value** プロパティではなく、**binaryValue** プロパティを使用し、適切にエンコードしたバイナリ・ストリームとしてメッセージ・コンテンツを保存します。binaryValue プロパティの値は任意の長さにすることができます。

メッセージの送信

作成したメッセージは、SendMessage() メソッドを実行することによってトピックに送信できます。例えば以下のようにします。

ObjectScript

```
 Set tSC = client.SendMessage(msg)
```

3.3 Kafka コンシューマ

InterSystems IRIS は、トピックのメッセージを取得する API を呼び出すことで、Kafka コンシューマとして機能できます。コンシューマのコンシューマ・グループを指定するために、クライアントの設定を定義するときに **groupId** プロパティを必ず定義します。以下のフローでは、クライアントを使用し、コンシューマとして Kafka を操作しています。

コンシューマとしてのクライアントの構成 (オプション)

Kafka クライアントの作成後、メッセージを取得する前に、標準の Apache ConsumerConfig 構成オプションをアプリケーションで使用して、コンシューマをカスタマイズできます。既定では、クライアントは標準のコンシューマとして構成されますが、Apache オプションを JSON 文字列として UpdateConsumerConfig() メソッドに渡すことで、この既定の構成を変更できます。サポートされているオプションの一覧は、Apache Kafka のドキュメントを参照してください。例えば、次のコードは、Apache 構成オプションを使用して、Kafka クライアントを構成します。

ObjectScript

```
 Set client = ##class(%External.Messaging.Client).CreateClient(kafkaSettings, .tSC)

 Set consumerSettings =
"{""key.deserializer"":""org.apache.kafka.common.serialization.StringDeserializer""}"
 Set tSC = client.UpdateConsumerConfig(consumerSettings)
```

メッセージ取得のための設定の構成 (オプション)

Kafka クライアントは、ReceiveMessage() メソッドを使用して Kafka コンシューマとして機能できます。このメソッドでは、JSON 形式の文字列をオプションの引数として指定することで、メッセージの取得をポーリングする際のタイムアウト (ミリ秒) を設定できます。そのためには、次のように %External.Messaging.KafkaReceiveSettings クラスの新しいインスタンスを作成し、pollTimeout プロパティを設定します。

ObjectScript

```
 Set rset = ##class(%External.Messaging.KafkaReceiveSettings).%New()
 Set rset.pollTimeout = 500
```

このように設定しない場合、ポーリングのタイムアウト時間は既定値の 100 ミリ秒になります。

メッセージの取得

メッセージを取得するには、Kafka クライアント ReceiveMessage() メソッドを呼び出します。このメソッドは、トピックの名前を引数として取り、参照渡しで %ListOfObjects としてメッセージを返します。前のセクションで説明したように KafkaReceiveSettings オブジェクトを使用してメッセージ取得設定を指定している場合は、設定オブジェクトの ToJSON() メソッドを使用して、その設定を 3 番目の引数として指定します。Kafka からは、コンシューマがまだ取得していない新しいメッセージのみが返されます。例えば、トピックのメッセージを取得し、表示する場合、次のようになります。

ObjectScript

```
 #dim messages As %ListOfObjects
 Set tSC = client.ReceiveMessage(topic, .messages, rset.ToJSON())

 For i=1:1:messages.Size {
        Set msg = messages.GetAt(i)
        Write "Message: ", msg.ToJSON(), !
 }
```

InterSystems IRIS の %String に指定された値を超える長さのメッセージを処理できるように、Kafka のメッセージ・クラスには、適切にエンコードしたバイナリ・ストリームとして任意の長さのメッセージを保存できる binaryValue プロパティも用意されています。

3.4 AdminClient Configs の定義

アプリケーションでは、標準の Apache AdminClient Configs を使用して Kafka クライアントをカスタマイズできます。クライアントは既定で標準の構成になりますが、Apache オプションを JSON 文字列として UpdateAdminConfig() メソッドに渡すことで、この既定の構成を変更できます。サポートされているオプションの一覧は、Apache Kafka のドキュメントを参照してください。例えば以下のようにします。

ObjectScript

```
 Set client = ##class(%External.Messaging.Client).CreateClient(kafkaSettings, .tSC)

 Set adminSettings = "{""ssl.protocol"":""TLSv1.3""}"
 Set tSC = client.UpdateAdminConfig(adminSettings)
```

3.5 トピックの操作

InterSystems IRIS は、新しい Kafka トピック、およびトピックを削除する別のトピックを作成するために使用できる API を提供します。トピックを作成するときは、パーティション数とレプリケーション・ファクタが引数として渡されます。パーティションとレプリケーション・ファクタの概要は、Apache Kafka のドキュメントを参照してください。トピックを作成するには以下のよう指定します。

ObjectScript

```
Set client = ##class(%External.Messaging.Client).CreateClient(kafkaSettings, .tSC)

Set topic = "quick-start-events"
Set numberOfPartitions = 1
Set replicationFactor = 3
Set tSC = client.CreateTopic(topic,numberOfPartitions,replicationFactor)
```

代わりに、すべてのメッセージング・プラットフォームに共通するメソッドを使用して、トピックを作成することもできます。この代替手法を使用する場合は、**%External.Messaging.KafkaTopicSettings** クラスのインスタンスにトピックの設定を定義し、ToJSON() メソッドを使用して、その設定を JSON オブジェクトとして共通メソッドに渡します。詳細は、"%External.Messaging.Client.CreateQueueOrTopic()" を参照してください。

3.5.1 トピックの削除

アプリケーションで DeleteTopic() メソッドを使用して Kafka トピックを削除できます。

ObjectScript

```
Set tSC = client.DeleteTopic(topic)
```

代わりに、すべてのメッセージング・プラットフォームに共通するメソッドを使用して、トピックを削除することもできます。詳細は、"%External.Messaging.Client.DeleteQueueOrTopic()" を参照してください。

3.6 クライアントの終了

Kafka との通信を完了した InterSystems IRIS アプリケーションは、Close() メソッドを使用してクライアントを閉じる必要があります。例えば以下のようにします。

ObjectScript

```
Do:client'="" client.Close()
```

4

RabbitMQ メッセージング API の使用法

インターシステムズでは、RabbitMQ メッセージの公開と利用に使用できる API を用意しています。コードは、クライアントを作成した後、メッセージの送信または受信のアクションを実行する、そのクライアントのメソッドを呼び出すことにより、メッセージのパブリッシャまたはコンシューマとして機能します。InterSystems IRIS も、RabbitMQ のキューとエクスチェンジを管理するメソッドと、これらの間にバインディングを設定するメソッドを用意しています。

RabbitMQ API は、他のメッセージング・プラットフォームと共有する共通のメッセージング・クラスを基本としています。このページでは、このような共通のクラスで確立されているワークフローに対し、プラットフォーム固有で発生するバリエーションについて説明します。

インターシステムズでは、ここで説明している API のほか、相互運用プロダクションの過程で RabbitMQ へのメッセージの送信と RabbitMQ からのメッセージの取得に使用できる専用のクラスを用意しています。

注釈　インターシステムズの RabbitMQ API は、AMQP 0-9-1 プロトコルの RabbitMQ 実装に相当します。AMQP 1.0 プロトコルを使用している外部メッセージング・プラットフォームでメッセージを送受信するには、Apache Qpid JMS などのプロトコルの JMS 実装に接続することをお勧めします。インターシステムズでは、JMS アプリケーションを操作できるように JMS API を用意しています。

4.1 RabbitMQ との接続

RabbitMQ に接続するには以下の手順に従います。

1. 設定オブジェクトを作成します。そのためには、%External.Messaging.RabbitMQSettings のインスタンスを作成し、必要に応じてそのプロパティを以下のように設定します。
   - username と password：クライアントの RabbitMQ 認証情報を定義します。これらのプロパティに値を設定しないと、既定で "guest" が指定されます。
   - host：ホスト サーバの名前を定義します。定義しない場合は、既定値の "localhost" になります。
   - port：ホスト サーバ側で使用するポート番号を定義する整数です。既定値は 5672 です。
   - virtualHost：必要に応じて仮想ホストの名前を定義します。

   例えば以下のようにします。

ObjectScript

```
Set settings = ##class(%External.Messaging.RabbitMQSettings).%New()
Set settings.username = "ronaldkellogg"
Set settings.password = "449!ds%t"
Set settings.host = "bazco.com"
Set settings.port = 5693
```

2. (オプション)SSL/TLS を使用して RabbitMQ に接続する場合は、設定の enableSSL プロパティを 1 に設定したうえで、使用している TLS 構成に応じて以下の各プロパティを設定します。
   - tlsVersion :使用している TLS プロトコルのバージョンを指定した文字列。既定値は "TLSv1.2" です。
   - enableHostnameVerification :サーバ証明書に記載されたホスト名とサーバのホスト名が一致することを相手検証プロセスで検証するかどうかを指定するブーリアン値。
   - clientKeyFile :相手検証を実行するようにサーバを構成している場合に、クライアントの秘密鍵ファイルへのパスを指定する文字列。
   - keyPassword :クライアントの鍵ファイルがセキュリティで保護されている場合に、そのファイルへのアクセスで必要なパスワード文字列。
   - keyStoreFile :鍵ストア・ファイルへのパスを指定する文字列。
   - keyStorePassword :クライアントの鍵ストア・ファイルがセキュリティで保護されている場合に、そのファイルへのアクセスで必要なパスワード文字列。

3. メッセージング・クライアント・オブジェクトを作成します。そのためには、汎用 %External.Messaging.Client クラスの CreateClient() メソッドを呼び出し、設定オブジェクトを 1 番目の引数として渡します。例えば以下のようにします。

   ObjectScript

   ```
   Set client = ##class(%External.Messaging.Client).CreateClient(settings, .tSC)
   If $$$ISERR(tSC) { //handle error scenario }
   ```

   このメソッドからは、参照渡しでステータス・コードが 2 番目の引数として返されます。コードでは、このステータスを確認したうえで処理を継続する必要があります。

   settings オブジェクトは %External.Messaging.RabbitMQSettings のインスタンスなので、返されるオブジェクト (client) は %External.Messaging.RabbitMQClient のインスタンスです。

4.2 RabbitMQ パブリッシャ

InterSystems IRIS は、メッセージを作成する API メソッドを呼び出し、作成したメッセージを送信することにより、RabbitMQ パブリッシャとして機能できます。メッセージの送信先とするエクスチェンジ、またはエクスチェンジからのメッセージのルーティング先とするキューをアプリケーションで作成する場合は、"エクスチェンジとキューの操作" を参照してください。以下のフローでは、クライアント・オブジェクトを使用し、パブリッシャとして RabbitMQ を操作しています。

メッセージの作成

送信するメッセージを作成するには、%External.Messaging.RabbitMQMessage オブジェクトの新しいインスタンスを作成します。つづいて、そのメッセージ・オブジェクトのプロパティを必要に応じて定義します。使用できるメッセージすべての説明は、RabbitMQ のドキュメントを参照してください。contentEncoding の値が UTF-8 の場合は SetEncodedContent() メソッドを呼び出し、それ以外の値の場合は SetContent() メソッドを呼び出すことで、メッセージのコンテンツを設定します。例えば以下のようにします。

ObjectScript

```
set exchange = "events_handler"
set routingKey = "quick_start"
set deliveryMode = 2
set body = "MyMessage"

set msg = ##class(%External.Messaging.RabbitMQMessage).%New()
set msg.exchange = exchange
set msg.routingKey = routingKey
set msg.deliveryMode = deliveryMode
do msg.SetEncodedContent(body)
```

メッセージの送信

作成したメッセージは、RabbitMQ クライアント・オブジェクトの SendMessage() メソッドを実行することにより、トピックに送信できます。例えば以下のようにします。

ObjectScript

```
set tSC = client.SendMessage(msg)
if $$$ISERR(tSC) { //handle error scenario }
```

4.3 RabbitMQ コンシューマ

InterSystems IRIS は、キューからメッセージを取得する API を呼び出すことで、RabbitMQ コンシューマとして機能できます。以下のフローでは、クライアント・オブジェクトを使用し、コンシューマとして RabbitMQ を操作しています。

メッセージ取得のための設定の構成 (オプション)

RabbitMQ クライアントは、ReceiveMessage() メソッドを使用して RabbitMQ コンシューマとして機能できます。このメソッドでは、JSON 形式の文字列をオプションの引数として指定することで、メッセージ取得操作の設定を指定できます。そのためには、%External.Messaging.RabbitMQReceiveSettings クラスの新しいインスタンスを作成し、必要に応じてそのプロパティを設定します。利用できるプロパティは以下のとおりです。

- autoAck : ブーリアン値。真の場合、サーバでは、メッセージが配信されたときに、そのメッセージが確認応答済みであると自動的に見なされます。偽の場合、サーバでは、コンシューマからの明示的な手動の確認応答があることが想定されます。このプロパティ値を設定していない場合、autoAck には既定値である `false` が設定されます。
- ackMultiple : ブーリアン値。真の場合、すべてのメッセージのバッチについて、手動による確認応答が実行されます。この確認応答で指定した配送タグに相当するメッセージも、このバッチの対象となります。偽の場合、確認応答で指定した配送タグに相当するメッセージのみについて、確認応答が実行されます。このプロパティ値を設定していない場合、ackMultiple には既定値である `false` が設定されます。

例えば以下のようにします。

ObjectScript

```
Set rset = ##class(%External.Messaging.RabbitMQReceiveSettings).%New()
Set rset.autoAck = 0
```

メッセージの取得

メッセージを取得するには、RabbitMQ クライアントが継承した ReceiveMessage() メソッドを呼び出します。このメソッドは、キューの名前を引数として取り、参照渡しで %ListOfObjects としてメッセージを返します。前のセクションで説明したメッセージ取得設定を指定している場合は、ToJSON() メソッドを使用して、その設定を 3 番目の引数として指定します。例えば以下のようにします。

ObjectScript

```
#dim messages As %ListOfObjects
Set tSC = client.ReceiveMessage(queue, .messages, rset.ToJSON())

For i=1:1:messages.Size {
       Set msg = messages.GetAt(i)
       Write "Message: ", msg.ToJSON(), !
}
```

4.4 エクスチェンジとキューの操作

InterSystems IRIS には、RabbitMQ のエクスチェンジとキューを管理するための API が用意されています。これには以下が含まれます。

- エクスチェンジの作成または削除
- キューの作成または削除
- エクスチェンジへのキューの結合

ここでは、この API を使用して上記のタスクを実行する方法について説明します。エクスチェンジとキューの詳しい説明は、RabbitMQ のドキュメントで AMQP 0-9-0 プロトコルの概要を参照してください。

4.4.1 エクスチェンジの作成または削除

AMQP 0-9-0 の仕様によれば、すべての RabbitMQ メッセージはエクスチェンジに送信する必要があり、そこから本来の送信先キュー (複数可) へルーティングされます。

エクスチェンジの作成

RabbitMQ クライアント・オブジェクトは、エクスチェンジを作成するための CreateExchange() メソッドを備えています。CreateExchange() は、以下の引数を、ここに記した順序で受け入れます。

1. exchangeName :エクスチェンジに割り当てる名前。
2. exchangeType :AMQP 0-9-1 エクスチェンジの 4 つの種類である `"direct"`、`"fanout"`、`"topic"`、`"headers"`. のいずれかを指定した文字列。
3. durable :ブーリアン値。真であれば、サーバが再起動してもエクスチェンジは存続します。偽であれば、サーバの再起動後はエクスチェンジを再作成する必要があります。
4. autoDelete :ブーリアン値。真であれば、すべてのキューの結合を解除されたエクスチェンジは削除されます。偽であれば、そのようなエクスチェンジは存続します。

例えば以下のようにします。

```
Set exchangeName = "broadcast"
Set exchangeType = "fanout"
Set durable = 1
Set autoDelete = 0

Set tSC = client.CreateExchange(exchangeName, exchangeType, durable, autoDelete)
```

エクスチェンジの削除

アプリケーションで RabbitMQ エクスチェンジの名前を引数に指定して RabbitMQ クライアント・オブジェクトの DeleteExchange() メソッドを呼び出すと、そのエクスチェンジを削除できます。

```
Set tSC = client.DeleteExchange(exchangeName)
```

4.4.2 キューの作成または削除

RabbitMQ コンシューマがサブスクライブしているキューへエクスチェンジによってメッセージがルーティングされると、RabbitMQ コンシューマはそのメッセージを受領します。

キューの作成

キューを作成するには、RabbitMQ クライアント・オブジェクトの CreateQueue() メソッドを呼び出します。CreateQueue() は、以下の引数を、ここに記した順序で受け入れます。

1. queueName :キューに割り当てる名前。
2. durable :ブーリアン値。真であれば、サーバが再起動してもキューは存続します。偽であれば、サーバの再起動後はキューを再作成する必要があります。
3. exclusive :ブーリアン値。真であれば、キューは 1 つの接続でのみ使用され、その接続が閉じると削除されます。
4. autoDelete :ブーリアン値。真であれば、すべてのコンシューマがサブスクライブを終了したキューが削除されます。偽であれば、そのようなキューは存続します。

例えば以下のようにします。

ObjectScript

```
Set queue = "quick-start-events"
Set durable = 1
Set exclusive = 0
Set autoDelete = 0
Set tSC = client.CreateQueue(queue, durable, exclusive, autoDelete)
```

また、すべてのメッセージング・プラットフォームに共通のメソッドを使用して、キューを作成することもできます。この代替手法を使用する場合は、%External.Messaging.RabbitMQQueueSettings クラスのインスタンスにキューの設定を定義し、ToJSON() メソッドを使用して、その設定を JSON オブジェクトとして共通メソッドに渡します。詳細は、"%External.Messaging.Client.CreateQueueOrTopic()" を参照してください。

キューの削除

アプリケーションで RabbitMQ キューの名前を引数に指定して RabbitMQ クライアント・オブジェクトの DeleteQueue() メソッドを呼び出すと、その RabbitMQ キューを削除できます。

```
Set tSC = client.DeleteQueue(queueName)
```

また、すべてのメッセージング・プラットフォームに共通のメソッドを使用して、キューを削除することもできます。詳細は、"%External.Messaging.Client.DeleteQueueOrTopic()" を参照してください。

4.4.3 エクスチェンジへのキューの結合

キューにメッセージをルーティングするには、キューをエクスチェンジに結合する必要があります。アプリケーションで RabbitMQ クライアント・オブジェクトの BindQueue() メソッドを呼び出すことで、エクスチェンジにキューを結合できます。

BindQueue() メソッドでは、キュー名、エクスチェンジ名、および複数の結合キーをコンマ区切りで記述した文字列を引数として受け入れます。結合キーの中で使用しているコンマは、その前に円記号 (\) を記述することでエスケープできます。結合キーの中で使用している円記号は、その前にもう 1 つ円記号を記述してエスケープできます。

例えば、必要な 2 つの結合キーとして "event-log,critical" と "event-log,urgent/important" があるとすると、アプリケーションのコードでは次のようにしてキューを結合できます。

```
Set bindingKeys = "event-log\,critical,event-log\,urgent\\important"
Set tSC = client.BindQueue(queueName, exchangeName, bindingKeys)
```

4.5 クライアントの終了

RabbitMQ との通信を完了した InterSystems IRIS アプリケーションは、Close() メソッドを使用してクライアントを閉じる必要があります。例えば以下のようにします。

ObjectScript

```
Do:client'="" client.Close()
```

5

Amazon SNS メッセージング API の使用法

インターシステムズでは、Amazon Simple Notification Service (SNS) を使用してメッセージを公開するために使用できる API を提供しています。コードは、クライアントを作成した後、メッセージを送信する、そのクライアントのメソッドを呼び出すことにより、パブリッシャとして機能します。InterSystems IRIS も、Amazon SNS トピックを作成するメソッドと削除するメソッドを提供しています。

Amazon SNS API は、他のメッセージング・プラットフォームと共有する共通のメッセージング・クラスを基本としています。このページでは、このような共通のクラスで確立されているワークフローに対し、プラットフォーム固有で発生するバリエーションについて説明します。

インターシステムズでは、ここで説明している API のほか、相互運用プロダクションの過程で Amazon SNS へのメッセージの送信に使用できる専用のクラスを用意しています。

5.1 Amazon SNS への接続

Amazon SNS との接続を作成するには以下の手順に従います。

1. 設定オブジェクトを作成します。そのためには、%External.Messaging.SNSSettings のインスタンスを作成し、そのプロパティを以下のように設定します。
   - credentialsFile : Amazon Simple Storage Service (S3) の認証情報ファイルが保存されている場所を指定した文字列。
   - accessKey : 使用している Amazon S3 アクセス キーを記述した文字列。credentialsFile を指定していれば、このプロパティの設定は不要です。
   - secretKey : 使用している Amazon S3 秘密鍵を記述した文字列。credentialsFile を指定していれば、このプロパティの設定は不要です。
   - sessionToken : 使用している Amazon S3 セッション・トークンを記述した文字列。セッション・トークンを記述した credentialsFile を指定していれば、このプロパティの設定は不要です。
   - region : Amazon S3 のリージョンを指定した文字列。

   例えば以下のようにします。

   ObjectScript

   ```
   Set settings = ##class(%External.Messaging.SNSSettings).%New()
   Set settings.credentialsFile = "~/.aws/credentials/cred.ini"
   Set settings.region = "us-east-1"
   ```

2. メッセージング・クライアント・オブジェクトを作成します。そのためには、汎用 %External.Messaging.Client クラスの CreateClient() メソッドを呼び出し、設定オブジェクトを 1 番目の引数として渡します。例えば以下のようにします。

```
ObjectScript

 Set client = ##class(%External.Messaging.Client).CreateClient(settings, .tSC)
 // If tSC is an error, handle error scenario
```

このメソッドからは、参照渡しでステータス・コードが 2 番目の引数として返されます。コードでは、このステータスを確認したうえで処理を継続する必要があります。

settings オブジェクトは %External.Messaging.SNSSettings のインスタンスなので、返されるオブジェクト (client) は %External.Messaging.SNSClient のインスタンスです。

5.2 Amazon SNS パブリッシャ

InterSystems IRIS は、メッセージを作成する API メソッドを呼び出し、作成したメッセージを送信することにより、Amazon SNS パブリッシャとして機能できます。メッセージの送信先とするトピックをアプリケーションで作成する必要がある場合は、"トピックの操作" を参照してください。以下のフローでは、クライアント・オブジェクトを使用し、パブリッシャとして Amazon SNS を操作しています。

メッセージの作成

送信するメッセージを作成するには、%External.Messaging.SNSMessage オブジェクトの新しいインスタンスを作成します。つづいて、そのメッセージ・オブジェクトのプロパティを定義します。メッセージの送信先とするトピックの Amazon Resource Name (ARN) を topicARN プロパティで指定し、メッセージ本文を message プロパティで指定する必要があります。このメッセージには、必要に応じて subject を指定することもできます。

```
ObjectScript

 Set topicARN = "arn:aws:sns:us-east-1:123456789012:quick-start-events"
 Set message = "MyMessage"
 Set subject = "EventNotification"

 Set msg = ##class(%External.Messaging.SNSMessage).%New()
 Set msg.topicARN = topicARN
 Set msg.message = message
 Set msg.subject = subject
```

メッセージの送信

作成したメッセージは、Amazon SNS クライアント・オブジェクトの SendMessage() メソッドを実行することにより、トピックに送信できます。例えば以下のようにします。

```
ObjectScript

 set tSC = client.SendMessage(msg)
 if $$$ISERR(tSC) {
       //handle error scenario
 }
```

5.3 トピックの操作

InterSystems IRIS は、Amazon SNS トピックの作成と削除に使用できる API を提供します。

トピックの作成

トピックを作成するには、クライアント・オブジェクトの CreateTopic() メソッドを呼び出します。このメソッドは、トピック名を引数として受け入れ、そのトピックの ARN を参照渡しで返します。例えば以下のようにします。

ObjectScript

```
Set topicName = "quick-start-events"
Set topicARN = ""
Set tSC = client.CreateQueue(topicName, .topicARN)
```

また、すべてのメッセージング・プラットフォームに共通の %External.Messaging.Client.CreateQueueOrTopic() メソッドを使用してトピックを作成することもできます。ただし、この一般的な方法では新しいトピックの ARN が返されません。トピックを削除する API メソッドとメッセージを送信する API メソッドでは、この ARN が必要です。CreateQueueOrTopic() を使用して作成したトピックで、API を使用してこれらのタスクを実行するには、そのトピックの ARN を手動で取得する必要があります。

トピックの削除

アプリケーションで Amazon SNS トピックの ARN を引数に指定してクライアント・オブジェクトの DeleteTopic() メソッドを呼び出すと、その Amazon SNS トピックを削除できます。

ObjectScript

```
Set tSC = client.DeleteTopic(topicARN)
```

代わりに、すべてのメッセージング・プラットフォームに共通するメソッドを使用して、トピックを削除することもできます。詳細は、"%External.Messaging.Client.DeleteQueueOrTopic()" を参照してください。

5.4 クライアントの終了

Amazon SNS との通信を完了した InterSystems IRIS アプリケーションは、Close() メソッドを使用してクライアントを閉じる必要があります。例えば以下のようにします。

ObjectScript

```
Do:client'="" client.Close()
```

6

Amazon SQS メッセージング API の使用法

インターシステムズでは、Amazon Simple Queue Service (SQS) によるメッセージの作成と利用に使用できる API を用意しています。コードは、クライアントを作成した後、メッセージの送信または受信のアクションを実行する、そのクライアントのメソッドを呼び出すことにより、メッセージのプロデューサまたはコンシューマとして機能します。InterSystems IRIS も、Amazon SQS のキューを作成するメソッドと削除するメソッドを提供しています。

Amazon SQS API は、他のメッセージング・プラットフォームと共有する共通のメッセージング・クラスを基本としています。このページでは、このような共通のクラスで確立されているワークフローに対し、プラットフォーム固有で発生するバリエーションについて説明します。

インターシステムズでは、ここで説明している API のほか、相互運用プロダクションの過程で Amazon SQS へのメッセージ送信と Amazon SQS からのメッセージ取得に使用できる専用のクラスを用意しています。

6.1 Amazon SQS への接続

Amazon SQS との接続を作成するには以下の手順に従います。

1. 設定オブジェクトを作成します。そのためには、%External.Messaging.SQSSettings のインスタンスを作成し、そのプロパティを以下のように設定します。
   - credentialsFile : Amazon Simple Storage Service (S3) の認証情報ファイルが保存されている場所を指定した文字列。
   - accessKey : 使用している Amazon S3 アクセス キーを記述した文字列。credentialsFile を指定していれば、このプロパティの設定は不要です。
   - secretKey : 使用している Amazon S3 秘密鍵を記述した文字列。credentialsFile を指定していれば、このプロパティの設定は不要です。
   - sessionToken : 使用している Amazon S3 セッション・トークンを記述した文字列。セッション・トークンを記述した credentialsFile を指定していれば、このプロパティの設定は不要です。
   - region : Amazon S3 のリージョンを指定した文字列。

   例えば以下のようにします。

   ObjectScript

   ```
   Set settings = ##class(%External.Messaging.SQSSettings).%New()
   Set settings.credentialsFile = "~/.aws/credentials/cred.ini"
   Set settings.region = "us-east-1"
   ```

2. メッセージング・クライアント・オブジェクトを作成します。そのためには、汎用 %External.Messaging.Client クラスの CreateClient() メソッドを呼び出し、設定オブジェクトを 1 番目の引数として渡します。例えば以下のようにします。

```
ObjectScript

Set client = ##class(%External.Messaging.Client).CreateClient(settings, .tSC)
If $$$ISERR(tSC) {
      //handle error scenario
}
```

このメソッドからは、参照渡しでステータス・コードが 2 番目の引数として返されます。コードでは、このステータスを確認したうえで処理を継続する必要があります。

settings オブジェクトは %External.Messaging.SQSSettings のインスタンスなので、返されるオブジェクト (client) は %External.Messaging.SQSClient のインスタンスです。

6.2 Amazon SQS プロデューサ

InterSystems IRIS は、メッセージを作成する API メソッドを呼び出し、作成したメッセージを送信することにより、Amazon SQS パブリッシャとして機能できます。メッセージの送信先とするトピックをアプリケーションで作成する必要がある場合は、"キューの操作" を参照してください。以下のフローでは、クライアント・オブジェクトを使用し、パブリッシャとして Amazon SQS を操作しています。

メッセージ属性の設定 (オプション)

Amazon SQS メッセージ属性を使用してメッセージにカスタム・メタデータを添付するには、SQS メッセージ属性オブジェクトの InterSystems IRIS %ListOfObjects コレクションを作成する必要があります。次のセクションで作成する Amazon SQS メッセージ・オブジェクトでは、この属性のコレクション (一覧) をオプションのプロパティとして受け入れます。属性ごとに以下を定義します。

1. %External.Messaging.SQSMessageAttribute オブジェクトの新しいインスタンスを作成します。
2. 次のように、このメッセージ属性オブジェクトのプロパティを設定します。
   - key :メッセージ属性キー
   - dataType :メッセージ属性値のデータ型 ("String"、"Number"、または "Binary")
   - stringValue または binaryValue :メッセージ属性値。指定したデータ型に適切な方のプロパティを設定します。
3. メッセージ属性オブジェクトの一覧に、各メッセージ属性オブジェクトを追加します。

メッセージ属性の詳細は、Amazon SQS メッセージ・メタデータのドキュメントを参照してください。以下の例ではタイム・スタンプ属性を作成しています。

```
ObjectScript

Set attrList = ##class(%ListOfObjects).%New()

Set key = "timestamp"
Set dataType = "String"
Set value = $zdatetime($horolog)

Set msgAttr1 = ##class(%External.Messaging.SQSMessageAttribute).%New()
Set msgAttr1.key = key
Set msgAttr1.dataType = dataType
Set msg.stringValue = value

Set tSC = attrList.Insert(msgAttr1)
```

メッセージの作成

送信するメッセージを作成するには、%External.Messaging.SQSMessage オブジェクトの新しいインスタンスを作成します。つづいて、そのメッセージ・オブジェクトのプロパティを定義します。メッセージの送信先名である queue とメッセージの body を指定する必要があります。前のセクションで説明したカスタム・メッセージ属性を定義している場合は、messageAttributes プロパティとしてメッセージ属性オブジェクトの一覧を指定します。例えば以下のようにします。

ObjectScript

```
Set queue = "quick-start-events"
Set body = "MyMessage"

Set msg = ##class(%External.Messaging.SQSMessage).%New()
Set msg.queue = queue
Set msg.body = body
Set msg.messageAttributes = attrList
```

メッセージの送信

作成したメッセージは、Amazon SQS クライアント・オブジェクトの SendMessage() メソッドを実行することにより、トピックに送信できます:

ObjectScript

```
set tSC = client.SendMessage(msg)
if $$$ISERR(tSC) {
      //handle error scenario
}
```

6.3 Amazon SQS コンシューマ

InterSystems IRIS は、トピックのメッセージを取得する API メソッドを呼び出すことにより、Amazon SQS コンシューマとして機能できます。以下のフローでは、クライアント・オブジェクトを使用し、コンシューマとして Amazon SQS を操作しています。

メッセージ取得のための設定の構成 (オプション)

Amazon SQS クライアントは、ReceiveMessage() メソッドを使用して Amazon SQS コンシューマとして機能できます。このメソッドでは、JSON 形式の文字列をオプションの引数として指定することで、メッセージ取得操作の設定を指定できます。そのためには、%External.Messaging.SQSReceiveSettings クラスの新しいインスタンスを作成し、必要に応じてそのプロパティを設定します。利用できるプロパティは以下のとおりです。

- maxNumberOfMessages :返すメッセージの最大数を指定した整数
- waitTimeSeconds :ポーリングがタイムアウトするまでの時間 (秒数) を指定した整数
- visibilityTimeout :メソッドから返されたメッセージを他のコンシューマが実質的に確認できる時間 (秒数) を指定した整数

例えば以下のようにします。

ObjectScript

```
Set rset = ##class(%External.Messaging.SQSReceiveSettings).%New()
Set rset.waitTimeSeconds = 5
Set rset.visibilityTimeout = 30
```

メッセージの取得

メッセージを取得するには、Amazon SQS クライアント・オブジェクトが継承した ReceiveMessage() メソッドを呼び出します。このメソッドは、キューの名前を引数として取り、参照渡しで %ListOfObjects としてメッセージを返します。前のセクションで説明したメッセージ取得設定を指定している場合は、ToJSON() メソッドを使用して、その設定を 3 番目の引数として指定します。例えば以下のようにします。

ObjectScript

```
#dim messages As %ListOfObjects
Set tSC = client.ReceiveMessage(queue, .messages, rset.ToJSON())
```

キューからのメッセージの削除

キューのメッセージを受信して処理した Amazon SQS コンシューマは、キューからそのメッセージを削除する必要があります。メッセージを削除するには、クライアント・オブジェクトの DeleteMessage() メソッドを呼び出します。DeleteMessage() では、キューの名前を 1 番目の引数、メッセージの受信ハンドルを 2 番目の引数としてそれぞれ指定する必要があります。受信ハンドルは、ReceiveMessage() メソッドから返された、メッセージごとの receiptHandle プロパティに保存されています。

ObjectScript

```
For i=1:1:messages.Size {
	Set msg = messages.GetAt(i)
	Write "Message: ", msg.ToJSON(), !
	Set tSC = client.DeleteMessage(queue, msg.receiptHandle)
}
```

6.4 キューの操作

InterSystems IRIS は、Amazon SQS キューの作成と削除に使用できる API を提供します。

キュー設定の指定 (オプション)

キューの設定を指定する場合は、%External.Messaging.SQSQueueSettings オブジェクトを作成し、そのオブジェクトのプロパティのうち、目的の設定に対応するものを設定します。使用できる構成オプションの詳細は、Amazon SQS のドキュメントを参照してください。

例えば、以下のコードでは、先入れ先出しキューを指定するキュー設定オブジェクトを作成し、そのキューへ 5 秒間の遅延ですべてのメッセージを配信します。

ObjectScript

```
Set queueSet = ##class(%External.Messaging.SQSQueueSettings).%New()
Set queueSet.FifoQueue = 1
Set queueSet.DelaySeconds = 5
```

キューの作成

キューを作成するには、クライアント・オブジェクトの CreateQueue() メソッドを呼び出します。CreateQueue() では、キュー名を引数で指定する必要があります。前のセクションの説明にあるようにキューのキュー設定オブジェクトを作成していれば、そのオブジェクトを 2 番目のオプションの引数として指定できます。

ObjectScript

```
Set queue = "quick-start-events"
Set tSC = client.CreateQueue(queue, queueSet)
```

また、すべてのメッセージング・プラットフォームに共通のメソッドを使用して、キューを作成することもできます。この代替手法を使用する場合は、ToJSON() メソッドを使用して、キュー設定オブジェクトのコンテンツを JSON オブジェクトとして指定できます。詳細は、"%External.Messaging.Client.CreateQueueOrTopic()" を参照してください。

キューの削除

アプリケーションで クライアント・オブジェクトの DeleteTopic() メソッドを呼び出すことで、Amazon SQS キューを削除できます。このメソッドは、キューの名前を引数として受け入れます。

ObjectScript

```
Set tSC = client.DeleteQueue(queue)
```

また、すべてのメッセージング・プラットフォームに共通のメソッドを使用して、キューを削除することもできます。詳細は、"%External.Messaging.Client.DeleteQueueOrTopic()" を参照してください。

6.5 クライアントの終了

Amazon SQS との通信を完了した InterSystems IRIS アプリケーションは、Close() メソッドを使用してクライアントを閉じる必要があります。例えば以下のようにします。

ObjectScript

```
Do:client'="" client.Close()
```