Nimbusパブリッシャー仕様書

2016年6月1日

目次

[1. 概要 1](#_Toc455993277)

[2. インタフェース設計 6](#_Toc455993278)

[2.1. Message 6](#_Toc455993279)

[2.2. ServerConnection 8](#_Toc455993280)

[2.3. ClientConnection 9](#_Toc455993281)

[2.4. RequestServerConnection 10](#_Toc455993282)

[2.5. ServerConnectionFactory 12](#_Toc455993283)

[2.6. ClientConnectionFactory 14](#_Toc455993284)

[2.7. ThinOutFilter 16](#_Toc455993285)

[2.8. MessageReceiver 17](#_Toc455993286)

[3. 実装設計 18](#_Toc455993287)

[3.1. ServerConnectionFactory 18](#_Toc455993288)

[3.1.1. jp.ossc.nimbus.service.publish.tcp.ConnectionFactoryService 18](#_Toc455993289)

[3.1.2. jp.ossc.nimbus.service.publish.udp.ConnectionFactoryService 27](#_Toc455993290)

[3.1.3. jp.ossc.nimbus.service.publish.DistributedConnectionFactoryService 44](#_Toc455993291)

[3.1.4. jp.ossc.nimbus.service.publish.GroupConnectionFactoryService 47](#_Toc455993292)

[3.1.5. jp.ossc.nimbus.service.publish.ThinOutServerConnectionFactoryService 50](#_Toc455993293)

[3.1.6. jp.ossc.nimbus.service.publish.RequestConnectionFactoryService 52](#_Toc455993294)

[3.2. ClientConnectionFactory 54](#_Toc455993295)

[3.2.1. jp.ossc.nimbus.service.publish.tcp.ConnectionFactoryService 54](#_Toc455993296)

[3.2.2. jp.ossc.nimbus.service.publish.udp.ConnectionFactoryService 54](#_Toc455993297)

[3.2.3. jp.ossc.nimbus.service.publish.ClusterConnectionFactoryService 54](#_Toc455993298)

[3.2.4. jp.ossc.nimbus.service.publish.ClusterClientConnectionFactoryService 58](#_Toc455993299)

[3.2.5. jp.ossc.nimbus.service.publish.DistributedConnectionFactoryService 59](#_Toc455993300)

[3.2.6. jp.ossc.nimbus.service.publish.GroupConnectionFactoryService 59](#_Toc455993301)

[3.2.7. jp.ossc.nimbus.service.publish.GroupClientConnectionFactoryService 60](#_Toc455993302)

[3.3. ThinOutFilter 62](#_Toc455993303)

[3.3.1. jp.ossc.nimbus.service.publish.SubjectThinOutFilterService 62](#_Toc455993304)

[3.3.2. jp.ossc.nimbus.service.publish.TimeIntervalThinOutFilterService 62](#_Toc455993305)

[3.4. MessageReceiver 63](#_Toc455993306)

[3.4.1. jp.ossc.nimbus.service.publish.MessageReceiverService 63](#_Toc455993307)

[3.4.2. jp.ossc.nimbus.service.publish.MessageForwardService 65](#_Toc455993308)

# 概要

リモートに存在する複数の相手に、メッセージを配信する機能を実現するのが、Nimbus Publisherである。

送信側には、ServerConnectionFactoryが存在し、そこからServerConnectionを取得し、Messageを生成して送信する。送信は、同期または非同期で送信できる。但し、ここで言う同期は、「受信者が受信した事」ではなく「送信者がネットワーク上にMessageを送信した事」を指す。つまり、送信のI/Oを待つ事である。非同期の場合は、送信I/Oを待たない。

送信するMessageには、宛先、件名、キー、内容が指定できる。宛先は、指定しなければ受信者全員を示す。特定の宛先に送りたい時のみ、宛先を明示する。件名とキーは、セットで複数指定する事が可能である。件名およびキーは、受信者が受信したい件名、または件名およびキーを指定する事で、受信登録を行うために使用する。また、キーは、Messageの到達順序保障にも用いられており、同一のキーを持つMessageは、到達順が保障される。

受信側には、ClientConnectionFactoryを取得するための何らかの手段があり、そこからClientConnectionを取得する。ClientConnectionFactoryの取得方法は、実装に依存するが、最も基本的な取得方法は、送信側のJVM上のRMIレジストリからRMIリモートオブジェクトとして取得する。受信するMessageの件名やキーの受信登録を行った上で、Messageの受信を開始する。受信開始時には、時間を遡って受信開始の要求を行える。但し、どこまで遡れるかは、設定に依存する。Messageの受信は、MessageListernerを登録して受動的に受信する。

送受信間の通信プロトコルは実装に依存し、TCP、UDP（ユニキャスト、ブロードキャスト、マルチキャスト）を提供する。

送信側

Message

受信側

受信側

ServerConnectionFactory

ClientConnectionFactory

同期送信

受信

ServerConnection

ClientConnection

MessageListener

Message

非同期送信

受信側には、複数の受信者たるMessageListenerが存在する場合がある。その場合、MessageListener毎に受信したいメッセージが異なる場合があるため、それらを統合管理しClientConnectionに受信登録を行う必要がある。そのような複数のMessageListenerの管理を行うのが、MessageReceiverである。

受信側

ClientConnectionFactory

**MessageReceiver**

ClientConnection

MessageListener

MessageListener

MessageListener

Message

Message

Message

MessageReceiverは、各MessageListenerの受信する量や、受信時の処理内容の違いから来る処理速度の違いを吸収するために、内部に複数のキューや受信処理スレッドを持ち、受信処理を並列に行うことができる。

ネットワーク構成の都合や、配信負荷の分散などから、配信を多段構成で行いたい場合がある。その場合、送信側と受信側の間で仲立ちを行い、メッセージを転送する役割が必要となる。そのような役割を実現するのが、MessageForwardServiceである。

送信側

Message

ServerConnectionFactory

ServerConnection

ClientConnection

転送

Message

ClientConnectionFactory

**MessageForwardService**

ServerConnectionFactory

Message

ServerConnection

ClientConnection

受信側

Message

ClientConnectionFactory

前述したServerConnectionは、受信側の応答を待たない1対多配信を行うインタフェースであるが、受信側の応答を待って1対多配信を行う要件もある。その要件を満たすのが、RequestServerConnectionである。送信したMessageに対する応答の待ち方として、同期的に応答を待つ方法、非同期的に応答を待つ方法、応答をコールバックしてもらう方法の３つがある。また、受信側では、受動的にMessageを受け取り、応答を返すRequestMessageListenerがある。

送信側（受信側）

RequestConnection

FactoryService

ServerConnectionFactory

ServerConnection

Message

ClientConnection

RequestMessageListener

MessageReceiverService

ClientConnectionFactory

応答Message

**RequestServerConnection**

受信側（送信側）

ServerConnection

ClientConnection

**RequestServerConnection**

ClientConnectionFactory

MessageReceiverService

RequestConnection

FactoryService

RequestMessageListener

ServerConnectionFactory

応答Message

Message

ClientConnectionFactoryおよびServerConnectionFactoryインタフェースの実装クラスには、通信プロトコルそのものを具現化する実装の他にも、クラスタ化、グループ化、通信分散、間引きなど様々な、デコレーションを行う実装も存在する。

JavaDocは、以下を参照。

<http://nimbus.osdn.jp/reports/apidocs/jp/ossc/nimbus/service/publish/package-summary.html>

<http://nimbus.osdn.jp/reports/apidocs/jp/ossc/nimbus/service/publish/tcp/package-summary.html>

<http://nimbus.osdn.jp/reports/apidocs/jp/ossc/nimbus/service/publish/udp/package-summary.html>

サンプルコードは、以下のzipファイル内のpublishを参照。

<https://osdn.jp/projects/nimbus/downloads/64236/nimbus-sample-1.2.3.zip>

# インタフェース設計

## Message

送信する情報を抽象化したインタフェースである。

基本的に、宛先、件名、キー、内容が指定可能である。宛先は、指定しなければ受信者全員を示す。特定の宛先に送りたい時のみ、宛先を明示する。件名とキーは、セットで複数指定する事が可能である。件名およびキーは、受信者が受信したい件名、または件名およびキーを指定する事で、受信登録を行うために使用する。また、キーは、Messageの到達順序保障にも用いられており、同一のキーを持つMessageは、到達順が保障される。

その実装は、送信する通信プロトコルを具現化するServerConnectionの実装に依存するため、ServerConnectionから生成する。生成時に、件名、キーを指定する。

また、受信側では、受信する接続を具現化するClientConnectionからMessageを受信し、内容を取得する。不要になったMessageオブジェクトをClientConnectionに再利用を促すために、recycle()メソッド呼び出す。（送信側では、送信後に自動的に再利用される）

ServerConnection

・件名

・キー

**Message**

**・件名**

**・キー**

・件名

・キー

内容

**宛先**

宛先

生成

件名追加

送信

宛先指定

受信

MessageListener

ClientConnection

内容指定

再利用

内容取得

内容

以下の属性を持つ。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 説明 | 備考 |
| 1 | 件名 | 受信者が受信要求を出すために用いる。  １つのメッセージに複数の用途がある場合もあるため、ServerConnectionから生成する際に指定した件名以外に、後から複数の件名を追加する事もできる。但し、生成する際に指定した件名、キーがプライマリな情報になる。 | 必須  １から複数 |
| 2 | キー | 受信者が受信要求を出すために用い、件名の補足情報としてセットで指定する。キーにnullを指定した場合は、セットとなる件名のMessageを全て受信する事を意味する。  また、Messageの到達順序保障にも用いられており、同一のキーを持つMessageは、到達順が保障される。但し、複数の件名、キーが指定されている場合は、生成する際に指定したキーが到達順保障のキーとなる。 | 0から複数（但し、件名ごとに１） |
| 3 | 内容 | Messageの本文たる任意のオブジェクト。  但し、直列化可能であること。 |  |
| 4 | 宛先 | 宛先のIDを指定して、送信相手を絞り込む。指定しない場合は、全員に送信する。  宛先のIDの一覧は、ServerConnectionから、取得可能。 | 0から複数 |

## ServerConnection

送信側で、Messageを送信するための接続を抽象化したインタフェースである。ServerConnectionFactoryから取得する。

受信者たるClientの管理、メッセージの生成、送信、ServerConnectionListenerの登録・解除を行う機能を持つ。

送信は、同期または非同期で送信できる。但し、ここで言う同期は、「受信者が受信した事」ではなく「送信者がネットワーク上にMessageを送信した事」を指す。つまり、送信のI/Oを待つ事である。非同期の場合は、送信I/Oを待たない。送信時には、各Clientが要求している件名、キーと合致するものだけを、各Clientに対応するClientConnectionへ送信する。

また、ClientConnectionからの、接続要求、受信件名登録、受信件名削除、受信開始、受信停止、切断などの要求イベントを、登録されたServerConnectionListenerに通知する。

ServerConnectionはServerConnectionFactoryから取得するため、実装はServerConnectionFactoryインタフェースの実装に依存する。

ServerConnectionFactory

取得

**Client**

**Client**

**宛先ID**

**件名**

**キー**

**キー**

生成

**ServerConnection**

同期送信

ClientConnection

ClientConnection

Message

非同期送信

要求イベント

ServerConnectionListener

## ClientConnection

受信側で、Messageを受信するための接続を抽象化したインタフェースである。ClientConnectionFactoryから取得する。

接続の確立、受信件名の登録・削除、受信の開始・終了、接続の切断、MessageListenerの登録・解除を行う機能を持つ。

Messageの受信は、受信通知を受けるMessageListenerを登録することで、受動的に受信する。

ClientConnectionはClientConnectionFactoryから取得するため、実装はClientConnectionFactoryインタフェースの実装に依存する。

ClientConnectionFactory

接続

**ClientConnection**

受信開始・終了

件名登録・削除

ServerConnection

取得

MessageListener

切断

受信

Message

## RequestServerConnection

Messageを送受信するための接続を抽象化したインタフェースで、ServerConnectionのサブインタフェースである。RequestConnectionFactoryServiceから取得する。

送受信とは、Messageを送信して、その応答を受け取る事を指す。応答の受信方法としては、同期的に受信する、コールバックで受信する、非同期に受信する方法がある。

また、送信側からMessageを受信して応答する側は、RequestConnectionFactoryServiceにRequestMessageListenerを登録する事で、送信側からのMessageを受信し、応答する。

送信側

RequestConnection

FactoryService

取得

生成

**RequestServerConnection**

RequestServerConnection

送信

ServerConnection

Message

ClientConnection

MessageReceiver

Message

同期受信

応答Object

非同期受信

ResponseCallBack

コールバック受信

受信側

RequestConnection

FactoryService

RequestMessageListener

Message

**RequestServerConnection**

RequestServerConnection

ClientConnection

MessageReceiver

Message

応答Object

ServerConnection

## ServerConnectionFactory

送信側で、Messageを送信するための接続であるServerConnectionを生成するファクトリインタフェースである。

ServerConnectionFactoryインタフェースの実装には、大きく分けて、通信プロトコル実装と、機能付加型のデコレーション実装がある。通信プロトコル実装は、ServerConnectionFactoryインタフェースと共に対となるClientConnectionFactoryインタフェースも実装している。デコレーション実装は、実体となるServerConnectionを生成するServerConnectionFactoryをインジェクションして利用するラップ型の実装である。

以下に、ServerConnectionFactoryインタフェースの通信プロトコル実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.publish.tcp.ConnectionFactoryService | TCPプロトコルを利用した実装クラスである。 |
| 2 | jp.ossc.nimbus.service.publish.udp.ConnectionFactoryService | UDPプロトコルを利用した実装クラスである。  設定によって、ユニキャスト、ブロードキャスト、マルチキャストを利用可能である。 |

以下に、ServerConnectionFactoryインタフェースのデコレーション実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.publish.DistributedConnectionFactoryService | 実体となるServerConnectionを複数使って通信を行う事で送信性能を向上させる実装クラスである。 |
| 2 | jp.ossc.nimbus.service.publish.GroupConnectionFactoryService | 複数のServerConnectionをグルーピングして、１つのServerConnectionに見せる実装クラスである。 |
| 3 | jp.ossc.nimbus.service.publish.ThinOutServerConnectionFactoryService | ThinOutFilterインタフェース実装サービスを利用して、間引き処理を行うServerConnectionを提供する実装クラスである。  複数のThinOutFilterを重ね掛けする事も可能である。 |
| 4 | jp.ossc.nimbus.service.publish.RequestConnectionFactoryService | RequestServerConnectionにラップする実装クラスである。 |

## ClientConnectionFactory

受信側で、Messageを受信するための接続であるClientConnectionを生成するファクトリインタフェースである。

ClientConnectionFactoryインタフェースの実装には、大きく分けて、通信プロトコル実装と、機能付加型のデコレーション実装がある。通信プロトコル実装は、ClientConnectionFactoryインタフェースと共に対となるServerConnectionFactoryインタフェースも実装している。デコレーション実装は、実体となるClientConnectionを生成するClientConnectionFactoryをインジェクションして利用するラップ型の実装である。

以下に、ClientConnectionFactoryインタフェースの通信プロトコル実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.publish.tcp.ConnectionFactoryService | TCPプロトコルを利用した実装クラスである。  ClientConnectionFactoryをリモートに提供する方法は、JNDIまたはRMIである。 |
| 2 | jp.ossc.nimbus.service.publish.udp.ConnectionFactoryService | UDPプロトコルを利用した実装クラスである。  設定によって、ユニキャスト、ブロードキャスト、マルチキャストを利用可能である。  ClientConnectionFactoryをリモートに提供する方法は、JNDIまたはRMIである。 |

以下に、ClientConnectionFactoryインタフェースのデコレーション実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.publish.ClusterConnectionFactoryService | 複数のサーバのServerConnectionをクラスタ化して、それに接続するClientConnectionを提供する実装クラスである。  FirstAvairableなServerConnectionに接続するActive-HotStandby方式。  接続数が少ないServerConnectionに接続する分散Active-Active方式。  全てのServerConnectionに接続するActive-Active方式。  の３つのクラスタ方式を選択可能である。  ClusterClientConnectionFactoryをリモートに提供する方法は、JNDI、RMIまたはClusterServiceである。 |
| 2 | jp.ossc.nimbus.service.publish.ClusterClientConnectionFactoryService | ClusterConnectionFactoryServiceと対にして使用する実装クラスである。  クラスタ化されたClientConnectionFactoryのホストサーバを隠蔽するために、ClusterServiceからClusterClientConnectionFactoryを取得する。 |
| 3 | jp.ossc.nimbus.service.publish.DistributedConnectionFactoryService | 実体となるServerConnectionを複数使って通信を行う事で送信性能を向上させるServerConnectionに接続するためのClientConnectionを提供する実装クラスである。  ClientConnectionFactoryをリモートに提供する方法は、JNDIまたはRMIである。 |
| 4 | jp.ossc.nimbus.service.publish.GroupConnectionFactoryService | 複数のServerConnectionをグルーピングして、１つのServerConnectionに見せるServerConnectionに接続するためのClientConnectionを提供する実装クラスである。  ClientConnectionFactoryをリモートに提供する方法は、JNDIまたはRMIである。 |
| 5 | jp.ossc.nimbus.service.publish.GroupClientConnectionFactoryService | 複数のClientConnectionをグルーピングして、１つのClientConnectionに見せる実装クラスである。 |

## ThinOutFilter

ThinOutServerConnectionFactoryServiceで利用する、送信Messageの間引きフィルタ機能のインタフェースである。

ThinOutServerConnection

FactoryService

ServerConnectionFactory

ThinOutServerConnection

ServerConnection

**ThinOutFilter**

**ThinOutFilter**

Message

以下に、ThinOutFilterインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.publish.SubjectThinOutFilterService | 特定の件名のMessageを通過許可したり、通過拒否したりする実装クラスである。 |
| 2 | jp.ossc.nimbus.service.publish.TimeIntervalThinOutFilterService | 時間間隔でMessageをフィルタする実装クラスである。 |

## MessageReceiver

受信者たる複数のMessageListenerを統合管理し、一括してClientConnectionに受信登録を行う機能を提供するインタフェースである。

MessageListenerで受信したい件名とキーをMessageReceiverに登録すると、各MessageListenerに登録された件名とキーを統合して、ClientConnectionに登録を行う。また、件名とキーの登録解除時も同様で、解除された件名及びキーが、すべてのMessageListenerにとって不要となった場合に、ClientConnectionから登録解除を行う。

ClientConnectionからMessageを受信すると、Messageの件名及びキーと、各MessageListenerに登録されている件名及びキーをマッチングさせて、該当するMessageListenerへの配信を行う。

**MessageReceiver**

件名

MessageListener

件名

キー

件名

キー

ClientConnection

キー

キー

件名

Message

件名

キー

MessageListener

件名

キー

件名

キー

MessageListener

件名

キー

件名

キー

以下に、MessageReceiverインタフェースの実装クラスの一覧を示す。

|  |  |  |
| --- | --- | --- |
| # | 実装クラス名 | 概要 |
| 1 | jp.ossc.nimbus.service.publish.MessageReceiverService | MessageReceiverインタフェースのデフォルト実装クラスである。 |
| 2 | jp.ossc.nimbus.service.publish.MessageForwardService | MessageReceiverServiceのサブクラスで、ClientConnectionから受信したメッセージを、ServerConnectionを使って送信する、Messageの転送機能を実現する実装クラスである。 |

# 実装設計

インタフェース設計で示した各インタフェースのうち、機能性のあるインタフェースの実装クラスの機能を説明する。

また、インタフェース設計に現れなかった、クラスの機能も説明する。

## ServerConnectionFactory

### jp.ossc.nimbus.service.publish.tcp.ConnectionFactoryService

TCPプロトコルで通信するServerConnectionを生成するServerConnectionFactoryインタフェースの実装クラスである。また同時に、TCPプロトコルで通信するClientConnectionを生成するClientConnectionFactoryインタフェースの実装クラスでもある。

このサービスは送信側に存在するため、ローカルでサービスを取得しServerConnectionFactoryにキャストして、ServerConnectionを取得する。

一方、このサービスは、RMI系のRepositoryインタフェース実装サービス（RMI実装や、JNDI実装が存在する）を設定する事で、ClientConnectionFactoryのリモートオブジェクトをバインドする事ができる。そのため、リモートの受信側では、RepositoryサービスからClientConnectionFactoryを取得し、RMIでClientConnectionを取得する。

但し、デコレーション系のServerConnectionFactoryと組み合わせて使用する場合、その実装に依存する。

送信側

受信側

Repository

Repository

lookup()

ClientConnectionFactory

（サーバオブジェクト）

ClientConnectionFactory

（リモートオブジェクト）

register()

**ConnectionFactoryService**

getServerConnection()

getClientConnection()

onMessage()

Message

ClientConnection

ServerConnection

send()

Message

MessageListener

NIOを使用する場合のServerConnection

ServerConnection

ServerSocketChannel

I/O処理デーモン

Selector

同期送信スレッド

件名

非同期送信

スレッド

非同期送信

スレッド

キー

キー

件名

宛先ID

非同期送信

スレッド

Client

Client

非同期送信受付デーモン

Client

SocketChannel

SocketChannel

同期送信スレッドプール

非同期送信スレッド

送信バッファ

送信キャッシュ

送信バッファチェックデーモン

NIOを使用する場合、TCPソケットの確立を受け付ける処理と、TCPソケットからの読み込み処理、TCPソケットへの書き込み処理を、NIOのSelectorを利用する事で、１つのスレッド「I/O処理デーモン」で行う。

クライアントからTCPソケットの確立が要求されると、確立されたSocketChannelとそのクライアント用に生成したClientを紐づけ、Selectorに読み込み待ちとして登録する。

クライアントから、TCPソケットに書き込まれると、Selectorからの通知でI/O処理デーモンが動き出し、要求された処理（ClientConnectionの接続、件名登録・削除、受信開始・終了、切断など）をClientに依頼する。

ServerConnectionに同期送信を行うと、各Clientの受信要求に合致する送信要求がClientごとに同期送信要求キューに投入され、同期送信応答キューから投入した分だけ応答が取得できるまでブロックされる。同期送信要求キューに投入された各Clientへの送信要求は、同期送信スレッドプール内のスレッドで並列に処理され、各Clientの送信バッファに書き込まれる。また、同時に送信キャッシュへの登録も行う。送信キャッシュは、クライアントからの受信開始要求で、時間を遡って送信するように要求された場合に、その要求に応えるために、一定時間分の送信メッセージをキャッシュしておく場所である。さらに同時に、Selectorへ書き込み要求を登録する。但し、送信バッファ時間および送信バッファサイズが指定されている場合は、それぞれの条件を満たすまでは、Selectorへ書き込み要求登録は行わない。送信バッファ時間の送信条件は、送信バッファの1件目の送信要求時刻が、送信バッファ時間以上経過している事である。また、送信バッファサイズの送信条件は、送信バッファのバイト数が送信バッファサイズを超えている事である。最後に、同期送信応答キューに送信完了の応答を投入する。Selectorへ書き込み要求登録によって、書き込み準備が整うと、Selectorから通知を受けたI/O処理デーモンが、送信バッファからTCPソケットへの書き込みを行い、送信される。

ServerConnectionに非同期送信を行うと、各Clientの受信要求に合致する送信要求がClientごとに非同期送信受付キューに投入される。非同期送信受付キューに投入された各Clientへの送信要求は、非同期送信受付デーモンによって、非同期送信要求キューに分散して投入され、非同期送信応答キューから投入した分だけ応答が取得できるまでブロックされる。この際のキューの振り分けは、同じClientに対する送信要求は、同じキューに投入される。非同期送信要求キューに投入された各Clientへの送信要求は、非同期送信スレッドプール内のスレッドで並列に処理され、各Clientの送信バッファに書き込まれる。以降は、同期送信と同様の動作である。

送信バッファ時間が指定されている場合は、送信バッファチェックデーモンが定期的に、送信バッファの経過時間をチェックし、条件を満たす場合は、Selectorへ書き込み要求登録を行う。

NIOを使用しない場合のServerConnection

ServerConnection

ServerSocket

同期送信スレッドプール

非同期送信受付デーモン

件名

件名

キー

キー

非同期送信

スレッド

非同期送信

スレッド

非同期送信

スレッド

送信キャッシュ

送信バッファ

送信バッファチェックデーモン

同期送信スレッド

非同期送信スレッド

受付処理デーモン

Client

Client

Client

リクエスト処理デーモン

宛先ID

Socket

Socket

NIOを使用しない場合、TCPソケットの確立を受け付ける処理を、「受付処理デーモン」で行う。

クライアントからTCPソケットの確立が要求されると、確立されたSocketとそのクライアント用に生成したClientを紐づけ、Client毎にリクエスト処理デーモンを立ち上げ、読み込み待ちにする。

クライアントから、TCPソケットに書き込まれると、リクエスト処理デーモンが読み込み、要求された処理（ClientConnectionの接続、件名登録・削除、受信開始・終了、切断など）をClientに依頼する。

ServerConnectionに同期送信を行うと、各Clientの受信要求に合致する送信要求がClientごとに同期送信要求キューに投入され、同期送信応答キューから投入した分だけ応答が取得できるまでブロックされる。同期送信要求キューに投入された各Clientへの送信要求は、同期送信スレッドプール内のスレッドで並列に処理され、各Clientの送信バッファに書き込まれる。また、同時に送信キャッシュへの登録も行う。さらにTCPソケットへ書き込みを行う。但し、送信バッファ時間および送信バッファサイズが指定されている場合は、それぞれの条件を満たすまでは、TCPソケットへ書き込まない。最後に、同期送信応答キューに送信完了の応答を投入する。

ServerConnectionに非同期送信を行うと、各Clientの受信要求に合致する送信要求がClientごとに非同期送信受付キューに投入される。非同期送信受付キューに投入された各Clientへの送信要求は、非同期送信受付デーモンによって、非同期送信要求キューに分散して投入され、非同期送信応答キューから投入した分だけ応答が取得できるまでブロックされる。この際のキューの振り分けは、同じClientに対する送信要求は、同じキューに投入される。非同期送信要求キューに投入された各Clientへの送信要求は、非同期送信スレッドプール内のスレッドで並列に処理され、各Clientの送信バッファに書き込まれる。以降は、同期送信と同様の動作である。

送信バッファ時間が指定されている場合は、送信バッファチェックデーモンが定期的に、送信バッファの経過時間をチェックし、条件を満たす場合は、TCPソケットへ書き込む。

受信側のClientConnectionは、NIOは使用しない。

ClientConnection

接続

受信件名登録・解除

受信開始・終了

切断

受信処理デーモン

Message

Socket

受信

MessageListener

接続、受信件名登録・解除、受信開始・終了、切断などの要求処理は、要求したスレッドで同期的にTCPソケットへ書き込み、応答を受け取る。Messageの受信は、受信処理デーモンがTCPソケットから読み取り、MessageListenerに通知する。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ClientAddressPropertyName | String | クライアント側でバインドするローカルアドレスを指定するシステムプロパティ名を設定する。  デフォルトは、”jp.ossc.nimbus.service.publish.tcp.bindAddress” |
| 2 | ClientPortPropertyName | String | クライアント側でバインドするローカルポート番号を指定するシステムプロパティ名を設定する。  デフォルトは、”jp.ossc.nimbus.service.publish.tcp.bindPort” |
| 3 | ClientReconnectCount | Int | クライアント側で接続切断を検知した場合の再接続試行回数を設定する。  デフォルトは、0で再接続しない。  サーバ側を再起動して、再接続する場合は、属性ServerPortでサーバ側のポート番号を固定する必要がある。 |
| 4 | ClientReconnectInterval | int | クライアント側で接続切断を検知した場合の再接続試行間隔[ms]を設定する。  デフォルトは、0。 |
| 5 | ClientReconnectBufferTime | Long | クライアント側で接続切断を検知した場合に最後に受信したメッセージの受信時刻からどのくらいの時間[ms]だけ遡って再送要求を出すかを設定する。  デフォルトは、0[ms]で、最終受信時刻から再送要求する。 |
| 6 | ServerAddress | String | サーバ側の接続待ち受けアドレスを設定する。  指定しない場合は、ローカルアドレス。 |
| 7 | ServerPort | Int | サーバ側の接続待ち受けポート番号を設定する。  デフォルトは、0で任意のポート番号。 |
| 8 | ServerBacklog | Int | サーバソケットのバックログを設定する。  デフォルトは0。 |
| 9 | ServerSocketFactoryServiceName | ServiceName | サーバソケットファクトリサービスのサービス名を設定する |
| 10 | SocketFactoryServiceName | ServiceName | ソケットファクトリサービスのサービス名を設定する |
| 11 | NIOSocketFactoryServiceName | ServiceName | サーバ側のソケットでjava.nioを使った接続を行う場合のサーバ側のソケットに適用するソケットファクトリサービスのサービス名を設定する。 |
| 12 | NIO | boolean | サーバ側のソケットでjava.nioを使った接続を行うかどうかを設定する。  デフォルトは、false。 |
| 13 | JndiName | String | ClientConnectionFactoryリモートオブジェクトのJNDI名を設定する。  デフォルトは、”publish/ClientConnectionFactory”。 |
| 14 | JndiRepositoryServiceName | ServiceName | ClientConnectionFactoryサーバオブジェクトをバインドするjp.ossc.nimbus.service.repository.Repositoryサービスのサービス名を設定する。 |
| 15 | RMIPort | Int | ClientConnectionFactoryリモートオブジェクトのRMI通信ポート番号を設定する。 |
| 16 | SendThreadSize | Int | 同期送信をする際に、各送信先へ並列に送信処理を行う同期送信スレッドプールのプール数を設定する。  デフォルトは1で、各送信先へ直列に送信処理を行う。 |
| 16 | SendQueueServiceName | ServiceName | 同期送信をする際に、並列送信を行うためのjp.ossc.nimbus.service.queue.Queueサービスのサービス名を設定する。  属性SendThreadSizeに、2以上を指定しない場合は、並列送信しないため、Queueは使用しない。  2以上を指定した場合で、この属性を指定しない場合は、内部でQueueを自動生成する。 |
| 17 | AsynchSendThreadSize | int | 非同期送信をする際に、各送信先へ並列に送信処理を行う非同期送信スレッドプール  のプール数を設定する。  デフォルトは0で、非同期送信をサポートしない。 |
| 18 | AsynchSendQueueServiceName | ServiceName | 非同期送信の受付に使用するjp.ossc.nimbus.service.queue.Queueサービスのサービス名を設定する。  属性AsynchSendThreadSizeに、1以上を指定しない場合は、非同期送信をサポートしないため、Queueは使用しない。  1以上を指定した場合で、この属性を指定しない場合は、内部でQueueを自動生成する。 |
| 19 | AsynchSendQueueFactoryServiceName | ServiceName | 非同期送信をする際に、並列送信を行うためのjp.ossc.nimbus.service.queue.Queuのファクトリサービス名を設定する。  属性AsynchSendThreadSizeに、1以上を指定しない場合は、非同期送信をサポートしないため、Queueは使用しない。  1以上を指定した場合で、この属性を指定しない場合は、内部でQueueを自動生成する。 |
| 20 | MaxSendRetryCount | int | 送信に失敗した場合に再送信を行う回数を設定する。  デフォルトは、0で再送信しない。 |
| 21 | SendMessageCacheTime | long | 送信メッセージのキャッシュ時間を設定する。  受信開始時に、時間を遡って受信するために、送信側で送信キャッシュに、送信したメッセージを送信した時刻からどのくらいの間残しておくかを設定する。  デフォルトは、5000[ms] |
| 22 | SendBufferTime | long | 送信メッセージをまとめて送信するための滞留時間を設定する。  デフォルトは、0で滞留させない。 |
| 23 | SendBufferSize | int | 送信メッセージをまとめて送信するための滞留バイト数を設定する。  デフォルトは、0で滞留させない。 |
| 24 | SendBufferTimeoutInterval | long | 送信メッセージをまとめて送信する場合に、送信滞留チェック間隔[ms]を設定する。  デフォルトは、1000[ms]。 |
| 25 | Acknowledge | boolean | クライアントからの要求に対して、応答を返すかどうかを設定する。  デフォルトでは、falseで応答を返さない。 |
| 26 | ClientResponseTimeout | long | クライアントがサーバからの応答を待つタイムアウト[ms]を設定する。  デフォルトは、30秒。 |
| 27 | ClientMessageRecycleBufferSize | int | クライアント側で受信メッセージを再利用する際の受信メッセージバッファ数を設定する。  デフォルトは、100。 |
| 28 | ExternalizerServiceName | ServiceName | メッセージの直列化/非直列化に使用するjp.ossc.nimbus.service.io.Externalizerサービスのサービス名を設定する。  指定しない場合は、java.io.ObjectOutputStreamでメッセージの直列化/非直列化を行う。 |
| 29 | ServerConnectionListenerServiceNames | ServiceName[] | ServerConnectionListenerのサービス名を設定する。 |

### jp.ossc.nimbus.service.publish.udp.ConnectionFactoryService

UDPプロトコルで通信するServerConnectionを生成するServerConnectionFactoryインタフェースの実装クラスである。また同時に、UDPプロトコルで通信するClientConnectionを生成するClientConnectionFactoryインタフェースの実装クラスでもある。

TCPプロトコルに比べて、軽量なプロトコルなので、通信量の削減、配信速度の向上などが望める。その反面、順序制御、再送制御、パケット分割など通信プロトコルが担保しない部分を補う必要があるため、複雑な仕組みとチューニングが必要である。

設定方法によって、UDPユニキャスト、UDPブロードキャスト、UDPマルチキャストの３つのプロトコルが選択可能である。ブロードキャストとマルチキャストは、通信プロトコル的に1対多なので、Javaプロセスの配信処理コストが低く、受信者が多い場合に適していると言える。しかし、件名のフィルタリングを受信側で行うため、使い方によってはユニキャストより通信量が増大する可能性もある。

このサービスは送信側に存在するため、ローカルでサービスを取得しServerConnectionFactoryにキャストして、ServerConnectionを取得する。

一方、このサービスは、RMI系のRepositoryインタフェース実装サービス（RMI実装や、JNDI実装が存在する）を設定する事で、ClientConnectionFactoryのリモートオブジェクトをバインドする事ができる。そのため、リモートの受信側では、RepositoryサービスからClientConnectionFactoryを取得し、RMIでClientConnectionを取得する。

但し、デコレーション系のServerConnectionFactoryと組み合わせて使用する場合、その実装に依存する。

送信側

受信側

**ConnectionFactoryService**

Repository

ClientConnectionFactory

（サーバオブジェクト）

ServerConnection

register()

getServerConnection()

Repository

ClientConnectionFactory

（リモートオブジェクト）

ClientConnection

lookup()

getClientConnection()

Message

Message

MessageListener

send()

onMessage()

NIOを使用する場合で、ユニキャスト設定のServerConnection

ServerConnection

ServerSocketChannel

受付処理デーモン

I/O処理

スレッドプール

Selector

Client

Client

同期送信スレッド

件名

非同期送信

スレッド

非同期送信

スレッド

キー

キー

件名

宛先ID

非同期送信

スレッド

非同期送信受付デーモン

Client

SocketChannel

SocketChannel

SocketChannel

同期送信スレッドプール

非同期送信スレッド

送信キャッシュ

DatagramSocket

DatagramSocket

DatagramSocket

NIOを使用する場合、クライアントからの要求を受け付けるTCPソケットの確立を受け付ける処理と、TCPソケットからの読み込み監視、TCPソケットへの書き込み監視をNIOのSelectorを利用する事で、１つのスレッド「受付処理デーモン」で行う。

クライアントからTCPソケットの確立が要求されると、確立されたSocketChannelとそのクライアント用に生成したClientを紐づけ、Selectorに読み込み待ちとして登録する。

クライアントから、TCPソケットに書き込まれると、Selectorからの通知で受付処理デーモンが動き出し、要求されたSocketChannelとClientをI/O処理受付Queueに投入し、I/O処理スレッドプールで、要求された処理（ClientConnectionの接続、件名登録・削除、受信開始・終了、切断など）が並列に処理される。応答が必要な場合は、各Clientの応答キューに書き込み、Selectorに書き込み待ちとして登録する。その後、Selectorからの通知で、受付処理デーモンが動き出し、要求されたSocketChannelとClientをI/O処理受付Queueに投入し、I/O処理スレッドプールで、応答キューから取得された書き込み要求をSocketChannelへ書き出す。

ServerConnectionに同期送信を行うと、各Clientの受信要求に合致する送信要求がClientごとに同期送信要求キューに投入され、同期送信応答キューから投入した分だけ応答が取得できるまでブロックされる。同期送信要求キューに投入された各Clientへの送信要求は、同期送信スレッドプール内のスレッドで並列に処理され、各ClientでMessage内にメッセージ通番が発番され、一定のサイズのパケットに分割され、DatagramSocketに書き込まれ、送信される。また、同時に送信キャッシュへの登録も行う。送信キャッシュは、クライアントからの受信開始要求で、時間を遡って送信するように要求された場合に、その要求に応えるために、一定時間分の送信メッセージをキャッシュしておく場所である。最後に、同期送信応答キューに送信完了の応答を投入する。

ServerConnectionに非同期送信を行うと、送信要求が非同期送信受付キューに投入される。非同期送信受付キューに投入された送信要求は、非同期送信受付デーモンによって、各Clientの受信要求に合致する分だけがClientごとの送信要求として、非同期送信要求キューに分散して投入され、非同期送信応答キューから投入した分だけ応答が取得できるまでブロックされる。この際のキューの振り分けは、同じClientに対する送信要求は、同じキューに投入される。非同期送信要求キューに投入された各Clientへの送信要求は、非同期送信スレッドプール内のスレッドで並列に処理され、各Clientで同期送信と同様に送信される。

NIOを使用しない場合で、ユニキャスト設定のServerConnection

ServerConnection

ServerSocket

受付処理デーモン

Client

Client

Client

リクエスト処理デーモン

同期送信スレッド

件名

非同期送信

スレッド

非同期送信

スレッド

キー

キー

件名

宛先ID

非同期送信

スレッド

非同期送信受付デーモン

Socket

Socket

同期送信スレッドプール

Socket

非同期送信スレッド

送信キャッシュ

DatagramSocket

DatagramSocket

DatagramSocket

NIOを使用しない場合、クライアントからの要求を受け付けるTCPソケットの確立を受け付ける処理を「受付処理デーモン」で行う。

クライアントからTCPソケットの確立が要求されると、確立されたSocketとそのクライアント用に生成したClientを紐づけ、Client毎にリクエスト処理デーモンを立ち上げ、読み込み待ちにする。

クライアントから、TCPソケットに書き込まれると、リクエスト処理デーモンが読み込み、要求された処理（ClientConnectionの接続、件名登録・削除、受信開始・終了、切断など）をClientに依頼する。

ServerConnectionに同期送信を行うと、各Clientの受信要求に合致する送信要求がClientごとに同期送信要求キューに投入され、同期送信応答キューから投入した分だけ応答が取得できるまでブロックされる。同期送信要求キューに投入された各Clientへの送信要求は、同期送信スレッドプール内のスレッドで並列に処理され、各ClientでMessage内にメッセージ通番が発番され、一定のサイズのパケットに分割され、DatagramSocketに書き込まれ、送信される。また、同時に送信キャッシュへの登録も行う。最後に、同期送信応答キューに送信完了の応答を投入する。

ServerConnectionに非同期送信を行うと、送信要求がClientごとに非同期送信受付キューに投入される。非同期送信受付キューに投入された送信要求は、非同期送信受付デーモンによって、各Clientの受信要求に合致する分だけがClientごとの送信要求として、非同期送信要求キューに分散して投入され、非同期送信応答キューから投入した分だけ応答が取得できるまでブロックされる。この際のキューの振り分けは、同じClientに対する送信要求は、同じキューに投入される。非同期送信要求キューに投入された各Clientへの送信要求は、非同期送信スレッドプール内のスレッドで並列に処理され、各Clientで同期送信と同様に送信される。

受信側のClientConnectionは、NIOを使用しない。

ユニキャスト設定のClientConnection

ClientConnection

接続

受信件名登録・解除

受信開始・終了

切断

Socket

応答受信処理デーモン

配信パケット

受信処理デーモン

DatagramSocket

配信メッセージ

受信処理デーモン

MessageListener

受信

Message

ロスト補完要求

ロスト検知デーモン

接続、受信件名登録・解除、受信開始・終了、切断などの要求処理は、要求したスレッドでTCPソケットへ書き込み、要求番号に対して待ち受ける。同時に、応答受信デーモンが応答を受信して、要求番号で待ち受けているスレッドに通知する。Messageの受信は、配信パケット受信処理デーモンがUDPソケットから読み取り、パケット受信キューに投入する。パケット受信キューを待ち受ける配信メッセージ受信処理デーモンが、分割されたパケットを結合しMessageに復元し、通番による到着順序の制御を行い、MessageListenerに通知する。通番が揃わない場合は、到着待ちキューに格納し、通番が揃うのを待たせる。並行して、ロスト検知デーモンが、一定時間以上通番が揃わない、または到着待ちメッセージが一定件数以上溜まった場合、TCPソケットを使って、サーバにロスト補完要求を行い、到着待ちキューの足りないMessageを補完し、配信メッセージ受信処理デーモンに通知する。

NIOを使用する場合で、ブロードキャスト及びマルチキャスト設定のServerConnection

ServerConnection

ServerSocketChannel

受付処理デーモン

I/O処理

スレッドプール

Selector

Client

Client

件名

キー

キー

件名

宛先ID

SocketChannel

同期送信スレッド

SocketChannel

非同期送信スレッド

送信キャッシュ

非同期送信受付デーモン

DatagramSocket

NIOを使用する場合、クライアントからの要求を受け付けるTCPソケットの確立を受け付ける処理と、TCPソケットからの読み込み監視、TCPソケットへの書き込み監視をNIOのSelectorを利用する事で、１つのスレッド「受付処理デーモン」で行う。

クライアントからTCPソケットの確立が要求されると、確立されたSocketChannelとそのクライアント用に生成したClientを紐づけ、Selectorに読み込み待ちとして登録する。

クライアントから、TCPソケットに書き込まれると、Selectorからの通知で受付処理デーモンが動き出し、要求されたSocketChannelとClientをI/O処理受付Queueに投入し、I/O処理スレッドプールで、要求された処理（ClientConnectionの接続、件名登録・削除、受信開始・終了、切断など）が並列に処理される。応答が必要な場合は、各Clientの応答キューに書き込み、Selectorに書き込み待ちとして登録する。その後、Selectorからの通知で、受付処理デーモンが動き出し、要求されたSocketChannelとClientをI/O処理受付Queueに投入し、I/O処理スレッドプールで、応答キューから取得された書き込み要求をSocketChannelへ書き出す。

ServerConnectionに同期送信を行うと、メッセージ通番が発番され、一定のサイズのパケットに分割され、DatagramSocketに書き込まれ、送信される。また、同時に送信キャッシュへの登録も行う。送信キャッシュは、クライアントからの受信開始要求で、時間を遡って送信するように要求された場合に、その要求に応えるために、一定時間分の送信メッセージをキャッシュしておく場所である。

ServerConnectionに非同期送信を行うと、送信要求が非同期送信受付キューに投入される。非同期送信受付キューに投入された送信要求は、非同期送信受付デーモンによって、同期送信と同様に送信される。

NIOを使用しない場合で、ブロードキャスト及びマルチキャスト設定のServerConnection

ServerConnection

ServerSocket

受付処理デーモン

Client

Client

Client

リクエスト処理デーモン

件名

キー

キー

件名

宛先ID

Socket

Socket

同期送信スレッド

非同期送信受付デーモン

非同期送信スレッド

Socket

送信キャッシュ

DatagramSocket

NIOを使用しない場合、クライアントからの要求を受け付けるTCPソケットの確立を受け付ける処理を「受付処理デーモン」で行う。

クライアントからTCPソケットの確立が要求されると、確立されたSocketとそのクライアント用に生成したClientを紐づけ、Client毎にリクエスト処理デーモンを立ち上げ、読み込み待ちにする。

クライアントから、TCPソケットに書き込まれると、リクエスト処理デーモンが読み込み、要求された処理（ClientConnectionの接続、件名登録・削除、受信開始・終了、切断など）をClientに依頼する。

ServerConnectionに同期送信を行うと、Message内にメッセージ通番が発番され、一定のサイズのパケットに分割され、DatagramSocketに書き込まれ、送信される。また、同時に送信キャッシュへの登録も行う。

ServerConnectionに非同期送信を行うと、送信要求が非同期送信受付キューに投入される。非同期送信受付キューに投入された送信要求は、非同期送信受付デーモンによって、同期送信と同様に送信される。

ブロードキャスト及びマルチキャスト設定のClientConnection

接続

受信件名登録・解除

受信開始・終了

切断

ClientConnection

Socket

応答受信処理デーモン

配信パケット

受信処理デーモン

DatagramSocket

件名

件名

キー

キー

配信メッセージ

受信処理デーモン

Message

ロスト補完要求

受信

MessageListener

ロスト検知デーモン

接続、受信件名登録・解除、受信開始・終了、切断などの要求処理は、要求したスレッドでTCPソケットへ書き込み、要求番号に対して待ち受ける。同時に、応答受信デーモンが応答を受信して、要求番号で待ち受けているスレッドに通知する。

Messageの受信は、配信パケット受信処理デーモンがUDPソケットから読み取り、パケット受信キューに投入する。パケット受信キューを待ち受ける配信メッセージ受信処理デーモンが、分割されたパケットを結合しMessageに復元し、通番による到着順序の制御、件名とキーのマッチングを行い、MessageListenerに通知する。通番が揃わない場合は、到着待ちキューに格納し、通番が揃うのを待たせる。並行して、ロスト検知デーモンが、一定時間以上通番が揃わない、または到着待ちメッセージが一定件数以上溜まった場合、TCPソケットを使って、サーバにロスト補完要求を行い、到着待ちキューの足りないMessageを補完し、配信メッセージ受信処理デーモンに通知する。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ClientAddressPropertyName | String | クライアント側でTCPソケットをバインドするローカルアドレスを指定するシステムプロパティ名を設定する。  デフォルトは、”jp.ossc.nimbus.service.publish.udp.bindAddress” |
| 2 | ClientPortPropertyName | String | クライアント側でTCPソケットをバインドするローカルポート番号を指定するシステムプロパティ名を設定する。  デフォルトは、”jp.ossc.nimbus.service.publish.udp.bindPort” |
| 3 | ClientUDPBindAddressPropertyName | String | クライアント側でUDPソケットをバインドするローカルアドレスを指定するシステムプロパティ名を設定する。  デフォルトは、“jp.ossc.nimbus.service.publish.udp.udpBindAddress” |
| 4 | ClientUDPNetworkInterfacesPropertyName | String | クライアント側でUDPパケットを受信するネットワークインタフェースを指定するシステムプロパティ名を設定する。  デフォルトは、“jp.ossc.nimbus.service.publish.udp.networkInterfaces” |
| 5 | ClientReconnectCount | Int | クライアント側で接続切断を検知した場合の再接続試行回数を設定する。  デフォルトは、0で再接続しない。  サーバ側を再起動して、再接続する場合は、属性ServerPortでサーバ側のポート番号を固定する必要がある。 |
| 6 | ClientReconnectInterval | Int | クライアント側で接続切断を検知した場合の再接続試行間隔[ms]を設定する。  デフォルトは、0。 |
| 7 | ClientReconnectBufferTime | Long | クライアント側で接続切断を検知した場合に最後に受信したメッセージの受信時刻からどのくらいの時間[ms]だけ遡って再送要求を出すかを設定する。  デフォルトは、0[ms]で、最終受信時刻から再送要求する。 |
| 8 | ClientPacketRecycleBufferSize | Int | クライアント側で受信パケットを再利用する際の受信パケットバッファ数を設定する。  デフォルトは、10。 |
| 9 | ClientWindowRecycleBufferSize | Int | クライアント側で受信ウィンドウを再利用する際の受信ウィンドウバッファ数を設定する。  デフォルトは、200。 |
| 10 | ClientMessageRecycleBufferSize | Int | クライアント側で受信メッセージを再利用する際の受信メッセージバッファ数を設定する。  デフォルトは、100。 |
| 11 | MissingWindowTimeout | Int | クライアント側でロストしたと判断するまでのタイムアウトを設定する。  デフォルトは1000[ms]。 |
| 12 | MissingWindowCount | Int | クライアント側でロストしたと判断するまでの滞留件数を設定する。  デフォルトは0で、滞留件数ではロストしたと判断しない。 |
| 13 | NewMessagePollingInterval | Int | クライアント側から後続のメッセージが来ていないかサーバ側へポーリングする間隔を設定する。  デフォルトは5000[ms]。 |
| 14 | ClientResponseTimeout | Int | クライアントがサーバからの応答を待つタイムアウト[ms]を設定する。  デフォルトは、30秒。 |
| 15 | ServerAddress | String | サーバ側の接続待ち受けアドレスを設定する。  指定しない場合は、ローカルアドレス。 |
| 16 | NetworkInterfaces | String[] | ネットワークインタフェースを設定する。 |
| 17 | ServerPort | Int | サーバ側の接続待ち受けポート番号を設定する。  デフォルトは、0で任意のポート番号。 |
| 18 | ServerBacklog | int | サーバソケットのバックログを設定する。  デフォルトは0。 |
| 19 | ServerSocketFactoryServiceName | ServiceName | サーバソケットファクトリサービスのサービス名を設定する。 |
| 20 | SocketFactoryServiceName | ServiceName | ソケットファクトリサービスのサービス名を設定する。 |
| 21 | NIOSocketFactoryServiceName | ServiceName | サーバ側のソケットでjava.nioを使った接続を行う場合のサーバ側のソケットに適用するソケットファクトリサービスのサービス名を取得する。 |
| 22 | NIO | Boolean | サーバ側のソケットでjava.nioを使った接続を行うかどうかを設定する。 |
| 23 | LocalPort | Int | マルチキャスト配信を行う場合の送信ポート番号を設定する。  デフォルトは、0で任意のポート番号。  ユニキャスト配信でポート番号を指定した場合は、全てのクライアントに対する配信を同じ送信ソケットで配信する。 |
| 24 | MulticastGroupAddress | String | マルチキャスト配信を行う場合の配信アドレスを設定する。  マルチキャストアドレスまたは、ブロードキャストアドレスを設定する。設定しない場合は、UDPユニキャスト配信となる。 |
| 25 | MulticastPort | Int | マルチキャスト配信を行う場合の宛先ポート番号を設定する。  デフォルトは、2000。 |
| 26 | TimeToLive | Int | マルチキャスト配信を行う場合で、マルチキャストアドレスを指定した場合の、マルチキャストパケットの有効期間を設定する。  デフォルトは、1。 |
| 27 | UnicastPort | Int | ユニキャスト配信を行う場合の宛先ポート番号を設定する。  デフォルトは、0で任意のポート番号。 |
| 28 | WindowSize | Int | UDPパケットのサイズを設定する。  デフォルトは、1024[byte]。 |
| 29 | SendMessageCacheTime | long | 送信パケットのキャッシュ時間を設定する。  受信側がパケットをロストしたと判断した時に、送信側にロストしたパケットを問い合わせて補間する。  そのための送信側の補間用送信キャッシュに、送信したパケットを送信した時刻からどのくらいの間残しておくかを設定する。  受信側がパケットをロストしたと判断する時間は、MissingWindowTimeoutで設定するため、その時間より長く設定すべきである。  デフォルトは、5000[ms]。 |
| 30 | JndiName | String | ClientConnectionFactoryリモートオブジェクトのJNDI名を設定する。  デフォルトは、”publish/ClientConnectionFactory” |
| 31 | JndiRepositoryServiceName | ServiceName | ClientConnectionFactoryリモートオブジェクトをバインドするjp.ossc.nimbus.service.repository.Repositoryサービスのサービス名を設定する。 |
| 32 | RMIPort | Int | ClientConnectionFactoryリモートオブジェクトのRMI通信ポート番号を設定する。 |
| 33 | SendThreadSize | Int | ユニキャスト配信の場合で、同期送信をする際に、各送信先へ並列に送信処理を行うスレッドの数を設定する。  デフォルトは1で、各送信先へ直列に送信処理を行う。 |
| 34 | SendQueueServiceName | ServiceName | ユニキャスト配信の場合で、同期送信をする際に、各送信先への配信を並列送信するためのjp.ossc.nimbus.service.queue.Queueのサービス名を設定する。  SendThreadSizeに、2以上を指定しない場合は、並列送信しないため、Queueは使用しない。  以上を指定した場合で、この属性を指定しない場合は、内部でQueueを生成する。 |
| 35 | AsynchSendThreadSize | Int | 非同期送信をする際に、各送信先へ並列に送信処理を行うスレッドの数を設定する。  デフォルトは0で、非同期送信をサポートしない。 |
| 36 | AsynchSendQueueServiceName | ServiceName | 非同期送信をする際に、受け付けるためのjp.ossc.nimbus.service.queue.Queueサービス名を設定する。  AsynchSendThreadSizeに、1以上を指定しない場合は、非同期送信をサポートしないため、Queueは使用しない。  1以上を指定した場合で、この属性を指定しない場合は、内部でQueueを生成する。 |
| 37 | AsynchSendQueueFactoryServiceName | ServiceName | 非同期送信をする際に、並列送信を行うためのjp.ossc.nimbus.service.queue.Queue Queueのファクトリサービス名を設定する。  AsynchSendThreadSizeに、1以上を指定しない場合は、非同期送信をサポートしないため、Queueは使用しない。  1以上を指定した場合で、この属性を指定しない場合は、内部でQueueを生成する。 |
| 38 | RequestHandleQueueServiceName | ServiceName | クライアントからの要求を受け付けるjp.ossc.nimbus.service.queue.Queueサービスのサービス名を設定する。  RequestHandleThreadSizeに、1以上を指定しない場合は、クライアントからの要求の並列処理をサポートしないため、Queueは使用しない。  1以上を指定した場合で、この属性を指定しない場合は、内部でQueueを生成する。 |
| 39 | RequestHandleThreadSize | Int | クライアントからの要求の並列処理を行うスレッドの数を設定する。  デフォルトは0で、並列処理をサポートしない。 |
| 40 | MaxSendRetryCount | int | 送信に失敗した場合に再送信を行う回数を設定する。  デフォルトは、0で再送信しない。 |
| 41 | Acknowledge | Boolean | クライアントからの要求に対して、応答を返すかどうかを設定する。  デフォルトでは、falseで応答を返さない。 |
| 42 | ServerMessageRecycleBufferSize | Int | サーバ側で送信メッセージを再利用する際の送信メッセージバッファ数を設定する。  デフォルトは、100。 |
| 43 | ServerWindowRecycleBufferSize | Int | サーバ側で送信ウィンドウを再利用する際の送信ウィンドウバッファ数を設定する。  デフォルトは、200。 |
| 44 | ExternalizerServiceName | ServiceName | メッセージの直列化/非直列化に使用するjp.ossc.nimbus.service.io.Externalizerサービスのサービス名を設定する。  指定しない場合は、java.io.ObjectOutputStreamでメッセージの直列化/非直列化を行う。 |
| 45 | ServerConnectionListenerServiceNames | ServiceName[] | ServerConnectionListenerのサービス名を設定する。 |

### jp.ossc.nimbus.service.publish.DistributedConnectionFactoryService

複数のServerConnection で分散して送信速度を稼ぐServerConnectionを生成するServerConnectionFactoryインタフェースのデコレーション実装クラスである。また同時に、そのServerConnectionに接続するClientConnectionを生成するClientConnectionFactoryインタフェースの実装クラスでもある。デコレーション実装であるため、ServerConnectionFactory及びClientConnectionFactoryインタフェースを実装した実体となる通信プロトコル実装をインジェクションする必要がある。

このサービスは送信側に存在するため、ローカルでサービスを取得しServerConnectionFactoryにキャストして、ServerConnectionを取得する。

一方、このサービスは、RMI系のRepositoryインタフェース実装サービス（RMI実装や、JNDI実装が存在する）を設定する事で、ClientConnectionFactoryのリモートオブジェクトをバインドする事ができる。そのため、リモートの受信側では、RepositoryサービスからClientConnectionFactoryを取得し、RMIでClientConnectionを取得する。

但し、他のデコレーション系のServerConnectionFactoryと組み合わせて使用する場合、その実装に依存する。

受信側

送信側

Repository

ClientConnectionFactory

（サーバオブジェクト）

Repository

ClientConnectionFactory

（リモートオブジェクト）

lookup()

register()

**Distributed**

**ConnectionFactoryService**

ClientConnectionFactory

getClientConnection()

ServerConnectionFactory

getServerConnection()

onMessage()

send()

MessageListener

Message

Message

ClientConnection

ServerConnection

DistributedSize属性で指定された数だけ、実体となる通信プロトコル実装のServerConnection及びClientConnectionが生成される。送信するMessageは、Messageの持つキーによって分流されて、どれか１つのServerConnectionとClientConnetionのペアを使って送信される。

DistributedServerConnection

DistributedClientConnection

ClientConnection

ServerConnection

ClientConnection

ServerConnection

ClientConnection

ServerConnection

MessageListener

Message

Message

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | DistributedSize | int | 分散するコネクション数を設定する。  デフォルトは、1で分散しない。 |
| 2 | ConnectionFactoryFactoryServiceName | ServiceName | 実体となる通信プロトコル実装のServerConnectionFactory及びClientConnectionFactoryインタフェース実装サービスのサービス名を設定する。 |
| 3 | JndiName | String | ClientConnectionFactoryをバインドするJNDI名を設定する。  デフォルトは、”publish/ClientConnectionFactory”。 |
| 4 | JndiRepositoryServiceName | ServiceName | ClientConnectionFactoryリモートオブジェクトをバインドするjp.ossc.nimbus.service.repository.Repositoryサービスのサービス名を設定する。 |
| 5 | RMIPort | int | ClientConnectionFactoryリモートオブジェクトのRMI通信ポート番号を設定する。 |

### jp.ossc.nimbus.service.publish.GroupConnectionFactoryService

複数のServerConnection グルーピングして１つのServerConnectionにみせかけるServerConnectionFactoryインタフェースのデコレーション実装クラスである。また同時に、そのServerConnectionに接続するClientConnectionを生成するClientConnectionFactoryインタフェースの実装クラスでもある。デコレーション実装であるため、ServerConnectionFactory及びClientConnectionFactoryインタフェースを実装した実体となる通信プロトコル実装をインジェクションする必要がある。

このサービスは送信側に存在するため、ローカルでサービスを取得しServerConnectionFactoryにキャストして、ServerConnectionを取得する。

一方、このサービスは、RMI系のRepositoryインタフェース実装サービス（RMI実装や、JNDI実装が存在する）を設定する事で、ClientConnectionFactoryのリモートオブジェクトをバインドする事ができる。そのため、リモートの受信側では、RepositoryサービスからClientConnectionFactoryを取得し、RMIでClientConnectionを取得する。

但し、他のデコレーション系のServerConnectionFactoryと組み合わせて使用する場合、その実装に依存する。

受信側

送信側

ClientConnectionFactory

（リモートオブジェクト）

Repository

ClientConnectionFactory

（サーバオブジェクト）

Repository

lookup()

register()

**Group**

**ConnectionFactoryService**

ClientConnectionFactory

ClientConnectionFactory

ServerConnectionFactory

getClientConnection()

ServerConnectionFactory

getServerConnection()

onMessage()

send()

MessageListener

Message

Message

ClientConnection

ServerConnection

GroupConnectionFactoryService.SubjectMappingに、件名とキーの正規表現を設定し、それに対応するClientConnectionFactoryやServerConnectionFactoryを設定し、GroupConnectionFactoryServiceに登録する事で、送信するMessageを、どのServerConnectionを使って送信するかを振り分ける。

ServerConnectionFactoryだけを設定し、ServerConnectionのみグルーピングしたり、ClientConnectionFactoryだけを設定し、ClientConnectionのみグルーピングしたりすることもできる。

**GroupConnectionFactoryService**

SubjectMapping

SubjectMapping

件名

キー正規表現

ClientConnectionFactory

ServerConnectionFactory

getClientConnection()

getServerConnection()

GroupClientConnection

GroupServerConnection

ClientConnection

キー正規表現

ServerConnection

件名

キー正規表現

件名

ServerConnection

キー正規表現

件名

ServerConnection

ClientConnection

ClientConnection

send()

MessageListener

onMessage()

Message

Message

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | JndiName | String | ClientConnectionFactoryをバインドするJNDI名を設定する。  デフォルトは、”publish/ClientConnectionFactory”。 |
| 2 | JndiRepositoryServiceName | ServiceName | ClientConnectionFactoryリモートオブジェクトをバインドするjp.ossc.nimbus.service.repository.Repositoryサービスのサービス名を設定する。 |
| 3 | RMIPort | int | ClientConnectionFactoryリモートオブジェクトのRMI通信ポート番号を設定する。 |
| 4 | SubjectMapping.Subject | String | このSubjectMappingに紐づくServerConnectionで配信するMessageの件名を設定する。 |
| 5 | SubjectMapping.Key | String | このSubjectMappingに紐づくServerConnectionで配信するMessageのキーの正規表現を設定する。  指定しない場合は、すべてのキーが対象となる。 |
| 6 | SubjectMapping.ClientConnectionFactoryJndiName | String | グルーピングするClientConnectionFactoryのJNDI名を設定する。 |
| 7 | SubjectMapping.ClientConnectionFactoryServiceName | ServiceName | グルーピングするClientConnectionFactoryのサービス名を設定する。 |
| 8 | SubjectMapping.ServerConnectionFactoryServiceName | ServiceName | グルーピングするServerConnectionFactoryのサービス名を設定する。 |

### jp.ossc.nimbus.service.publish.ThinOutServerConnectionFactoryService

送信するMessageに間引き処理を付与するServerConnectionFactoryインタフェースのデコレーション実装クラスである。デコレーション実装であるため、ServerConnectionFactoryインタフェースを実装した実体となる通信プロトコル実装をインジェクションする必要がある。また、間引き処理を行うThinOutFilterを複数インジェクションして、間引き処理を付与することができる。

このサービスは送信側に存在するため、ローカルでサービスを取得しServerConnectionFactoryにキャストして、ServerConnectionを取得する。

**ThinOutServerConnection**

**FactoryService**

ServerConnection

ThinOutFilter

ThinOutServerConnection

ThinOutFilter

ThinOutFilter

ServerConnectionFactory

ThinOutFilter

Message

getServerConnection()

send()

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ServerConnectionFactoryServiceName | ServiceName | 実体となる通信プロトコル実装のServerConnectionFactoryインタフェース実装サービスのサービス名を設定する。 |
| 2 | ThinOutFilterServiceNames | ServiceName[] | 間引き処理を行うThinOutFilterインタフェース実装サービスのサービス名を設定する。 |
| 3 | ThinOutTimeoutCheckInterval | long | 間引き対象となったメッセージが最後のメッセージだった場合に送信するための監視間隔[ms]を設定する。  デフォルトは、1秒。 |
| 4 | ThinOutTimeout | long | 間引き対象となったメッセージが最後のメッセージだった場合に送信するための、間引きタイムアウト[ms]を設定する。  デフォルトは、3秒。 |

### jp.ossc.nimbus.service.publish.RequestConnectionFactoryService

メッセージの送受信を行うRequestServerConnectionを生成するServerConnectionFactoryインタフェースのデコレーション実装クラスである。また、RequestServerConnectionから送信されたMessageを受け取り応答するRequestMessageListenerを登録する必要があるため、MessageReceiverインタフェースの実装クラスでもある。送受信双方向のデコレーション実装であるため、送信のためのServerConnectionFactoryインタフェースを実装した実体となる通信プロトコル実装と、受信のためのMessageReceiverインタフェース実装をインジェクションする必要がある。

このサービスは送受信双方に存在するため、送信側では、ローカルでサービスを取得しServerConnectionFactoryにキャストして、ServerConnectionを取得し、さらにRequestServerConnectionにキャストして使用する。また、受信側では、ローカルでサービスを取得しMessageReceiverにキャストして、RequestMessageListenerを登録する。

**RequestConnectionFactoryService**

MessageReceiver

ServerConnectionFactory

ClientConnectionFactory

RequestMessageListener

getServerConnection()

request()

RequestServerConnection

ClientConnection

Message

getClientConnection()

ServerConnection

ServerConnection

RequestServerConnection

ClientConnection

MessageReceiver

onRequestMessage()

Message

**RequestConnectionFactoryService**

ClientConnectionFactory

RequestMessageListener

ServerConnectionFactory

RequestServerConnectionでMessageを送受信するが、同期的に応答を待つメソッド、送信して後で応答を取得する非同期的に応答を待つメソッド、送信して応答をコールバックするメソッドの３種類のオーバーロードメソッドがある。

要求を受信する側では、RequestMessageListenerのonRequestMessage()で要求を受信して、応答のMessageを返す。但し、応答を非同期で返したい場合は、onRequestMessage()でnullを返し、後でRequestServerConnectionのresponse()で応答を返す。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ServerConnectionFactoryServiceName | ServiceName | 要求メッセージ及び応答メッセージの送信に使用するServerConnectionFactoryサービスのサービス名を設定する。 |
| 2 | MessageReceiverServiceName | ServiceName | 要求メッセージ及び応答メッセージの受信に使用するMessageReceiverサービスのサービス名を設定する。 |
| 3 | AsynchResponse | boolean | 応答メッセージの送信を非同期で行うかどうかを設定する。  デフォルトは、falseで同期送信する。 |
| 4 | ResponseRetryCount | int | 応答メッセージの送信のリトライ回数を設定する。  デフォルトは、1。 |
| 5 | ResponseRetryInterval | long | 応答メッセージの送信のリトライ間隔[ms]を設定する。  デフォルトは、50[ms]。 |

## ClientConnectionFactory

### jp.ossc.nimbus.service.publish.tcp.ConnectionFactoryService

TCPプロトコルで通信するClientConnectionを生成するClientConnectionFactoryインタフェースの実装クラスである。また同時に、TCPプロトコルで通信するServerConnectionを生成するServerConnectionFactoryインタフェースの実装クラスでもある。

詳細は、3.1.1を参照。

### jp.ossc.nimbus.service.publish.udp.ConnectionFactoryService

UDPプロトコルで通信するClientConnectionを生成するClientConnectionFactoryインタフェースの実装クラスである。また同時に、UDPプロトコルで通信するServerConnectionを生成するServerConnectionFactoryインタフェースの実装クラスでもある。

詳細は、3.1.2を参照。

### jp.ossc.nimbus.service.publish.ClusterConnectionFactoryService

Java VMを跨いで存在する複数のServerConnectionをクラスタ化して接続するClientConnectionを生成するClientConnectionFactoryインタフェースのデコレーション実装クラスである。

サービスの設定によって、以下のクラスタ構成が選択できる。

|  |  |  |
| --- | --- | --- |
| # | クラスタ構成 | 説明 |
| 1 | First Available | 最初のクラスタメンバのClientConnectionに接続する。  接続中のClientConnectionが、クラスタメンバからいなくなった場合は、次のクラスタメンバのClientConnectionに再接続する。  デフォルトの設定。 |
| 2 | Distribute Available | クラスタメンバ内で、最も接続数が少ないClientConnectionに接続する。  接続中のClientConnectionが、クラスタメンバからいなくなった場合は、次のクラスタメンバのClientConnectionに再接続する。 |
| 3 | All Available | クラスタメンバ内の全てのClientConnectionに接続する。  新しいクラスタメンバが増えた場合は、そのClientConnectionにも接続する。 |

このサービスは送信側に存在し、RMI系のRepositoryインタフェース実装サービス（RMI実装や、JNDI実装が存在する）を設定する事で、ClientConnectionFactoryのリモートオブジェクトをバインドする事ができる。そのため、リモートの受信側では、RepositoryサービスからClientConnectionFactoryを取得し、RMIでClientConnectionを取得する。但し、他のデコレーション系のServerConnectionFactoryと組み合わせて使用する場合、その実装に依存する。

受信側

送信側

送信側

Repository

ClientConnectionFactory

（サーバオブジェクト）

Repository

ClientConnectionFactory

（リモートオブジェクト）

lookup()

register()

**Cluster**

**ConnectionFactoryService**

getClientConnection()

ClusterClientConnection

ClientConnectionFactory

ClusterService

ClusterService

ClusterService

ClientConnectionFactory

（リモートオブジェクト）

ClientConnectionFactory

（リモートオブジェクト）

ClientConnectionFactory

（リモートオブジェクト）

ServerConnectionFactory

getClientConnection()

ServerConnection

Message

send()

getServerConnection()

ClientConnection

ClientConnection

ServerConnection

MessageListener

Message

onMessage()

受信側でClientConnectionFactoryから取得したClientConnectionに接続すると、内部にClusterServiceが生成され、送信側のClusterServiceと通信を行いクラスタメンバが共有される。クラスタメンバのオプション情報から、各送信側のClientConnectionFactoryのリモートオブジェクトが取得できるため、設定されているクラスタ構成に従って、実体となるClientConnectionを取得して接続を行う。

また、送信側のサーバが増減すると、ClusterServiceがクラスタメンバの変更を検知し、設定されているクラスタ構成に従って、ClientConnectionの再接続を行う。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | JndiName | String | ClientConnectionFactoryをバインドするJNDI名を設定する。  デフォルトは、”publish/ClientConnectionFactory”。 |
| 2 | JndiRepositoryServiceName | ServiceName | ClientConnectionFactoryリモートオブジェクトをバインドするjp.ossc.nimbus.service.repository.Repositoryサービスのサービス名を設定する。 |
| 3 | RMIPort | int | ClientConnectionFactoryリモートオブジェクトのRMI通信ポート番号を設定する。 |
| 4 | ClusterServiceName | ServiceName | jp.ossc.nimbus.service.keepalive.ClusterServiceサービスのサービス名を設定する。 |
| 5 | ClientConnectionFactoryServiceName | ServiceName | クラスタ化したいClientConnectionFactoryサービスのサービス名を設定する。 |
| 6 | Distribute | boolean | クラスタ構成をDistribute Availableに設定する。  デフォルトは、false |
| 7 | Multiple | boolean | クラスタ構成をAll Availableに設定する。  デフォルトは、false |
| 8 | ReceiveOwnMessage | boolean | 自分自身からのメッセージを受信するかどうかを設定する。  デフォルトは、falseで受信しない。 |
| 9 | FlexibleConnect | boolean | 接続時に、クラスタメンバが存在しなくても接続可能な柔軟な接続とするかどうかを設定する。  デフォルトは、falseで、クラスタメンバが存在しない場合は、接続できない。 |
| 10 | FailoverBufferTime | long | フェイルオーバー時に最後に受信したメッセージの受信時刻からどのくらいの時間[ms]だけ遡って再送要求を出すかを設定する。  デフォルトは、0[ms]で最終受信時刻から再送要求する。 |

### jp.ossc.nimbus.service.publish.ClusterClientConnectionFactoryService

送信側のClusterConnectionFactoryServiceが存在するJavaVMを意識せずに、透過的に接続可能なClientConnectionFactoryインタフェースのデコレーション実装クラスである。

このサービスは受信側に存在するため、ローカルでサービスを取得しClientConnectionFactoryにキャストして、ClientConnectionを取得する。ClientConnectionを接続すると、送信側のClusterConnectionFactoryServiceが提供するClientConnectionFactory をClusterServiceから取得して、実体となるClientConnectionを取得するため、送信側がどのサーバに存在するかを意識する必要がない。

送信側

送信側

受信側

ClusterService

ClientConnectionFactory

Cluster

ConnectionFactoryService

ClusterService

ClientConnectionFactory

（リモートオブジェクト）

**ClusterClient**

**ConnectionFactoryService**

ClusterService

ClientConnectionFactory

（リモートオブジェクト）

ClientConnectionFactory

（リモートオブジェクト）

getClientConnection()

ServerConnection

Message

send()

getServerConnection()

ServerConnectionFactory

ServerConnection

ClusterClientConnection

getClientConnection()

ClientConnection

ClientConnection

MessageListener

onMessage()

Message

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ClusterServiceName | ServiceName | jp.ossc.nimbus.service.keepalive.ClusterServiceサービスのサービス名を設定する。 |
| 2 | FlexibleConnect | boolean | 接続時に、クラスタメンバが存在しなくても接続可能な柔軟な接続とするかどうかを設定する。  デフォルトは、falseで、クラスタメンバが存在しない場合は、接続できない。 |

### jp.ossc.nimbus.service.publish.DistributedConnectionFactoryService

複数のClientConnection で分散して受信速度を稼ぐClientConnectionを生成するClientConnectionFactoryインタフェースのデコレーション実装クラスである。また同時に、そのClientConnectionから接続するServerConnectionを生成するServerConnectionFactoryインタフェースの実装クラスでもある。

詳細は、3.1.3を参照。

### jp.ossc.nimbus.service.publish.GroupConnectionFactoryService

グルーピングされた複数のServerConnection に接続するClientConnectionを生成するClientConnectionFactoryインタフェースのデコレーション実装クラスである。また同時に、そのClientConnectionから接続するServerConnectionを生成するServerConnectionFactoryインタフェースの実装クラスでもある。

詳細は、3.1.4を参照。

### jp.ossc.nimbus.service.publish.GroupClientConnectionFactoryService

複数のClientConnection をグルーピングして１つのClientConnectionにみせかけるClientConnectionFactoryインタフェースのデコレーション実装クラスである。デコレーション実装であるため、ClientConnectionFactoryインタフェースを実装した実体となる通信プロトコル実装をインジェクションする必要がある。

このサービスは受信側に存在するため、ローカルでサービスを取得しClientConnectionFactoryにキャストして、ClientConnectionを取得する。

GroupClientConnectionFactoryService.SubjectMappingに、件名とキーの正規表現を設定し、それに対応するClientConnectionFactoryを設定し、GroupClientConnectionFactoryServiceに登録する事で、受信するMessageをどの件名及びキーなら、どのClientConnectionを使って受信するかを振り分ける。ClientConnectionFactoryは、ロカール及びRepositoryから取得可能である。

**GroupClientConnectionFactoryService**

ClientConnectionFactory

キー正規表現

件名

SubjectMapping

SubjectMapping

getClientConnection()

SubjectMapping

Message

onMessage()

MessageListener

ClientConnection

件名

キー正規表現

ClientConnection

件名

キー正規表現

件名

ClientConnection

キー正規表現

GroupClientConnection

ClientConnectionFactory

（リモートオブジェクト）

lookup()

Repository

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | JndiRepositoryServiceName | ServiceName | グルーピングするClientConnectionFactoryをlookupするjp.ossc.nimbus.service.repository.Repositoryサービスのサービス名を設定する。 |
| 2 | SubjectMapping.Subject | String | このSubjectMappingに紐づくClientConnectionで受信するMessageの件名を設定する。 |
| 3 | SubjectMapping.Key | String | このSubjectMappingに紐づくClientConnectionで受信するMessageのキーの正規表現を設定する。  指定しない場合は、すべてのキーが対象となる。 |
| 4 | SubjectMapping.ClientConnectionFactoryJndiName | String | グルーピングするClientConnectionFactoryのJNDI名を設定する。 |
| 5 | SubjectMapping.ClientConnectionFactoryServiceName | ServiceName | グルーピングするClientConnectionFactoryのサービス名を設定する。 |
| 6 | SubjectMapping.JndiRepositoryServiceName | ServiceName | このSubjectMappingに紐づくClientConnectionFactoryをlookupするjp.ossc.nimbus.service.repository.Repositoryサービスのサービス名を設定する。 |

## ThinOutFilter

### jp.ossc.nimbus.service.publish.SubjectThinOutFilterService

Messageの件名で間引きするThinOutFilterインタフェースの実装クラスである。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | FilterSubjects | String[] | 間引く件名を設定する。 |
| 2 | NotFilterSubjects | String[] | 間引かない件名を設定する。 |

### jp.ossc.nimbus.service.publish.TimeIntervalThinOutFilterService

Messageの送信時刻を元に時間間隔で間引きするThinOutFilterインタフェースの実装クラスである。

同じキーを持つMessageが、一定時間内に１件しか送信されないように間引き処理を行う。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ThinOutInterval | long | 間引き間隔[ms]を設定する。  デフォルトは、1秒。 |

## MessageReceiver

### jp.ossc.nimbus.service.publish.MessageReceiverService

MessageReceiverインタフェースのデフォルト実装クラスである。

**MessageReceiverService**

ClientConnectionFactory

MessageListener

MessageListener

配信スレッド

受信フィルタ

受信スレッド

受信スレッド

受信スレッド

ClientConnection

MessageListener

MessageListener

配信スレッド

MessageListener

MessageListener

配信スレッド

MessageListener

MessageListener

配信スレッド

宛先

キー

MessageListener

MessageListener

宛先

キー

MessageListener

MessageListener

ClientConnectionFactoryからClientConnectionを取得して、自分自身をMessageListenerとして設定する。ClientConnectionから受信したMessageを、キー単位に分流して受信キューに投入する。

それぞれの受信キューに対して、受信スレッドが存在し、Messageの受信処理を並列に行う。受信処理では、各MessageListenerに指定された受信する宛先とキーの集合と、Messageのマッチングを行い、受信するMessageListenerを特定する。その後、MessageListener毎に分流して、配信キューに投入する。

それぞれの配信キューに対して、配信スレッドが存在し、MessageをMessageListenerに配信する処理を並列に行う。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ClientConnectionFactoryJndiName | String | ClientConnectionFactoryリモートオブジェクトのJNDI名を設定する。 |
| 2 | JndiRepositoryServiceName | ServiceName | ClientConnectionFactoryリモートオブジェクトがバインドされているjp.ossc.nimbus.service.repository.Repository Repositoryサービスのサービス名を設定する。 |
| 3 | ClientConnectionFactoryServiceName | ServiceName | ClientConnectionFactoryサービスのサービス名を設定する。 |
| 4 | MessageQueueFactoryServiceName | ServiceName | 受信キューに使用するjp.ossc.nimbus.service.queue.Queueサービスを生成するファクトリサービスのサービス名を設定する。  設定しない場合は、内部でQueueが生成される。 |
| 5 | MessageQueueDistributedSize | int | 受信キューの分流数を設定する。  デフォルトは、0で、そのまま受信処理を行う。 |
| 6 | MessageListenerQueueFactoryServiceName | ServiceName | 配信キューに使用するjp.ossc.nimbus.service.queue.Queueサービスを生成するファクトリサービスのサービス名を設定する。  設定しない場合は、内部でQueueが生成される。 |
| 7 | MessageListenerQueueDistributedSize | int | 配信キューの分流数を設定する。  デフォルトは、0で、そのまま配信処理を行う。 |
| 8 | ConnectOnStart | boolean | サービスの開始時に、ClientConnectionを接続するかどうかを設定する。 |
| 9 | StartReceiveOnStart | boolean | サービスの開始時に、ClientConnectionから受信を開始するかどうかを設定する。 |

### jp.ossc.nimbus.service.publish.MessageForwardService

MessageReceiverServiceのサブクラスで、ClientConnectionから受信したMessageをServerConnectionへ転送する MessageReceiverインタフェースの実装クラスである。

**MessageForwardService**

ClientConnectionFactory

MessageListener

MessageListener

配信スレッド

Client

Client

受信フィルタ

受信スレッド

受信スレッド

受信スレッド

ClientConnection

MessageListener

MessageListener

配信スレッド

Client

Client

MessageListener

MessageListener

配信スレッド

Client

Client

MessageListener

MessageListener

配信スレッド

Client

Client

ServerConnectionFactory

宛先

キー

MessageListener

MessageListener

宛先

キー

MessageListener

MessageListener

ServerConnectionListener

ServerConnection

ServerConnectionFactoryからServerConnectionを取得して、ServerConnectionListenerを設定する。クライアント側から、接続要求が送信されると、ServerConnectionListenerでイベント通知を受け、Clientに対応するMessageListenerを生成する。続いて、クライアント側から件名及びキーの登録要求が送信されると、ServerConnectionListenerでイベント通知を受け、Clientに対応するMessageListenerを自身に登録し、ClientConnectionにも件名及びキーを登録する。

ClientConnectionFactoryからClientConnectionを取得して、自分自身をMessageListenerとして設定する。ClientConnectionから受信したMessageを、キー単位に分流して受信キューに投入する。

それぞれの受信キューに対して、受信スレッドが存在し、Messageの受信処理を並列に行う。受信処理では、各MessageListenerに指定された受信する宛先とキーの集合と、Messageのマッチングを行い、受信するMessageListenerを特定する。その後、MessageListener毎に分流して、配信キューに投入する。

それぞれの配信キューに対して、配信スレッドが存在し、MessageをMessageListenerに配信する処理を並列に行う。MessageListenerでは、ServerConnectionに送信を行う。

以下にサービスの属性一覧を示す。

|  |  |  |  |
| --- | --- | --- | --- |
| # | 属性名 | 型 | 説明 |
| 1 | ClientConnectionFactoryJndiName | String | ClientConnectionFactoryリモートオブジェクトのJNDI名を設定する。 |
| 2 | JndiRepositoryServiceName | ServiceName | ClientConnectionFactoryリモートオブジェクトがバインドされているjp.ossc.nimbus.service.repository.Repository Repositoryサービスのサービス名を設定する。 |
| 3 | ClientConnectionFactoryServiceName | ServiceName | ClientConnectionFactoryサービスのサービス名を設定する。 |
| 4 | MessageQueueFactoryServiceName | ServiceName | 受信キューに使用するjp.ossc.nimbus.service.queue.Queueサービスを生成するファクトリサービスのサービス名を設定する。  設定しない場合は、内部でQueueが生成される。 |
| 5 | MessageQueueDistributedSize | int | 受信キューの分流数を設定する。  デフォルトは、0で、そのまま受信処理を行う。 |
| 6 | MessageListenerQueueFactoryServiceName | ServiceName | 配信キューに使用するjp.ossc.nimbus.service.queue.Queueサービスを生成するファクトリサービスのサービス名を設定する。  設定しない場合は、内部でQueueが生成される。 |
| 7 | MessageListenerQueueDistributedSize | int | 配信キューの分流数を設定する。  デフォルトは、0で、そのまま配信処理を行う。 |
| 8 | ConnectOnStart | boolean | サービスの開始時に、ClientConnectionを接続するかどうかを設定する。 |
| 9 | StartReceiveOnStart | boolean | サービスの開始時に、ClientConnectionから受信を開始するかどうかを設定する。 |
| 10 | ServerConnectionFactoryServiceName | ServiceName | 転送先のServerConnectionFactoryサービスのサービス名を設定する。 |
| 11 | AsynchSend | boolean | 転送先のServerConnectionに非同期送信するかどうかを設定する。  デフォルトは、falseで同期送信。 |