Borland VisiBroker™ 8.0

GateKeeper ガイド

Borland Software Corporation
20450 Stevens Creek Blvd., Suite 800
Cupertino, CA 95014 USA
www.borland.com

VB80 GateKeeper ガイド
2007 年 4 月

Borland

目次

第 1 章

Gatekeeper の概要

ここでは、Gatekeeper の概要と Gatekeeper のさまざまな起動方法について説明します。

GateKeeper の概要

Gatekeeper は Borland Software Corporation が開発した OMG-CORBA 準拠の GIOP プロキシサーバーです。これは、インターネットブラウザ、ファイアウォール、Java サンドボックスセキュリティによるセキュリティ制約の下で、CORBA クライアントと CORBA サーバーがネットワークを介して通信するためのサーバーです。したがって、セキュリティ制約のために、クライアントがサーバーと直接通信できない場合は、Gatekeeper がクライアントからサーバーへのゲートウェイまたはプロキシを提供します。

サーバーを直接クライアントに開放したくない場合、あるいはクライアントによるサーバーへのアクセスが制約されている場合に、Gatekeeper を使用することになります。後者は、クライアントが署名のないアプレットであるか、間にファイアウォールがあるかのどちらかです。

ゲートウェイまたはプロキシとしての Gatekeeper

VisiBroker ORB に基づく分散システムをインターネットまたはイントラネットを介してデプロイメントする場合は、次のような多くのセキュリティ制約をシステムに適用できます。

- 一定のホストに対するクライアントからのアクセスを抑止するサーバー側ファイアウォール。
- 発信接続を抑止するクライアント側ファイアウォール。
- HTTP 以外のプロトコルを禁止するクライアント側ファイアウォール。

Gatekeeper は、VisiBroker ORB の下でクライアントおよびサーバー間のゲートウェイまたはプロキシとして動作します。これにより、OMG CORBA ファイアウォール仕様に基づいて上記の制限を機能させるメカニズムを提供します。特定の制約があるためにクライアントがサーバーに直接には接続できない場合、クライアントは Gatekeeper への接続を選択できます。クライアントは Gatekeeper にメッセージを送信し、Gatekeeper はサーバーにそのメッセージを転送します。

また、特定の制約があるためにサーバーがクライアントに接続してコールバックを戻すことができない場合、サーバーは Gatekeeper への接続を選択します。サーバーは Gatekeeper にコールバックメッセージを送信し、Gatekeeper はクライアントにそのメッセージを転送します。

これをまとめると、Gatekeeper には次の機能があります。

- ファイアウォールを通過できるプロキシ
- コールバック機能
- ネットワーク上の位置透過性

Java
- HTTP トンネリング

Gatekeeper の追加機能

GateKeeper のその他の機能について簡単に紹介します。

Java
- Java クラスをロードする簡単な Web サーバーとして機能します。Java サンドボックスセキュリティは、署名のない Java アプレットに対して、そのアプレットがダウンロードされたホストマシン上で実行されているサーバー以外とは、通信できないようにします。この問題は、Gatekeeper を設定することで解決できます。

Java
- ブートストラップ。Gatekeeper は、サーブレットをサポートする任意の Web サーバーの内部でサーブレットとして実行できます。この設定により、HTTP（HIOP）の IIOP が有効になり、Java クライアントで使用することができます。
- 負荷分散とフォールトトレランス。マスター Gatekeeper と 1 つ以上のスレーブ Gatekeeper をまとめてクラスタリングし、クライアントに対して 1 つの Gatekeeper として見せることができます。この設定は、負荷分散に柔軟性を与え、一定レベルのフォールトトレランスを実現できます。
- カスタマイズ可能な IP ベースのアクセスコントロール。オペレーション、署名などの条件に基づいて、アクセスを拒否または許可するように Gatekeeper を設定できます。

メモ GateKeeper の設定については、第 4 章「高度な機能」を参照してください。

Gatekeeper の主な用途

Gatekeeper は主に、ファイアウォールや転送の制約に対処するためにプロキシとして利用します。また Gatekeeper は Web サーバーとして機能し、負荷分散やアクセスコントロールの機能も取り入れています。ただし、Gatekeeper を本格的な Web サーバー、本格的な負荷分散システム、本格的なアクセスコントロールシステムとしては使用しないでください。Gatekeeper はそれらのシステムをサポートするようにデザインされています。

Gatekeeper のインストール

Gatekeeper は、VisiBroker のコンポーネントとして提供されています。Gatekeeper の動作には、次のコンポーネントが必要です。

- VisiBroker スマートエージェント
- VisiBroker ORB ライブラリ
- VisiBroker Gatekeeper プロパティファイル
- VisiBroker コンソール

メモ Gatekeeper はスタンドアロンのプロセスです。CORBA IDL コンパイラは必要ありません。

Gatekeeper の起動

GateKeeper を起動するディレクトリは、GateKeeper の使用方法によって決まります。

- ファイアウォールに対する IIOP プロキシとして
- HIOP をサポートする Web サーバーとして
- VisiBroker for Java で HIOP をサポートする別の Web サーバーと組み合わせて

通常は、ファイアウォールの管理者がプロキシの責任者になるため、Gatekeeper を IIOP プロキシとして使用する場合は、ファイアウォールの管理者に問い合わせてください。

Gatekeeper を補足的な Web サーバーとして使用する場合は、Gatekeeper を Java アプレットのコードベースと同じディレクトリで起動することをお勧めします。前述の最初の 2 つの機能では、Gatekeeper をコマンドラインで起動したり、Windows/NT サービスとして実行できます。

Gatekeeper を別の Web サーバーと組み合わせて使用する場合は、その Web サーバーでサーブレットとして Gatekeeper を起動できます。

GateKeeper をコマンドラインから起動する

Gatekeeper を起動するには、次のコマンドを使用します。

```
prompt> gatekeeper
```

メモ GateKeeper をコマンドラインから起動するには、CLASSPATH 設定のパスに servlet.jar が含まれている必要があります。
servlet.jar は、VisiBroker に付属してインストールされる Tomcat の下にあります。次に例を示します。

```
<intalldir>/lib/tomcat/common
```

ここで、<intalldir> は、VisiBroker がインストールされているルートディレクトリの場所を表します。たとえば、Windows では C:¥visibroker です。

たとえば、Windows では、CLASSPATH を環境変数として指定し、検索パスに servlet.jar を入れます。

Gatekeeper を起動すると、起動メッセージに続いて、起動中のサービスを示す一連のメッセージが表示されます。このメッセージの例を次に示します。

```
Sun Feb 16 23:43:28 2003: Starting GateKeeper for VisiBroker ...
Sun Feb 16 23:43:31 2003: Request Forwarding Service is started.
Sun Feb 16 23:43:31 2003: Administrative Service is started.
Sun Feb 16 23:43:31 2003: IOR is stored in GateKeeper.ior.
Sun Feb 16 23:43:31 2003: GateKeeper for VisiBroker is started.
```

コマンドラインオプション

gatekeeper コマンドを使用する際は、次のコマンドラインオプションを利用できます。

Option	説明
-props file_name	Gatekeeper のプロパティファイルの名前を示します。ファイル名を指定する場合は、フルパスを入力できます。 このファイルのデフォルトの場所は、Gatekeeper をインストールしたディレクトリです。このファイルのデフォルトの名前は GateKeeper.properties です。
-J-D<Property-name>=<value>	起動時に Gatekeeper のプロパティを指定します。
-h、-help、-usage、-?	コマンドの使用方法を表示します。
-quiet	GateKeeper が出力を生成しないように指定します。

Gatekeeper を NT サービスとして実行する

Gatekeeper は、NT サービスとしてインストールできます。ただし、その前に、目的の NT プラットフォームで DOS プロンプトから Gatekeeper を実行できることを確認してください。

Gatekeeper を NT サービスとしてインストールするには、コマンドラインで次のコマンドを入力します。ここで、servicename は、インストールする Gatekeeper の名前です。

```
gatekeeper -install "servicename"
```

-props オプションでプロパティファイルを指定する場合は、指定するプロパティファイルのフルパス名を入力してください。

Gatekeeper を NT サービスとしてインストールした後は、標準の［Services］コントロールパネルから Gatekeeper を起動できます。

Gatekeeper を NT サービスとして削除する

GateKeeper NT サービスを削除するには、コマンドプロンプトで次の構文を使用します。

```
gatekeeper -remove "servicename"
```

Gatekeeper を Web サーバーでサーブレットとして実行する

Gatekeeper は、サーブレットをサポートする任意の Web サーバーの内部でサーブレットとして実行できます。Gatekeeper は、正しい HIOP コンポーネントを Gatekeeper の IOR に生成することを目的とした特別な HIOP リスナーとして起動します。HIOP コンポーネントには、Web サーバーのホスト、ポート、および Gatekeeper サーブレットへのパスを組み込みます。クライアントは、HIOP コンポーネント内で指定されている Gatekeeper に HIOP 要求を送信します。この機能の利点は、デプロイメントやパッケージングの際に、Web サーバーや Borland パーティションなどのシステムに組み込むほかのコンポーネントと密接に統合できるところです。

通常、Web サーバーでサーブレットとして Gatekeeper を実行しても、パフォーマンスの大幅な向上は期待できません。トンネリングされたすべての要求は、引き続き Gatekeeper がスタンドアロンで実行されている場合と同じように Gatekeeper を通過するからです。

メモ コマンドラインからではなく、サーブレットとして Gatekeeper を実行する場合は、管理機能の一部のほか、Gatekeeper の出力機能を利用できなくなります。

Gatekeeper の管理

VisiBroker コンソールから、ネットワークシステムの条件に応じた Gatekeeper のプロパティを設定できます。Gatekeeper のプロパティはプロパティファイルに保存され、Gatekeeper は起動時にこのファイルを参照します。

第 2 章

GateKeeper とネットワーク間デバイスの設定

ここでは、Gatekeeper とネットワーク間デバイスを設定して、ネットワーク間でクライアントオブジェクトとサーバーオブジェクトの間の通信を可能にする方法について説明します。まず、Gatekeeper をデプロイメントする場所について説明します。

GateKeeper のデプロイメント場所

ここでは、Gatekeeper に適したデプロイメント場所を特定するための基本ポイントをいくつか説明します。

次の情報を用意します。

- クライアントの場所
- サーバーの場所とサーバーのリスナーポート
- クライアントとサーバーを接続するネットワーク
- 接続ネットワーク上のファイアウォール、ルーター、およびゲートウェイの設定

クライアントとサーバー間の接続パスを探します。パスは複数のネットワークにまたがる場合があります。クライアントがサーバーに接続できるようにするには、接続パスが必要です。接続パスがなければ、クライアントはサーバーと通信できません。

同じネットワーク上のクライアントとサーバー

クライアントとサーバーが同一ネットワーク上に存在する場合、クライアントは常にサーバーと直接通信できます。ただし、次に示すケース 1 と 2 の例のように、状況によっては、GateKeeper が必要な場合があります。GateKeeper が必要な場合は、同じネットワーク上の任意のホストに GateKeeper をデプロイメントします。

ケース 1：クライアントのトランスポートの種類に制限がある場合

クライアント側のプロパティを使用して、クライアントがサーバーとの接続に使用できるトランスポートを制限できます。次の場合は、Gatekeeper が必要です。

- クライアントが常にプロキシを介して接続する（vbroker.orb.alwaysProxy）
- クライアントが常に HTTP トンネリングモードを使用する（vbroker.orb.alwaysTunnel）

詳細は、23 ページの「クライアント プロパティの設定」を参照してください。

ケース 2: Java サンドボックスセキュリティ

Java サンドボックスセキュリティは、署名のない Java アプレットが、そのアプレットのダウンロード元のホストで実行されているサーバー以外のサーバー上にあるサーバーオブジェクトと通信できないようにします。この場合は、Java サンドボックスセキュリティ制限に対応するために、クライアントとサーバー間のゲートウェイとして Gatekeeper が必要です。

隣接したネットワーク上のクライアントとサーバー

クライアントとサーバーのネットワークどうしが互いに隣接している場合は、ゲートウェイやルーターなどのネットワーク間デバイスを使用して接続します。場合によっては、どちらかのネットワークまたは両方のネットワークにファイアウォールが存在します。説明をわかりやすくするため、ここではファイアウォールをネットワーク間デバイスの一部と仮定します。ネットワーク間デバイスは 2 つのネットワーク間のメッセージ転送とルーティングを管理します。また、特定のメッセージを別のネットワークに移動できないようにブロックすることもできます。これがファイアウォールの役割です。クライアントのプロパティを使用して、クライアントがサーバーとの接続に使用するトランスポートを制限できます。

次の場合は、Gatekeeper が必要です。

- クライアントが常にプロキシを介して接続する
- クライアントが常に HTTP トンネリングモードを使用する

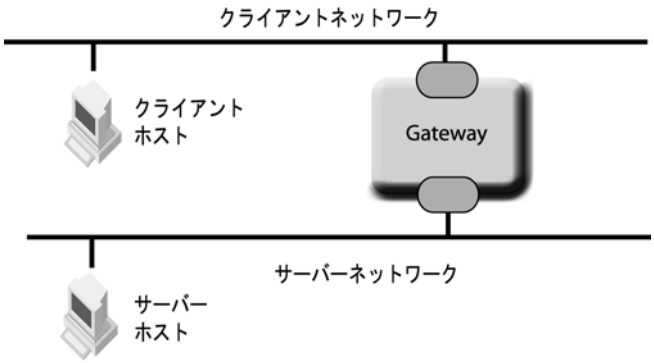
詳細は、23 ページの「クライアント プロパティの設定」を参照してください。

ケース 1：Java サンドボックスセキュリティ

Java サンドボックスセキュリティは、署名のない Java アプレットが、そのアプレットのダウンロード元のホストで実行されているサーバー以外のサーバー上にあるサーバーオブジェクトと通信できないようにします。この場合は、Java サンドボックスセキュリティ制限に対応するために、クライアントとサーバー間のゲートウェイとして Gatekeeper が必要です。

次の図は、隣接したネットワーク上のクライアントとサーバーです。

図 2.1　隣接したネットワーク上のクライアントとサーバー



ケース 1：クライアントのトランスポートの種類に制限がある場合

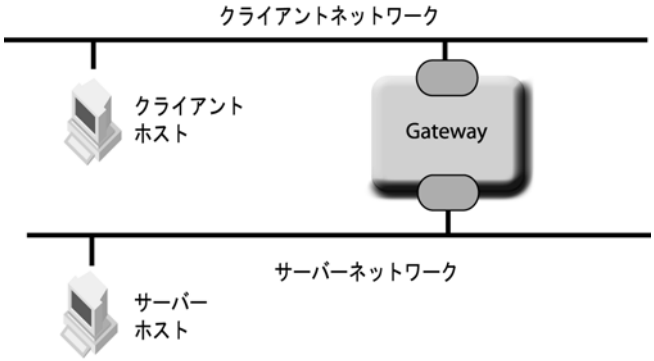
クライアントのメッセージをサーバーに届けるためには、ネットワーク間デバイスが、メッセージをクライアントネットワークからサーバーネットワークに転送する必要があります。Gatekeeper をデプロイメントするのに適切な場所を見つけるには、ネットワーク間デバイスがクライアントネットワークからサーバーネットワークに転送できるメッセージの種類を判別してください。

次のケースは、隣接するクライアントとサーバーネットワークの Gatekeeper をデプロイメントできる場所をすべて示しています。

ケース 1：Gatekeeper が不要な場合

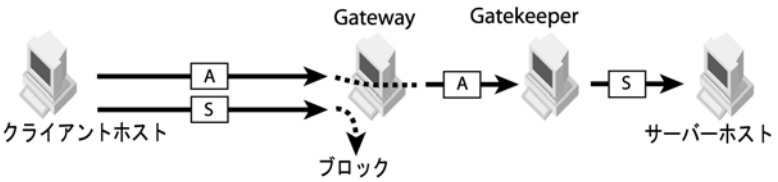
ゲートウェイが、すべてのクライアントメッセージをクライアントネットワークからサーバーネットワークに転送できる場合は、Gatekeeper は必要ありません。

次の図は、クライアントがタイプ A のメッセージをサーバーに送信し、サーバーはタイプ A のメッセージを監視します。ゲートウェイは、メッセージ（タイプ A）をサーバーネットワークに転送します。次に、サーバーがそのメッセージ（タイプ A）を受信します。タイプ A の一般的な例は、IIOP と IIOP ／ SSL です。



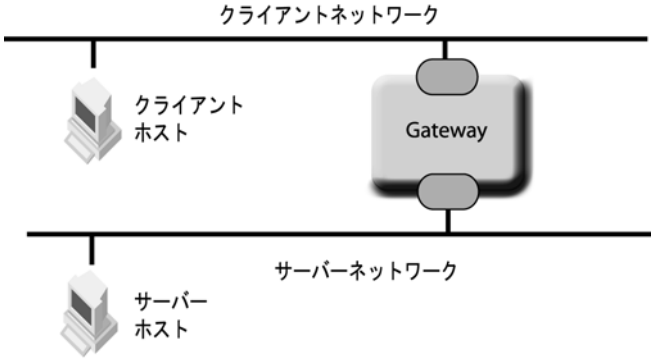
ケース 2：サーバーネットワークに GateKeeper がある場合

次の図では、サーバーがタイプ S のメッセージを監視します。ゲートウェイは、タイプ S のメッセージはブロックしますが、タイプ A のメッセージはクライアントネットワークからサーバーネットワークに転送できます。クライアントがタイプ S のメッセージをサーバーに送信しても、そのメッセージはゲートウェイによってブロックされてしまいます。ゲートウェイがメッセージをサーバーネットワークに転送できるように、クライアントはタイプ A のメッセージを送信する必要があります。サーバーネットワークで GateKeeper はプロキシとして必要です。クライアントがタイプ A のメッセージで Gatekeeper と通信するのに対して、Gatekeeper はタイプ S のメッセージでサーバーと通信します。HTTP と IIOP が、それぞれタイプ A とタイプ S の例です。この構成例は HTTP トンネリングモードで、ゲートウェイ／ファイアウォールは HTTP パケットは許可していますが、IIOP パケットは許可していません。



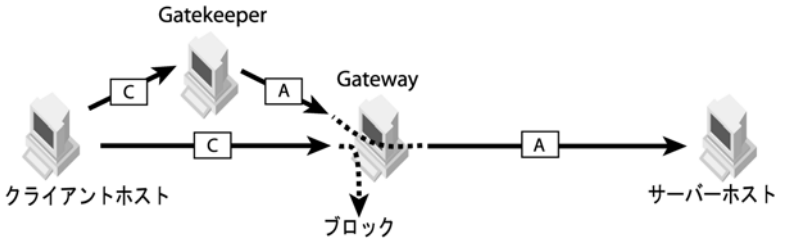
ケース 3：クライアントネットワークに GateKeeper がある場合

サーバーはタイプ A のメッセージを監視しますが、クライアントがサーバーとの通信に使用できるのはトランスポートタイプ C だけです。ゲートウェイは、タイプ C のメッセージはブロックしますが、タイプ A のゲートウェイは転送します。クライアントネットワークには Gatekeeper が必要です。クライアントはトランスポートタイプ C で Gatekeeper と通信し、Gatekeeper はタイプ A のメッセージでサーバーと通信します。ゲートウェイは、タイプ A のメッセージを Gatekeeper からサーバーネットワークに転送します。トランスポートタイプ C とトランスポートタイプ A の例は、それぞれ HTTP と IIOP です。



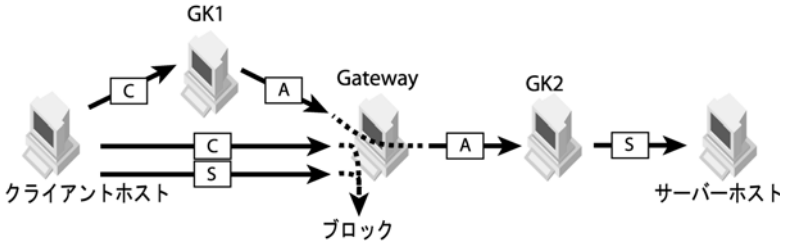
ケース 4：両方のネットワークに GateKeeper がある場合

ゲートウェイは、クライアントが送信したメッセージ（タイプ C）と、サーバーが監視できるメッセージ（タイプ S）の両方をブロックします。その他のタイプのメッセージ（タイプ A）は転送できます。したがって、クライアントネットワークとサーバーネットワークの両方に Gatekeeper が必要です。クライアントは、タイプ C のメッセージを使用して GK1 と通信します。GK1 は、タイプ A のメッセージを使用して GK2 と通信し、タイプ A のメッセージは、タイプ S のメッセージを使用してサーバーと通信する GK2 によって転送されます。タイプ C のメッセージの例は HTTP、タイプ A のメッセージの例は SSL、タイプ S のメッセージの例は IIOP です。



ケース 5：ネットワーク間デバイスに GateKeeper（デュアルホーム）がある場合

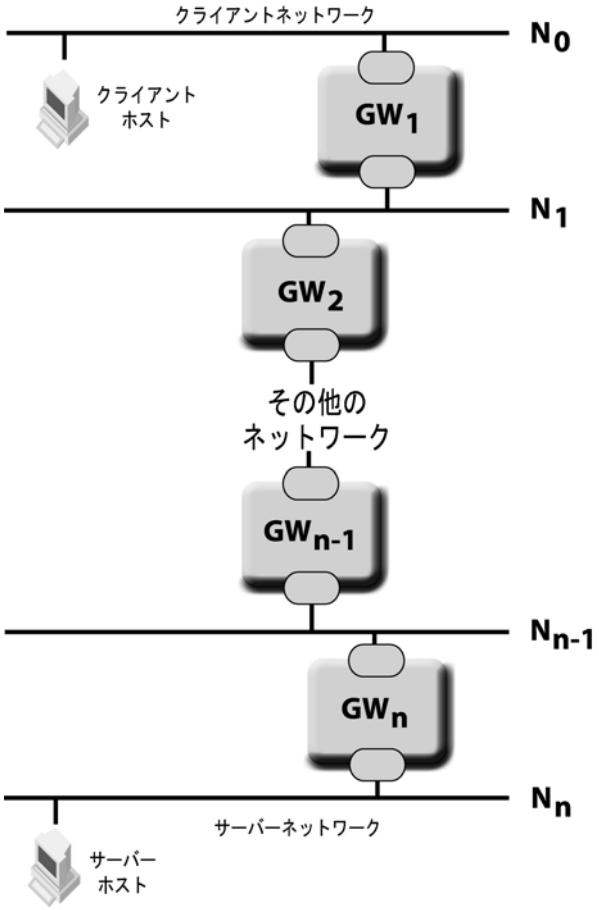
デュアルホームホストに Gatekeeper をインストールすると、サーバーネットワークに Gatekeeper をデプロイメントした場合（ケース 2）と同じような働きをします。違いは、Gatekeeper が、外部ネットワークにクライアントメッセージがないか常に監視する点です。内部ネットワークに存在するクライアントが同じ GateKeeper を使用してサーバーにバインドする必要がある場合、メッセージを GateKeeper リスナーに届けるには、まず外部ネットワークに転送する必要があります。HTTP と IIOP が、それぞれタイプ A およびタイプ S のメッセージの例です。



間に複数のネットワークが存在するクライアントとサーバー

複雑な環境になると、クライアントネットワークとサーバーネットワークの間に複数のネットワークが存在します。隣接する各ネットワークは、ネットワーク間デバイスを使用して接続されています。

図 2.2　間に複数のネットワークが存在するクライアントとサーバー



説明のため、クライアントネットワークに N_0 と番号を付けます。クライアントネットワークの隣のネットワークの番号は N_1、その次は N_2、以下サーバーネットワークまで同様にして番号を与えます。以下の説明で、サーバーネットワークの番号は N_n とします。ネットワークの設定に応じて、n を実際の数字に置き換えてください。また、ネットワーク N_{n-1} と N_n の間のネットワーク間デバイスの番号は GW_n とします。

クライアントは、複数のトランスポートタイプを使用してサーバーと通信できます。トランスポートタイプの例としては、IIOP、IIOP ／ SSL、HTTP、HTTPS などがあります。有効なトランスポートタイプごとに、クライアントメッセージが到達できる範囲で最も遠いネットワークを見つけます。ネットワーク N_0 にあるクライアントは、ネットワーク N_0 にメッセージを送信します。GW_1 は、ネットワーク N_1 にメッセージを送信する場合とそうでない場合があります。GW_1 が N_0 から N_1 にメッセージを送信できる場合、メッセージはネットワーク N_1 に届きます。続いて、GW_2 は、ネットワーク N_2 にメッセージを送信する場合とそうでない場合があります。クライアントネットワークからサーバーネットワークへと、ネットワークを通過していきます。メッセージが到達する最後のネットワークを N_c とします。つまり、GW_{c+1} ではネットワーク N_{c+1} にメッセージを送信できません。

サーバーには、1 つまたは複数のリスナーポートがあります。各ポートごとに、クライアントから送信される一種類のメッセージを監視します。たとえば、IIOP リスナーポートと

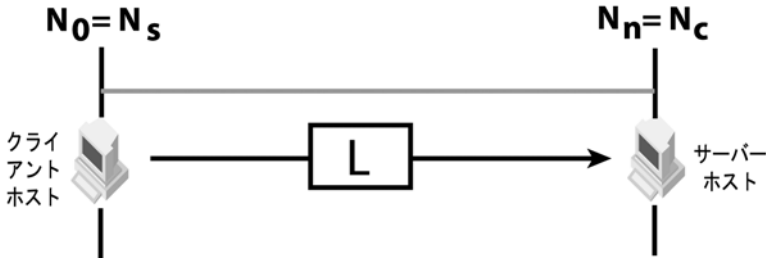
SSL リスナーポートがあるサーバーの場合、IIOP ポートを使用して IIOP メッセージを監視し、SSL ポートを使用して SSL 上の IIOP メッセージを監視します。各リスナーポートごとに、クライアントメッセージが到達できる範囲でサーバーから最も遠いネットワークを見つけてください。最遠のネットワークを N_s とします。つまり、ネットワーク N_s にあるクライアントはサーバーにメッセージを送信できます。

メモ VisiBroker 3.x スタイルのコールバックが必要な場合、N_c と N_s にはさらに条件が必要です。サーバー（ネットワーク N_n）からのコールバックメッセージは、ネットワーク N_s に届くものとします。GateKeeper を使用するとき、クライアント側でコールバック通信チャネルをネットワーク N_c に設定できるものとします。

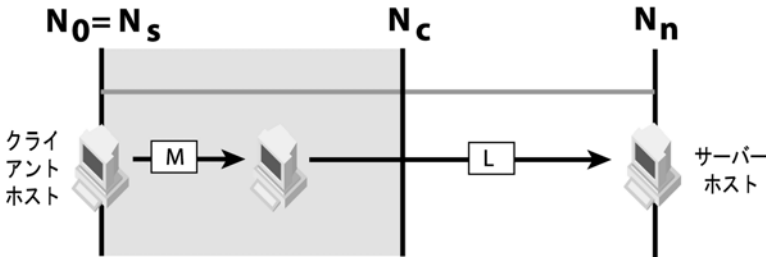
ケース 1：サーバーがクライアントネットワークからメッセージを受信できる場合（s=0）

サーバーがトランスポートタイプ L を監視すると仮定します。クライアントネットワークから送信されたトランスポートタイプ L のメッセージは、サーバーネットワークに届き、その後サーバーに届きます。

クライアントがトランスポートタイプ L を使用してメッセージを送信できる場合は、タイプ L のクライアントメッセージをサーバーネットワークに転送できるため、Gatekeeper は必要ありません。たとえば、サーバーは IIOP を監視しているため、クライアントは IIOP メッセージを送信できます。クライアントの IIOP メッセージは、ファイアウォール、ゲートウェイまたはルーターによってブロックされることなくサーバーに転送できます。

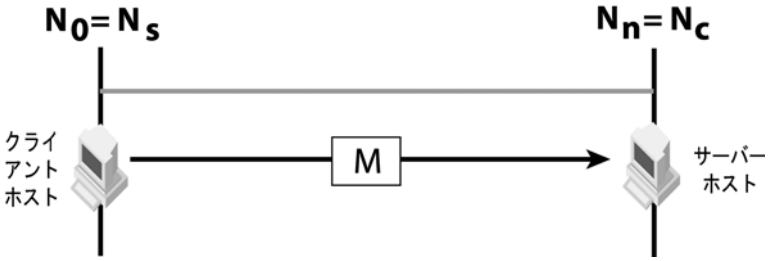


クライアントがトランスポートタイプ L でメッセージを送信できない場合、N_0 から N_c までのネットワークに GateKeeper をデプロイメントして、他のトランスポートタイプ (M) のクライアントメッセージをトランスポートタイプ L にプロキシ処理してください。たとえば、サーバーは IIOP を監視しているため、クライアントは IIOP over HTTP だけを使用して通信できます。

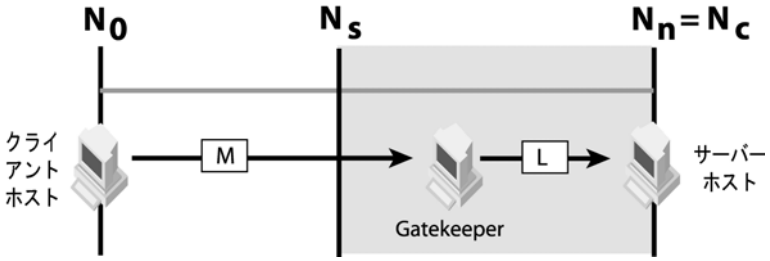


ケース 2：クライアントメッセージがサーバーネットワークに到達できる場合（c = n）

特定のトランスポートタイプ（M）のクライアントメッセージは、サーバーネットワークに到達できます。クライアントのトランスポートタイプがサーバーが監視するトランスポートタイプである場合は、GateKeeper は必要ありません。たとえば、クライアントが IIOP メッセージを送信し、サーバーも IIOP を監視する場合です。

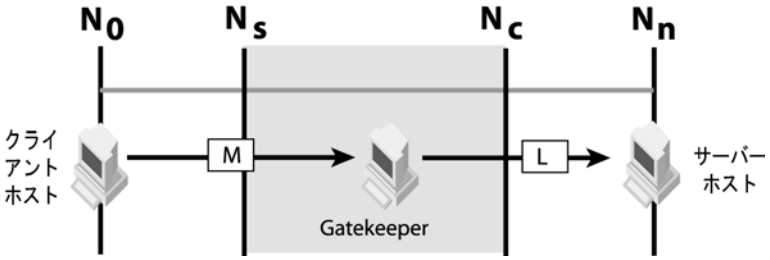


サーバーがクライアントのメッセージトランスポートタイプを監視しない場合、N_s から N_n のすべてのネットワークに GateKeeper が必要です。Gatekeeper はプロキシの役割を果たし、タイプ M のクライアントメッセージをサーバーリスナータイプ（L）のポートに中継します。たとえば、M（クライアントメッセージのトランスポートタイプ）は HTTP 上の IIOP、L（サーバーリスナータイプ）は IIOP です。



ケース 3：クライアントが到達可能なネットワークとサーバーまでネットワークが重複する場合（c >= s）

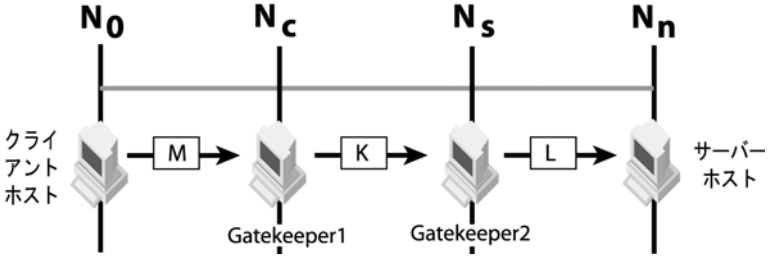
c >= s の場合、クライアントのトランスポートタイプ（M）とサーバーリスナータイプ（L）は異なるタイプを使用します。N_s と N_c 間のすべてのネットワークに GateKeeper をデプロイメントしてください。この場合、Gatekeeper はプロキシの役割を果たし、タイプ M のクライアントメッセージをタイプ（L）のサーバーリスナーポートに中継します。たとえば、HTTP メッセージ上のクライアントの IIOP は N_c までのネットワークに到達できます。N_s から N_n までの間のいずれかのネットワークから送信された IIOP メッセージはサーバーに到達します。N_s と N_c 間に GateKeeper をデプロイメントすると、HTTP メッセージ上のクライアントの IIOP をサーバーの IIOP リスナーポートにバインドできます。



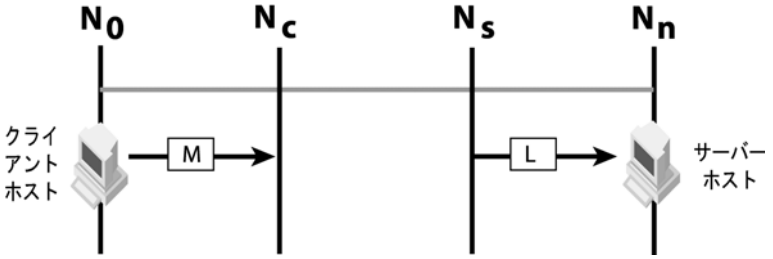
ケース 4：クライアントが到達可能なネットワークとサーバーまでネットワークが重複しない場合（c < s）

Gatekeeper チェインが可能かどうかを確認してください。Gatekeeper チェインの詳細は、29 ページの「GateKeeper のチェイン化」を参照してください。2 つの GateKeeper で別のトランスポートタイプ（K）を使用でき、それで N_c から N_s に通信できる場合にのみ GateKeeper チェインが可能です。ネットワーク（N_c）に GateKeeper を 1 つデプロイメントし、別のネットワーク N_s にさらに別の GateKeeper をデプロイメントします。各 Gatekeeper を連結します。たとえば、クライアントが IIOP over HTTP メッセージを送信し、サーバーが IIOP メッセージを監視する場合、2 つの Gatekeepers インスタンスは SSL を使用して相互に通信できます。クライアントは HTTP を使用して Gatekeeper 1

に接続し、Gatekeeper 1 は SSL を使用して Gatekeeper 2 と通信し、Gatekeeper 2 は IIOP を使用してサーバーと通信します。



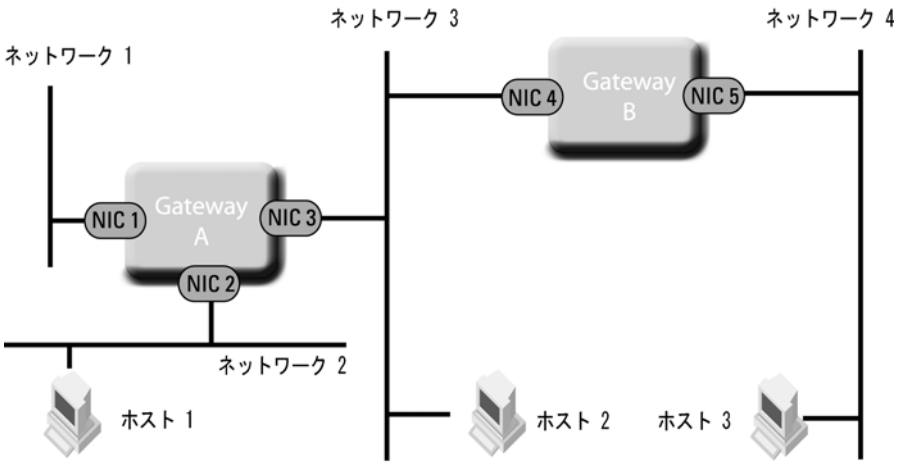
チェインが不可能な場合は、Gatekeeper のデプロイメントに適したネットワークが存在しないことになります。ネットワーク N_c とネットワーク N_s を接続するネットワーク間デバイスは、N_c から N_s に正しい種類のメッセージを送信できる設定とします。その後、新しい N_c と N_s を探して、先の事例を参照してください。



マルチホームホストの設定

マルチホームホストまたはルーターは、2 つ以上の物理ネットワークを接続します。これには複数のインターフェイスがあり、ネットワークインターフェイスカード（NIC）とも呼ばれます。各 NIC が 1 つのネットワークを接続します。マルチホームホストにより、接続されたネットワーク間の通信が可能になります。次の図は、2 つのマルチホームホスト（ゲートウェイ A とゲートウェイ B）を使用したネットワーク設定です。

図 2.3　マルチホームマシンネットワーク設定



マルチホームホストがデータパケットをあるネットワークから別のネットワークに正しくルーティングできるようにするには、IP 転送を有効にし、ルーティングテーブルを正しく設定する必要があります。同様に、ホストのルーティングテーブルも正しく設定する必要があります。

ホスト 1 にあるクライアントがホスト 3 にあるサーバーとの通信を試みる場合、ホスト 1 のクライアントは、ホスト 3 向けのメッセージをまずネットワーク 2 上で送信します。ゲートウェイ A は NIC 2 上でメッセージを受信し、そのメッセージを、NIC 3 を使用してネットワーク 3 にルーティングします。ゲートウェイ B は NIC 4 上でメッセージを受信し、そのメッセージを、NIC 5 を使用してネットワーク 4 にルーティングします。メッセージは、ホスト 3 のサーバーオブジェクトに届きます。この通信は、IP 転送が有効になっており、すべてのルーティングテーブルが正しく設定されている場合にのみ実行されます。

IP 転送の有効化

データパケットをあるネットワークから別のネットワークに転送するには、マルチホームホストが IP 転送が有効にする必要があります。IP 転送が無効になっている場合は、マルチホームホストがデータパケットを転送またはルーティングできません。

ルーティングテーブル

ルーティングテーブルのエントリの 1 つは、送信先ホストまたはネットワークに使用されます。各エントリに次の情報が含まれている必要があります。

- 送信先ホストまたはネットワーク
- 通信を行うゲートウェイ
- データパケットの送信に使用するインターフェイス

上の図のネットワーク設定例に対するルーティングテーブルの例を次の表に示します。

送信先	ゲートウェイ	インターフェイス
ネットワーク 1	ゲートウェイ A	NIC 1
ネットワーク 2	ゲートウェイ A	NIC 2
ネットワーク 3	ゲートウェイ A	NIC 3
ネットワーク 4	ゲートウェイ B	NIC 3

送信先	ゲートウェイ	インターフェイス
ネットワーク 1	ゲートウェイ A	ホスト 1
ネットワーク 2	ホスト 1	ホスト 1
ネットワーク 3	ゲートウェイ A	ホスト 1
ネットワーク 4	ゲートウェイ A	ホスト 1

送信先	ゲートウェイ	インターフェイス
ネットワーク 1	ゲートウェイ A	ホスト 2
ネットワーク 2	ゲートウェイ A	ホスト 2
ネットワーク 3	ホスト 2	ホスト 2
ネットワーク 4	ゲートウェイ B	ホスト 2

マルチホームホストのルーティングテーブルは、データパケットの転送先 NIC に関する情報を格納します。ゲートウェイ情報は、ルート内の次のゲートウェイとの通信に使用します。上の例のゲートウェイ A のルーティングテーブルを参照してください。NIC 3 を使用する場合は、ゲートウェイ A がゲートウェイ B と通信してネットワーク 4 にパケットをルーティングする必要があります。

ホストにも、固有のルーティングテーブルがあります。ホストが正しいゲートウェイと通信するには、ゲートウェイ情報が不可欠です。正しいゲートウェイと通信すれば、パケットは正しくルーティングされます（ホスト 2 のルーティングテーブルを参照）。ホスト 2 は、ネットワーク 1 とネットワーク 2 に到達するには、ゲートウェイ A と通信する必要があります。しかし、ネットワーク 2 がネットワーク 4 に到達するには、ゲートウェイ B と通信する必要があります。

次の方法を使用して、ルーティングテーブルが正しく設定されているかどうかを確認してください。

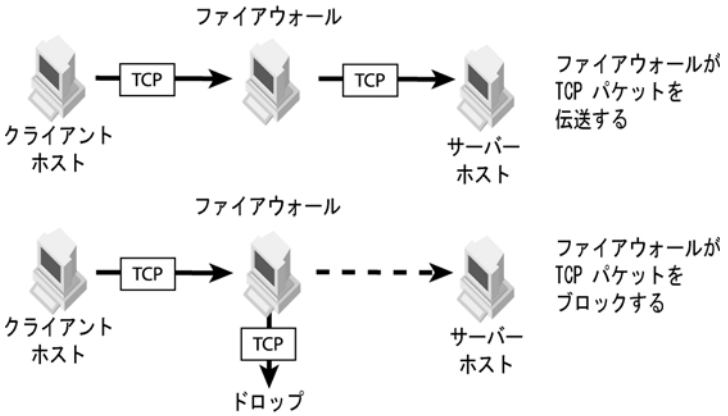
- ルーティングテーブルを印刷する
- 関連ホストを ping する
- 関連ホストまでのルートをトレースする

ファイアウォールの設定

ファイアウォールは、データパケットのフィルタリングを行うネットワークデバイスです。ファイアウォールは、受信したすべてのデータパケットを検査し、ファイアウォールのセキュリティポリシーに基づいて、パケットを転送またはドロップします。

ケース 1：クライアントのトランスポートの種類に制限がある場合

ファイアウォールのパケットフィルタリングの例を次の図に示します。



通常、ファイアウォールのセキュリティポリシーは、メッセージの種類、メッセージソースおよびメッセージの送信先を検査してフィルタリングを行います。ファイアウォールは、サービスのタイプ（ストリーム指向、データグラムパケットなど）や基底のプロトコルの種類（IP、ICMP、TCP、UDP など）に基づいてパケットフィルタ規則を適用できます。ファイアウォールで通信パスが TCP パケットストリームであると識別されると、そのファイアウォールは、セキュリティポリシーで定義されたパケットフィルタリング規則を適用して、そのパケットを許可するかドロップするかを判断します。TCP パケットストリームは、複数の種類のデータまたはペイロード（HTTP、IIOP、FTP、SSL など）を伝送できます。一般に、各ストリームには固有のポート番号が割り当てられ、1 クラスまたは 1 種類のメッセージのみ伝送できます。たとえば、IIOP メッセージは、TCP ポート 683 のパケットストリームで伝送できます。同様に、HTTP メッセージは、TCP ポート 80 で伝送できます。ファイアウォールは、パケットフィルタリング規則に基づいて、TCP ポート 80 は許可しますが、TCP ポート 683 は許可しない場合があります。特別な技術を使用すれば、TCP パケットストリームが複数の種類のメッセージを伝えることができます。Gatekeeper は、HTTP トンネリングと呼ばれる特別な技術を使用して IIOP メッセージを HTTP メッセージの中に埋め込み、TCP パケットストリームを通じて伝送します。

ファイアウォールがクライアントとサーバー間の通信パスに存在する場合は、そのファイアウォールが、クライアントからサーバーに送信されるデータパケットを転送またはドロップします。クライアントとサーバー間で正しく通信を行うには、ファイアウォールがクライアントのメッセージをサーバーに転送する必要があります。サーバーとなるのは、ユーザーアプリケーション、GateKeeper、またはスマートエージェントやネーミングサービスなど、他の VisiBroker サービスプロバイダです。サーバーのリスナーポートに送信されるクライアントのメッセージを転送するようにファイアウォールを設定してください。

ネットワークアドレス変換（NAT）の使い方

マルチホームホスト、ルーターおよびファイアウォールは、固有の機能に加えて、NAT も実行できます。NAT により、すべてのネットワークパケット内に存在するソースホストアドレス、ソースポート番号、送信先ホストアドレスおよび送信先ポート番号を変換できます。

クライアント側では、通常、ファイアウォールはソースホストアドレスを変換します。この手法は、限られた数のインターネット IP アドレスを共有する場合によく使用されます。

サーバー側では、ファイアウォールは、送信先ホストアドレスと送信先ポート番号の両方またはどちらかを変換します。これにより、実際の送信先ホストアドレスが、部外者からは見えなくなります。そのため、サーバーにアクセスする必要があるすべての部外者に通知せずに送信先ホストアドレスを変更できます。この柔軟性は、ポート番号にも適用されます。

Gatekeeper は、静的 NAT のみサポートしており、動的 NAT はサポートしていません。静的 NAT では、すべてのアドレスとポートが常に固定値に変換される事前定義のマッピングテーブルに基づいて変換が行われます。動的 NAT では、アドレスとポートを一定の範囲の値に変換するための規則を設定できます。この場合、ネットワークパケットの変換後のアドレスは、指定された範囲内の任意のアドレスでよいため、変換後のアドレスを事前に定義することはできません。

TCP ファイアウォールと NAT を使用するためのサーバーオブジェクトの設定方法については、第 3 章「ユーザープログラムの設定」を参照してください。クライアントとサーバーオブジェクト間の通信を正しく行うため、必ず NAT 変換マッピングを NAT デバイスに追加してください。

NAT を実行する場合は、関連するすべてのゲートウェイのルーティングテーブルが、使用するすべての偽ネットワークアドレスを処理するように設定する必要があります。この設定を怠ると、偽送信先アドレスを持つデータパケットを正しくルーティングできません。また、ファイアウォールが、NAT 実行時に使用するすべての偽送信先ホストアドレスおよび偽ポートにメッセージを転送するように設定しておくことも必要です。ファイアウォールが偽アドレスまたは偽ポートをブロックしてしまうと、そのパケットは、目的地に届きません。

GateKeeper の設定

ここでは、GateKeeper のポートとサービスの設定方法について説明します。リスナーポートは、設定の必要がある代表的なパラメータです。通常は、異なるファイアウォールが同じ範囲のポートを開いて通信を行うことはありません。Gatekeeper には多数のサービスがあり、事前に有効にしておかないと使用できないサービスもあります。

リスナーポート

次のプロパティは、GateKeeper の外部 IIOP および HTTP リスナーポートの番号を指定します。これらは、GateKeeper がクライアントの要求を監視するポートです。

```
vbroker.se.exterior.scm.ex-iiop.listener.port=683
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
```

GateKeeper がファイアウォールの背後にデプロイメントされている場合、外部クライアントは、ファイアウォールが IIOP または IIOP over HTTP メッセージの転送を、それぞれポート 683 と 8088 を介して実行することを許可している場合にのみ GateKeeper と通信できます。セキュリティの制約により、ファイアウォールが他のポート番号だけを許可する場合は、ファイアウォールで承認されたポートを使用するように Gatekeeper のリスナーポートを設定する必要があります。

管理サービス

GateKeeper の管理サービスを使用すると、VisiBroker Console を使用して GateKeeper を管理および設定できます。管理サービスでは、Gatekeeper がアクティブなときに Gatekeeper を動的に設定できます。次のプロパティは、管理サービスポートの番号を指定します。0 と 9091 が、それぞれ IIOP ポートと HTTP ポートのデフォルト値です。値 0 は、GateKeeper が起動時にポートをランダムに選択することを示します。

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=0
vbroker.se.iiop_tp.scm.hiop_ts.listener.port=9091
```

コールバックの有効化（VisiBroker 3.x スタイル）

コールバック機能（VisiBroker 3.x スタイル）は、VisiBroker 4.x 以降のバージョンでは、双方向通信に置き換えられました。GateKeeper が VisiBroker 3.x コールバックを使用するクライアントをサポートするには、次のプロパティ設定が必要です。

```
vbroker.gatekeeper.callbackEnabled=true
vbroker.gatekeeper.backcompat.callback=true
```

上のプロパティを設定すると、Gatekeeper はその内部サーバーエンジンを起動して、サーバーからのコールバックメッセージを受信します。リスナーは、in-iiop SCM と in-ssl SCM を使用して設定できます。また、コールバックリスナーが起動されて、クライアントがコールバックメッセージ用の追加通信チャネルを確立できるようになります。リスナーポートの指定方法と関連情報については、68 ページの「VisiBroker 3.x スタイルのコールバック」を参照してください。選択したポートがクライアントとサーバーから到達可能であること、つまりポートがどのファイアウォールにもブロックされていないことを確認してください。

パススルー接続の有効化

次のプロパティは、Gatekepper のパススルー接続を有効にします。

```
vbroker.gatekeeper.enablePassthru=true
```

クライアントがパススルー接続を要求すると、Gatekeeper は、サーバーとクライアントの間を通過するメッセージをまったく検査しません。上のプロパティを **false** に設定すると、Gatekeeper は、クライアントがパススルー接続を要求しても通常の（非パススルー）接続を使用して、クライアントをサーバーにバインドします。この場合は、Gatekeeper がルートおよびバインドするために交換メッセージを検査します。

Gatekeeper におけるパススルー接続の設定に利用できる次のようなプロパティが用意されています。

```
vbroker.gatekeeper.passthru.blockSize=16384
vbroker.gatekeeper.passthru.connectionTimeout=0
vbroker.gatekeeper.passthru.logLevel=0
vbroker.gatekeeper.passthru.streamTimeout=2000
vbroker.gatekeeper.passthru.inPortMin=1024
vbroker.gatekeeper.passthru.inPortMax=165535
vbroker.gatekeeper.passthru.outPortMin=0
vbroker.gatekeeper.passthru.outPortMax=65535
```

上のプロパティの詳細は、71 ページの「パススルー接続のサポート」を参照してください。

注意 パススルー機能は、Gatekeeper のリソースに大きな負担をかけます。この機能を使用する場合は、メモリーを十分に用意し、ソケットの数を増やしてください。

ロケーションサービスの有効化

Gatekeeper は、アプレットなど、Java サンドボックスセキュリティや既存のファイアウォールにより、スマートエージェント（osagent）と直接通信できないクライアントにロケーションサービスを提供します。ロケーションサービスを使用すると、クライアントは、GateKeeper を介してサーバーに「バインド」できます。

```
vbroker.gatekeeper.locationService=true
```

スマートエージェント（osagent）の指定

Gatekeeper は、スマートエージェントでサーバーオブジェクトを探します。同じネットワーク上にスマートエージェントが存在すれば、Gatekeeper は、それを自動的に検出します。Gatekeeper が稼動しているネットワーク上で稼動しているスマートエージェントがない場合は、スマートエージェントの場所を明示的に指定する必要があります。他のネットワークで稼動しているスマートエージェントを追加指定することもできます。

```
vbroker.agent.addr=< ホスト >
vbroker.agent.addrfile=< ファイル名 >
vbroker.agent.port=< ポート >
```

最初のプロパティはスマートエージェントのホスト IP アドレスを指定します。2 番めのプロパティは、スマートエージェントが実行されているホストのリストを定義するファイルを指定します。3 番めのプロパティは、OSAGENT_PORT を指定します。最初の 2 つのプロパティのデフォルト値は null で、この場合、Gatekeeper は同一ネットワークで稼動しているスマートエージェントと通信します。

スマートエージェントの設定とスマートエージェントパラメータの設定方法の詳細は、『*VisiBroker for C++ 開発者ガイド*』の「スマートエージェントの使い方」または『*VisiBroker for Java 開発者ガイド*』の「スマートエージェントの使い方」を参照してください。

オブジェクトアクティベーションデーモン（OAD）の指定

OAD サービスを使用して、GateKeeper は、バインドする必要があるサーバーを自動的に起動できます。この場合、サーバーは OAD サービスに登録されますが、アプレットがサーバーを呼び出した場合などは、GateKeeper を介してのみアクセスできます。OAD サービスを使用するために、GateKeeper は OAD IOR をロードする必要があります。次のプロパティは、GateKeeper に OAD IOR の場所を通知します。

```
vbroker.oad.iorFile=<OAD IOR>
```

OAD の詳細は、『*VisiBroker for C++ 開発者ガイド*』の「オブジェクトアクティベーションデーモンの使い方」または『*VisiBroker for Java 開発者ガイド*』の「オブジェクトアクティベーションデーモンの使い方」を参照してください。

GateKeeper サーバーエンジンの設定

Gatekeeper には、いくつかのデフォルトサーバーエンジンがあります。各サーバーエンジンには、少なくとも 1 つのサーバー接続マネージャ（SCM）があります。

- 外部サーバーエンジンは、GateKeeper　がクライアントオブジェクトをサーバーオブジェクトにバインドできるようにします。外部サーバーエンジンには、ex-hiop と ex-iiop の 2 つのデフォルト SCM があります。
- 内部サーバーエンジンはコールバックサービスを提供し、コールバックが有効になっている場合にのみ使用できます。外部サーバーエンジンには、in-iiop と in-ssl の 2 つのデフォルト SCM があります。
- iiop_tp サーバーエンジンは、管理サービスを提供します。外部サーバーエンジンには、hiop_ts と iiop_tp の 2 つのデフォルト SCM があります。

上の SCM のすべてのプロパティのリストについては、58 ページの「外部サーバー エンジン」、63 ページの「内部サーバー エンジン」、および 66 ページの「管理」を参照してください。

セキュリティサービス

次のプロパティで IIOP ／ SSL と HTTPS 上の IIOP を有効にして、GateKeeper を起動します。

```
vbroker.security.disable=false
vbroker.orb.dynamicLibs=com.borland.security.hiops.Init
vbroker.se.exterior.scms=ex-iiop,ex-hiop,ex-ssl,ex-hiops
```

- `vbroker.security.disable=false` プロパティは、必要なセキュリティパッケージを GateKeeper の VisiBroker ORB にロードします。
- `vbroker.orb.dynamicLibs=com.borland.security.hiops.Init` プロパティは、追加の HIOPS パッケージをロードします。これにより、HTTPS 経由の IIOP メッセージを使用できます。これは別にロードされます。
- `vbroker.se.exterior.scms=ex-iiop,ex-hiop,ex-ssl,ex-hiops` プロパティは、SCM の ex-ssl と ex-hiops を外部サーバーエンジンに追加します。

未使用の SCM を SCM リストから削除して、必要な SCM だけを起動できます。ただし、最初から指定されている scm ex-iiop と in-iiop は、リストから削除できません。

次のように、IIOP や HTTP などのセキュリティで保護されていないリスナーポートを無効にして、すべての通信を確実に暗号化することできます。

```
vbroker.se.exterior.scm.ex-iiop.listener.type=Disabled-IIOP
vbroker.se.exterior.scm.ex-hiop.listener.type=Disabled-IIOP
```

IIOP/SSL リスナーと HTTPS リスナーは、前に `vbroker.se.exterior.scm.ex-hiops` と `vbroker.se.exterior.scm.ex-ssl` を付けた SCM プロパティを使用して設定できます。この SCM プロパティの包括的なリストについては、付録 A「GateKeeper のプロパティ」を参照してください。

SSL トランスポート ID と信頼ポイント

SSL の場合は、SSL ネゴシエーションが公開キーなしで Diffie Hellman 鍵共有アルゴリズムを使用できるため、トランスポート ID はオプションです。

ただし、トランスポート ID なしでは、`peerAuthenticationMode` を `require` および `require_and_trust` として設定されたクライアントが接続されません。さらに、SSL サーバーとしての GateKeeper 自身がクライアントトランスポート ID を持たない場合は、クライアントトランスポート ID を必要としないこともあります。

ウォレットプロパティを使用した SSL ID のインストール

GateKeeper に証明書をインストールする最も簡単な方法は、次のウォレットプロパティを使用することです。

```
vbroker.security.wallet.type=Directory:<path_to_identities>
vbroker.security.wallet.identity=< ユーザー名 >
vbroker.security.wallet.password=< パスワード >
vbroker.security.trustpointsRepository=Directory:<path_to_trustpoints>
```

証明書ログインを使用して GateKeeper に SSL ID をインストール

ウォレットと信頼ポイントのプロパティを設定する簡単な方法のほかに、起動時に認証情報を取得（ログイン）することで、GateKeeper に SSL ID をインストールできます。認証情報の取得時に、ユーザーは、ファイルとディレクトリに関する質問、つまり証明書、秘密キー、および信頼されるルート証明書がどこに保存されているかに回答する必要があります。秘密キーの暗号を解除するためのパスワードは必ず質問されます。

ログイン時に質問されるファイルとディレクトリは、証明書ストレージのタイプによって異なります。デフォルトのストレージは、次のファイル内の JDK セキュリティ設定によって決定されます。

```
${JAVA_HOME}/jre/lib/security/java.security
```

JDK の標準では、Java キーストア（jks）として jks が設定されます。

```
#
# デフォルトのキーストアタイプ
#
keystore.type=jks
```

PKCS#12 ストレージの場合は、上の文字列を pkcs12 に変更できます。このストレージ形式は、証明書、信頼される証明書、および秘密キーが 1 つのファイルに格納される場合にのみ使用されます。JDK keytool のマニュアルを参照してください。

証明書ログインでは、次のプロパティを GateKeeper に明示的に設定する必要があります。

```
vbroker.security.login=true
vbroker.security.login.realms=< 領域リスト >
```

領域リストでは、特に Certificate#CLIENT、Certificate#SERVER、Certificate#ALL が必要です。

- Certificate#CLIENT は、発信 SSL 接続に使用される SSL ID です。
- Certificate#SERVER は、着信 SSL 接続に使用されます。
- Certificate#ALL は、両方に使用できます。

極端な例として、< 領域リスト > に 3 つの領域がすべてある場合を考えます。この場合は、GateKeeper の起動時に SSL ID の 3 つの異なる設定がユーザーから取得されます。

発信 SSL 接続を開く場合は、次のように動作します。

1. まず、Certificate#CLIENT を使用します。
2. Certificate#CLIENT に何も設定されていない場合は、Certificate#ALL を使用します。
3. Certificate#ALL にも何も設定されていない場合、発信 SSL 接続は ID を持たなくなります。

メモ　受信（サーバー）SSL 接続にも同様の優先順位が適用されます。

簡単なウォレットプロパティ設定を使用して設定される ID は、常に Certificate#ALL に格納されます。

peerAuthenticationMode の設定

peerAuthenticationMode ポリシーを通常どおりに使用します。次のようにプロパティを設定します。

```
vbroker.security.peerAuthenticationMode=none
```

アプレットと Java Webstart

Java プログラミング言語は、1 つの場所から即座にデプロイメントおよび実行されるプログラムを開発できる強力なツールです。この言語は、CORBA、特に VisiBroker for Java と組み合わせると、より機能が強化されます。

クライアントコードは、Java アプレットか、JNLP (Java Network Launching Protocol) を利用する Java Webstart アプリケーションのどちらかとして、Web サイトから即座にダウンロードしてインストールできます。

標準のアプレットクライアントに関する VisiBroker の設定

クライアントがアプレットの場合は、さらに次のプロパティ設定が必要です。

```
<applet archive=vbjorb.jar,vbsec.jar,lm.jar,sanct4.jar,
      sanctuary.jar,code="ClientApplet.class" width="200"
      height="80">
        <param name="vbroker.security.disabled" value="false">
        <param name="vbroker.orb.dynamicLibs"
            value="com.borland.security.hiops.Init">
...
</applet>
```

メモ

1. すべての VisiBroker jar が GateKeeper HTTP ルートディレクトリ（GateKeeper を起動する現在のディレクトリ）に格納されている必要はありません。
2. ライセンス jar lm.jar,sanct4.jar,sanctuary.jar は、アプレットが永続的 POA を作成する場合にのみ必要です。
3. VisiSecure 機能が呼び出される場合は、アプレットのアーカイブリストに vbsec.jar が必要です。また、それを有効にするアプレットパラメータも必要です。オプションで、HIOPS 機能が呼び出される場合は、上のように、dynamicLibs を使用して個別にロードする必要があります。

Java Webstart としてデプロイメントされる VisiBroker アプリケーション

Java Webstart アプリケーションは、独自の起動プログラムを持つため、Web ブラウザなしで実行できます。UNIX ではコマンドシェルから直接起動され、Windows ではダブルクリックして起動されます。この起動プログラムは、application/x-java-jnlp-file のデフォルトの MIME ハンドラです。これは、Windows では JDK/JRE のインストール時に、UNIX ではその他の方法で自動的に関連付けられます。したがって、Web ページでリンクをクリックすると、その MIME タイプの HTTP 応答が発生し、インストールされている Java Webstart 起動プログラムが起動されて、その応答の内容が処理されます。実際の内容は、必要な jar が配置されている場所に関する情報と、アプリケーションの実行に関連する他の情報を含む XML です。たとえば、必要な Java セキュリティ権限が含まれます。

Java Webstart としてデプロイメントされる標準の VisiBroker アプリケーションについては、gatekeeper bank_jws の例を参照してください。

第 3 章

ユーザープログラムの設定

この章では、ファイアウォールと Gatekeeper を使用するための、ユーザープログラム（クライアントとサーバー）を設定する方法について説明します。設定は、クライアントおよびサーバーのプロパティで行います。プロパティの設定方法については、付録 A「GateKeeper のプロパティ」を参照してください。

ファイアウォール背後のオブジェクトの使い方

オブジェクトがファイアウォールの背後で機能するには、通常、プログラミング環境と実行時環境の両方で設定を行う必要があります。特定のポータブルオブジェクトアダプタ（POA）のファイアウォールポリシーは、プログラムから設定する必要があります。ただし、すべての POA に対して同じファイアウォールポリシーをグローバルに設定する場合は、1 つのプロパティ設定を使用するだけで、ソースコードの変更は必要ありません。

単一 POA のプログラミング

サーバーがファイアウォールを通過できるように特定の POA のファイアウォールポリシーを設定するには、サーバーがアクティブ化される POA のファイアウォールポリシーを指定する必要があります。特に、次のコードをサーバーに追加する必要があります。次の例では、Bank サンプルをベースとして使用します。

単一の POA をプログラムで設定するには

1 ファイアウォールポリシーを作成します。

Java

```
org.omg.CORBA.Any fw_policy_value = orb.create_any();
com.inprise.vbroker.firewall.FirewallPolicyValueHelper.insert(
   fw_policy_value, com.inprise.vbroker.firewall.EXPORT.value);
org.omg.CORBA.Policy firewall_policy = orb.create_policy(
   com.inprise.vbroker.firewall.FIREWALL_POLICY_TYPE.value,
fw_policy_value);
org.omg.CORBA.Policy[] policies = {
   firewall_policy,
   rootPOA.create_lifespan_policy(LifespanPolicyValue.PERSISTENT)
};
```

C++

```
CORBA::PolicyList policies;
policies.length(2);
```

```
policies[(CORBA::ULong)0] = rootPOA->create_lifespan_policy
   (PortableServer::PERSISTENT);
CORBA::Any policy_value;
policy_value <<= Firewall::EXPORT;
CORBA::Policy_ptr fpolicy= orb->create_policy
   (Firewall::FIREWALL_POLICY_TYPE, policy_value);
policies[(CORBA::ULong)1] = fpolicy;
```

2 サーバーをアクティブ化する POA にこのポリシーを適用します。

Java
```
POA bankPOA = rootPOA.create_POA("bank_agent_poa", rootPOA.the_POAManager(),
   policies);
```

C++
```
PortableServer::POA_var bankPOA = rootPOA->create_POA("bank_agent_poa",
   poa_manager, policies);
```

ルート POA だけがデフォルトのポリシーを使用するため、ファイアウォールの後ろでアクセスする必要があるすべてのサーバーをアクティブ化するために、ルート POA を使用できます。また、Account サーバーをアクティブ化するために、別の POA を作成する必要があります。Account サーバーはクライアントから直接バインドしない方がよいため、POA を一時 POA として作成します。

Java
```
 policies = new org.omg.CORBA.Policy[] {
   firewall_policy,
   rootPOA.create_lifespan_policy(LifespanPolicyValue.TRANSIENT)
};

POA accountPOA = rootPOA.create_POA(
   "account_agent_poa", rootPOA.the_POAManager(), policies);
```

C++
```
policies.length(2);
policies[(CORBA::ULong)0] = rootPOA->create_lifespan_policy
   (PortableServer::TRANSIENT);
policies[(CORBA::ULong)1] = fpolicy;
PortableServer::POA_var accountPOA = rootPOA->create_POA("account_agent_poa",
   poa_manager, policies);
```

サーバーに関連付けられているすべての POA のファイアウォールポリシーを設定

次のプロパティを使用すると、サーバーに関連付けられているすべての POA のファイアウォールポリシーを設定できます。

```
-Dvbroker.orb.exportFirewallPath=true
```

exportFirewallPath プロパティを指定すると、POA の作成時にファイアウォールポリシーを追加する必要がないため、ソースコードに変更を加える必要がありません。

実行時のファイアウォールパッケージのロード

Gatekeeper を使用するクライアントとサーバーでは、ORB を最初に初期化するとき、つまり次のメソッドを呼び出すときに、ファイアウォールパッケージとそのプロパティをロードする必要があります。

Java
```
org.omg.CORBA.ORB.init(String[] args,java.util.Properties property);
```

C++
```
CORBA::ORB_ptr CORBA::ORB_init(int& argc, char *const *argv);
```

Java 次のプロパティを使用すると、ファイアウォールパッケージが VisiBroker for Java ORB にロードされます。firewall コンポーネントを定義しているので、Gatekeeper は、firewall.Init パッケージをロードする必要はありません。

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
```

C++ VisiBroker for C++ ORB には Gatekeeper とファイアウォールのサポート機能が組み込まれているので、新たなライブラリを追加しなくても Gatekeeper とファイアウォールを使用できます。

クライアント プロパティの設定

クライアントとサーバー間の通信方法は制限できます。特に、ポリシーとプロパティを使用することにより、次の指定を有効または無効にできます。

- クライアントが実サーバーのプロキシとして Gatekeeper を使用する。
- Java クライアントが HTTP トンネル チャネルでサーバーと通信する。
- クライアントが IIOP/SSL でサーバーと通信する。
- Java クライアントが HTTPS トンネル チャネルでサーバーと通信する。
- クライアントとサーバー間の接続で、Gatekeeper がメッセージを検査しないで転送する（パススルー モード）。
- クライアントが使用できる送信方法から適切な方法を選択する。

HTTP トンネル チャネルについて

HTTP プロトコルによりトンネルを通過する VisiBroker for Java の IIOP 用拡張機能は、HIOP と呼ばれます。このモードでは、ORB は同時に 1 つの要求のみトンネル処理できます。つまり、クライアントに複数のスレッドがあり、同時に要求を出す場合は、ORB は各要求を直列化します。したがって、サーバーに到着する要求も直列化します。クライアントが HTTP トンネルを使っているとき、サーバーに到着する同時要求の数は、クライアントの ORB の数と一致します。これは、クライアントで HTTP トンネルを使っている場合の制限です。

以下では、クライアントのさまざまなポリシーとそのプロパティ設定を示します。これらのポリシーの組み合わせを変えて、クライアントからサーバーまでの通信方法も決定できます。

クライアントに常にプロキシを指定

次のプロパティを設定すると、クライアントは、常に GateKeeper を使ってサーバーに要求を送信します。

クライアントのプロパティ：

```
vbroker.orb.alwaysProxy=true
```

このプロパティはオプションです。このプロパティを設定しなければ、オブジェクトの所在がサーバー側ファイアウォールの背後かどうかをクライアントはサーバーの IOR で判断し、それに基づいてファイアウォールを通過します。上記のプロパティを設定しない方がよい場合もあります。たとえば、信頼できるネットワーク内にあるローカルオブジェクトとファイアウォール背後のリモートオブジェクトの両方をクライアントで起動する場合がそうです。プロパティを設定しなければ、クライアントは、Gatekeeper を通過せずに直接ローカルオブジェクトを起動できるので効率的です。

クライアントのプロパティ：

```
vbroker.orb.gatekeeper.ior=<IOR>
```

クライアントは、上記のプロパティで GateKeeper IOR を指定することもできます。このメソッドは、クライアントがスマートエージェントで Gatekepper を検出できない場合に役立ちます。

クライアントでの HTTP トンネルの指定

次のプロパティを設定すると、クライアントは、HTTP トンネル チャネルでサーバーと通信します。

クライアントのプロパティ：

```
vbroker.orb.alwaysTunnel=true
```

トンネルの制限： HTTP プロトコルでは、要求は同じ接続で多重化されます。ORB は同時に 1 つの要求のみトンネル処理できます。つまり、クライアントに複数のスレッドがあり、同時に要求を出す場合は、ORB は各要求を直列化します。したがって、サーバーに到着する要求も直列化します。

クライアントが HTTP トンネルを使っているとき、サーバーに到着する同時要求の数は、クライアントの ORB の数と一致します。これは、クライアントで HTTP トンネルを使っている場合の制限です。

この設定により、クライアントのアプレットまたはアプリケーションは、IIOP over HTTP を介して Gatekepper と通信し、Gatekepper は IIOP で要求を実サーバー オブジェクトに中継します。サーバー オブジェクトから Gatekeeper への応答は、IIOP を介して送信されます。Gatekeeper は、IIOP over HTTP を介して応答をクライアントに転送します。

Java HTTP トンネリングをクライアントが実行するとき、アプレットでは `vbroker.orb.alwaysTunnel` を設定します。アプレット クライアントは、URL ネーミングまたは文字列化された IOR でプロパティ `vbroker.orb.gatekeeper.ior` を設定して Gatekeeper の IOR を取得する必要があります。また、アプレット クライアントでは、`vbroker.locator.ior` プロパティは設定できません。

メモ HTTP トンネリングでコールバックは使用できません。

注意 HTTP プロキシ サーバーにはさまざまな実装方法があるので、プロキシ サーバーの種類によっては、HTTP トンネリングが正常に機能しない場合があります。詳細については、Borland の Web サイトにある「VisiBroker Gatekeeper FAQ」を参照してください。

クライアントにセキュリティで保護された接続を指定

クライアントのプロパティ：

```
vbroker.orb.alwaysSecure=true
```

クライアントは、IIOP/SSL または HTTP 上の HTTPS を介してサーバーと通信します。

クライアントのプロパティ：

```
vbroker.orb.alwaysSecure=true
vbroker.orb.alwaysTunnel=true
```

クライアントは、HTTPS 上の IIOP を使用するサーバーだけと通信します。

クライアントにパススルー接続を指定

この種類の接続では、接続の終了やメッセージの解釈が Gatekeeper では行われません。この種類の接続は、Gatekeeper にクライアントとの信頼関係を確立するための SSL 機能や関連する証明書がない場合に役に立ちます。このような場合、クライアントとサーバーは、GateKeeper を介さずに SSL 接続をネゴシエートします。したがって、GateKeeper は、クライアントとサーバーの間で渡されるメッセージを解釈しません。

クライアントのプロパティ：

```
vbroker.orb.proxyPassthru=true
```

Gatekeeper のプロパティ：

```
vbroker.gatekeeper.enablePassthru=true
```

- vbroker.orb.proxyPassthru プロパティでは、ORB レベル PROXY_MODE_POLICY プロパティの値を設定します。true に設定すると、クライアント側でプロキシを使用するすべてのオブジェクトがパススルー接続を要求します。または、特定のオブジェクトだけにパススルー接続を要求させるには PROXY_MODE_POLICY を設定します。
- vbroker.gatekeeper.enablePassthru プロパティを指定すると、GateKeeper はパススルー接続を受け入れます。このプロパティは Gatekeeper 全体に対して有効で、Gatekeeper の動作だけに影響します。

vbroker.orb.proxyPassthru プロパティを設定すると、クライアントは Gatekeeper からパススルー接続の取得を試みます。ただし、Gatekeeper がパススルー接続を許可するのは、vbroker.gatekeeper.enablePassthru プロパティが **true** に設定されている場合だけです。ほかの GateKeeper のパススループロパティについては、16 ページの「パススルー接続の有効化」を参照してください。

パススルー接続の有効化

vbroker.gatekeeper.enablePassthru を **false** に設定すると、GateKeeper はパススルー接続の確立を許可せず、クライアントはサーバーとの通常の（非パススルー）接続だけを取得できます。Gatekeeper は、この場合、クライアントとサーバーの間で交換されるメッセージをルーティングおよびバインディングの目的で検査します。Gatekeeper が SSL メッセージに SSL 認証を与えられない場合、接続は失敗します。

クライアントのビッド順の指定

クライアントのプロパティ：

クライアントのビッド順は、サーバーとの接続に使用するさまざまなトランスポートの相対的な重要度を指定します。上位のトランスポートほど優先順位が高くなります。次のプロパティを設定すると、クライアントは、サーバーの IOR で利用できるトランスポートのうち優先順位が高いものから先に試します。1 つのトランスポートが失敗すると、クライアントは、次に利用できるトランスポートを試します。

```
vbroker.orb.bidOrder=inprocess:liop:ssl:iiop:proxy:hiop:locator
```

上記の例で、IOR に LIOP と IIOP の両方のプロファイルがある場合、クライアントはまず LIOP を試します。LIOP が失敗した場合にだけ、IIOP を試します。

クライアントのプロパティ：

```
vbroker.orb.bidCritical=inprocess
```

クリティカルビッドには、ビッド順における指定位置に関係なく、最高の優先順位が適用されます。複数のクリティカルビッドがある場合、ビッド間の相対的な重要度はビッド順で決まります。

クライアントのコールバックリスナーポートの指定（VisiBroker 3.x スタイル用）

次のプロパティは、サーバーが VisiBroker 3.x スタイルのコールバック接続を確立するときのクライアントのリスナーポートを指定します。リスナーの種類は、Callback-IIOP に設定して、標準 IIOP リスナーと区別します。

クライアントのプロパティ：

```
vbroker.se.iiop_ts.scm.iiop_tp.listener.port=<port>
vbroker.se.iiop_ts.scm.iiop_tp.listener.type=Callback-IIOP
```

サーバープロパティの設定

サーバープロパティを使用して、サーバーの IOR を構築します。クライアントは、この IOR を使ってサーバーまでの通信パスを確立します。

サーバーのリスナーポートの指定

以下では、サーバーのリスナーポートを指定するために使用されるプロパティ設定について説明します。

ランダムリスナーポート

次のプロパティのデフォルト値は 0 で、その場合、システムはサーバーの起動時にランダムにポート番号を選択します。

サーバーのプロパティ：

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=0
```

特定リスナーポート

次のサーバープロパティは、サーバーがクライアントからの IIOP メッセージを監視するポートを割り当てます。

サーバーのプロパティ：

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<port number>
```

メモ 上の例のように指定すると、同じネットワーク上にあるすべてのクライアントが、指定したポートを使ってサーバーとの通信を直接確立できます。別のネットワーク上にあるクライアントから送信されたメッセージは、ゲートウェイまたはルーターから転送する必要があります。サーバーがサブネットの外にあるクライアントからの接続を許可する場合は、ルーターまたはファイアウォールも、指定されたポートのメッセージを許可するように設定されている必要があります。逆に、サーバーが同じサブネット上にあるクライアントからの接続だけを許可する場合、ルーターまたはファイアウォールは、指定したポートのメッセージをブロックして、外部クライアントオブジェクトによる無許可のアクセスを防ぐように設定する必要があります。

ポート変換（NAT）

ネットワークアドレス変換（NAT）を使用して、偽ポート（プロキシポート）からサーバーの実際の IIOP リスナーポートにポート変換する場合、次のプロパティ設定を使ってサーバーの IOR に偽ポートを公開します。

サーバーのプロパティ：

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<real_port>
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=<fake_port>
```

上の設定では、サーバーが実ポートを監視し、クライアントが偽ポートにメッセージを送信します。proxyPort プロパティのデフォルト値は 0 で、その場合、プロキシポートは使用しません。

メモ NAT を指定する方法としては、TCP ファイアウォールのプロパティを使用する方が適切です。これについては、後で説明します。

IIOP ポートの無効化

次のプロパティを設定すると、サーバーの IIOP リスナーポートが無効になり、サーバーは、IIOP/SSL などのセキュリティで保護された特定のポートでのクライアントの要求を許可します。サーバーは、公開された IIOP ポートでの IIOP メッセージを許可しません。

サーバーのプロパティ：

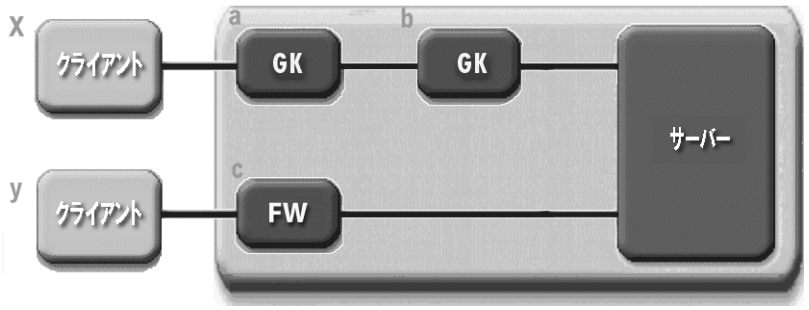
```
vbroker.se.iiop_tp.scm.iiop_tp.listener.type=Disabled-IIOP
```

サーバーまでの通信パスの指定

パスのパターンとしては,1 本のクライアントメッセージに対してサーバーまでのパスが複数ある場合と、さまざまなクライアントからのメッセージがさまざまなパスを通じて同じサーバーに到達する場合があります。これらのパスはすべてサーバーのプロパティで設定します。これにより、クライアントがサーバーにメッセージを送信するために必要な情報が、生成された IOR に追加されます。

次の図は、パスが 2 つあるファイアウォール構成の例で、サーバーまでの通信パスを示します。構成 X には、2 つの Gatekeepers のチェインがあります。構成 Y には、1 つの TCP ファイアウォールがあります。

図 3.1　サーバーまでの通信パス



上の図のような構成でサーバーを設定するには、次の情報をサーバーのプロパティファイルに入力します。

1 すべてのファイアウォールパスを宣言します。

```
vbroker.se.iiop_tp.firewallPaths=x,y
```

2 各パスのコンポーネントを指定します。

```
vbroker.firewall-path.x=a,b
vbroker.firewall-path.y=c
```

以下では、ファイアウォールのコンポーネントを指定する方法について説明します。

プロキシサーバーのコンポーネントの指定

ファイアウォールを通過する IIOP プロキシの例を次に示します。

サーバーのプロパティ：

```
vbroker.firewall-path.x=a,b
vbroker.firewall.a.type=PROXY
vbroker.firewall.a.ior=http://www.inprise.com/GK/GateKeeper.ior
vbroker.firewall.b.type=PROXY
vbroker.firewall.b.ior=IOR:<GateKeeper's stringified ior>
```

最初のプロパティは、パス x にあるファイアウォールコンポーネントを定義します。2 番めと 4 番めのプロパティは、それぞれコンポーネント a と b の種類を指定します。どちらの種類のコンポーネントも、PROXY として定義します。これは、GateKeeper がすべての IIOP 要求を送信する IIOP プロキシサーバーであることを表します。3 番めのプロパティは、URL ネーミングを使って *Gatekeeper a* の IOR を定義します。5 番めのプロパティは、文字列化された IOR を使って *Gatekepper b* の IOR を定義します。

NAT による TCP ファイアウォールコンポーネントの指定

クライアントのメッセージがサーバーに到達するまでに、1 つまたは複数の TCP ファイアウォールを通過します。TCP ファイアウォールで NAT を実行する場合、TCP ファイアウォールのコンポーネントを定義します。TCP ファイアウォールのコンポーネントが NAT を実行しない場合は、コンポーネントを無視できます。

ルーターまたはファイアウォールを使って TCP レベルの IIOP メッセージを転送する方法を次に示します。

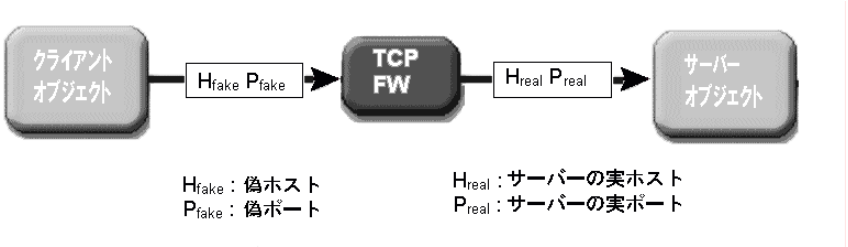
サーバーのプロパティ：

```
vbroker.firewall-path.y=c
vbroker.firewall.c.type=TCP
vbroker.firewall.c.host=<fake_host>
vbroker.firewall.c.iiop_port=<IIOP fake port>
vbroker.firewall.c.ssl_port=<SSL fake port>
vbroker.firewall.c.hiop_port=<HTTP fake port>
```

最初のプロパティは、パス y にあるファイアウォールコンポーネントを定義します。2 番めのプロパティは、コンポーネント c の種類が TCP であることを定義します。これは、ルーターやほかのネットワークデバイスで送信する HTTP 上のすべての IIOP、SSL、IIOP 用に定義済みのポートを提供します。3 番めのプロパティは、サーバーの偽ホストを定義します。最後の残り 3 つのプロパティは、IIOP、SSL、および HTTP メッセージタイプ用の偽ポートを定義します。

上の例では、コンポーネント「c」に指定した TCP ファイアウォールがホストになりポート変換（NAT）を行います。TCP ファイアウォールは、偽ホストをサーバーの実ホストに変換し、すべての偽ポートをサーバーの実リスナーポートに変換する設定にします。

図 3.2　NAT 使用の TCP ファイアウォール



第 4 章

高度な機能

ここでは、Gatekeeper のチェイン化、コールバック、アクセスコントロール、負荷分散、フォールトトレランス、SSL などの高度な機能について説明します。また、Gatekeeper のパフォーマンスを向上させるポイントについても説明します。

GateKeeper のチェイン化

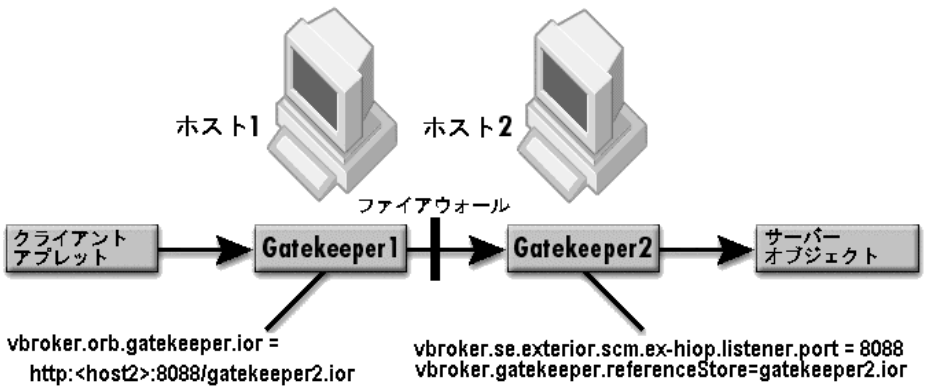
ファイアウォールを介してパスを提供するために、Gatekeeper をチェイン化できます。チェイン化には、次の 2 種類があります。

- 静的チェイン化
- 動的チェイン化

GateKeeper の静的チェイン化

静的なチェイン化では、1 つ前の Gatekeeper が次の Gatekeeper にメッセージを転送するように設定します。通信パスは固定されているため、静的なチェインと呼ばれます。

図 4.1 GateKeeper の静的チェイン化



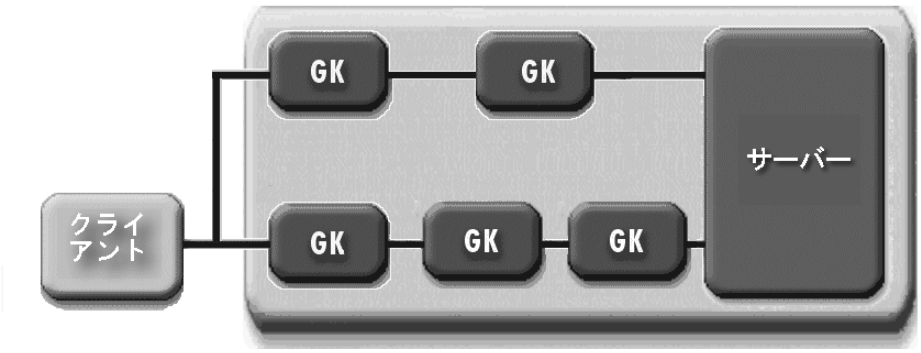
上の図では、ファイアウォール越しに通信するために 2 つのチェイン化された Gatekeeper が必要です。クライアントアプレットは Gatekeeper1 にメッセージを送信し、Gatekeeper1 はファイアウォール越しに Gatekeeper2 へメッセージを転送します。Gatekeeper1 は

Gatekeeper2 にメッセージを転送できますが、これは Gatekeeper1 に Gatekeeper2 の IOR があるからです。Gatekeeper2 の IOR は、Gatekeeper1 から Gatekeeper2 へメッセージを送信する方法、つまりファイアウォールを越える方法を指示します。

GateKeeper の動的チェイン化

静的チェイン化では通信パスを Gatekeeper の IOR に指定しますが、動的チェイン化の場合は通信パスをサーバーの IOR ファイルに指定します。サーバーの IOR ファイルがある場合、クライアントはサーバーの IOR 情報を使ってパスを選択します。クライアントは、最初のパスを使って接続に失敗すると、次のパスを試し、以下同様に続きます。

図 4.2 GateKeeper の動的チェイン化



上の図では、クライアントからサーバーへのパスが 2 つあります。どちらのパスでも、GateKeeper のチェイン化が必要です。サーバーの IOR にはこの 2 つのパスが指定されており、クライアントはこれを読み取って最初のパスを試し、接続に失敗すると第 2 のパスを試します。パスは実行時に動的に選択されます。

サーバーまでのパスを指定する方法については、27 ページの「サーバーまでの通信パスの指定」を参照してください。

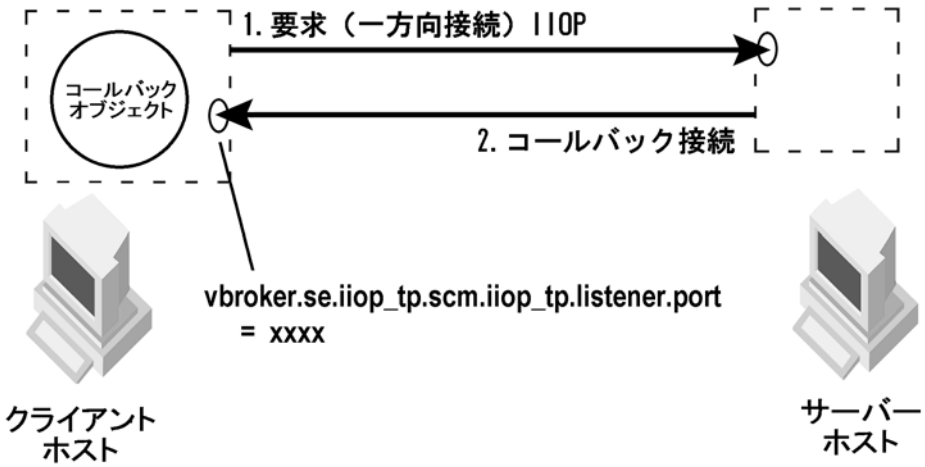
コールバック

ほとんどのインプリメンテーションでは、クライアントによって要求が開始され、それにサーバーが応答を戻します。また、クライアントが要求していない情報がクライアントに送られるインプリメンテーションもあります。これは、コールバックオブジェクトを作成して実装できます。コールバックオブジェクトのインプリメンテーションには、次の 3 つの方法があります。

GateKeeper なしのコールバック

次の図に示したインプリメンテーションは、クライアントとサーバーが双方向通信できる場合に適用されます。この場合は、間に入るファイアウォールが存在しないか、ファイアウォールが存在してもクライアントとサーバーの通信を妨げることはありません。

図 4.3 GateKeeper なしのコールバックの使用

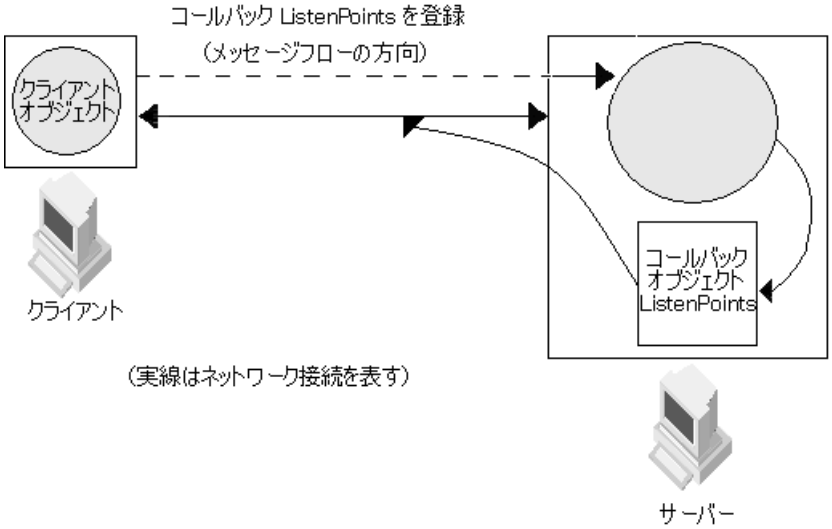


上の例で、クライアントはオブジェクトの作成、リスナーの起動、IOR の生成、サーバーへの要求と IOR の送信を行います。サーバーはクライアントのリスナーを呼び出し、コールバック接続を確立します。したがって、コールバックオブジェクトに送信されるメッセージはすべて、コールバック接続を介して配信されます。

双方向 GIOP を使用する GateKeeper なしの接続

双方向 IIOP がある場合、サーバーはクライアントが開始した接続で、クライアントに非同期情報を送信します。サーバーはクライアントとの接続を開始する必要がありません。

図 4.4 双方向 GIOP をサポートする GateKeeper なしのコールバックの使用



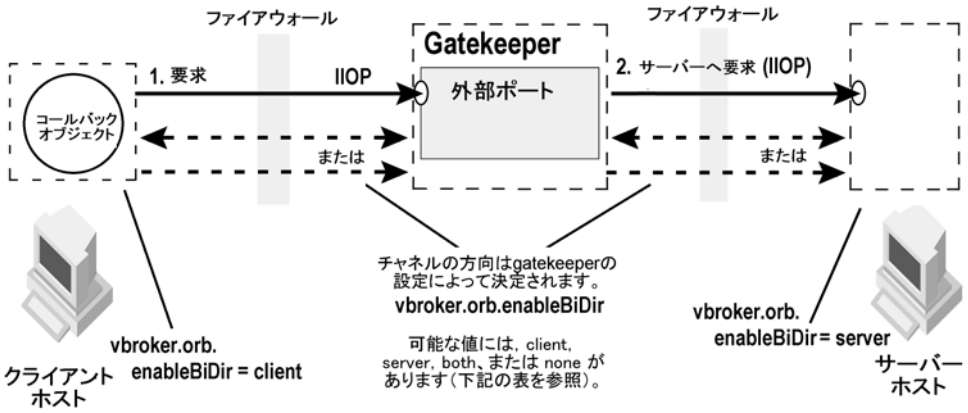
上の図で、クライアントは、サーバーとの接続を直接確立できますが、サーバーは、間にファイアウォールがあるため別個のコールバック接続を確立できません。そのため、クライアントとサーバーは双方向 GIOP 接続をネゴシエートし、クライアントが IIOP トラフィック用に最初に確立した接続を双方向で共有します。

CORBA 仕様でも、この機能を簡単に制御できる新しいポリシーを追加しています。GateKeeper 以外の双方向通信については、Borland Enterprise Server の『開発者ガイド』を参照してください。

Gatekeeper の双方向サポートによるコールバック

双方向 IIOP がある場合、サーバーはクライアントが開始した接続で、クライアントに非同期情報を送信します。サーバーはクライアントとの接続を開始する必要がありません。CORBA 仕様でも、この機能を簡単に制御できる新しいポリシーを追加しています。GateKeeper 以外の双方向通信については、『**開発者ガイド**』を参照してください。

図 4.5 Gatekeeper の双方向サポートによるコールバック



上の図で、Gatekeeper はクライアントとサーバーの間に置かれ、クライアントに対してはサーバーとして、サーバーに対してはクライアントとして機能します。クライアントと Gatekeeper の間、および Gatekeeper とサーバーの間の通信チャネルは、一方向接続と双方向接続のどちらにも設定できます。

チャネル設定は、一方向と双方向から選択できます。次の表は、クライアントで `vbroker.orb.enableBiDir=client` と定義し、サーバーで `vbroker.orb.enableBiDir=server` と定義している場合に、GateKeeper の `vbroker.orb.enableBiDir` の値を変えたときのチャネルの種類をまとめたものです。

vbroker.orb.enableBiDir=	クライアントと GateKeeper の間	Gatekeeper と サーバーの間
client	一方向	双方向
server	双方向	一方向
both	双方向	双方向
none	一方向	一方向

双方向接続サンプル

GateKeeper の双方向接続のサポートのサンプルは、GateKeeper for Java インストールの `examples/vbroker/bank_bidir` サブディレクトリにあります。

Bank BiDir サンプルは、Bank Callback サンプルに似ていますが、BiDir サンプルでは、クライアント、Gatekeeper、およびサーバーの間で双方向接続が確立される点が異なります。つまり、双方向インプリメンテーションでは送信呼び出しとコールバックの両方に同じ接続を使用します。

このサンプルでは、次の作業を行います。

- プロパティファイルで、クライアントにおける双方向接続を有効に設定します。
- サーバーオブジェクトに対する呼び出しに引数として渡すことができるコールバックオブジェクトを作成するように、クライアントをプログラミングします。
- プロパティファイルを使用して、サーバーの着信側ファイアウォールを含むファイアウォールパスを設定します。また、双方向接続を受け付けることができるようにサーバーを設定します。

- サーバーオブジェクト IOR 内でファイアウォールパスをエキスポートするように、サーバーをプログラミングします。
- 双方向接続をサポートするように、Gatekeeper を設定します。

クライアント

このサンプルでは、クライアント Client.java は次の処理を行います。

1. callback_poa という名前の POA にコールバックオブジェクトを作成します。このコールバックオブジェクトは、Gatekeeper を介してサーバーから呼び出されます。
2. **AccountManager** オブジェクトへのバインド
3. open() を呼び出し、コールバックオブジェクトを引数として渡して銀行口座を開くことで、このコールバックオブジェクトのオブジェクトリファレンスをサーバーに送信します。
4. 取得した **Account** オブジェクトリファレンスに口座残高を照会します。このとき、再びコールバックオブジェクトを（今度は balance メソッドに）渡します。

サーバー

このサンプルで、サーバー Server.java は次の処理を行います。

1. ファイアウォールポリシー値 EXPORT で、bank_poa という名前の永続的 POA と account_poa という名前の一時的 POA を作成します。
2. **AccountManager** サーバントのインスタンスを作成します。
3. bank_poa でサーバントを起動します。
4. クライアント要求の待機を開始します。
5. クライアントが Gatekeeper を介して開始したコールバックオブジェクトのメソッドを呼び出して要求に応答します。

ファイル名	説明
server.properties	bank サーバーを設定するためのプロパティファイル。このサンプルでは、Gatekeeper とサーバー間に双方向接続を確立するために、監視ポイントを受け付けるようにサーバーを設定します。接続を一方向にするには、vbroker.orb.enableBiDir プロパティを削除するか、値を none に設定します。このファイル内のほかのプロパティは、ファイアウォールパッケージをロードし、ファイアウォールパスを設定して、クライアントがサーバー側のオブジェクトにバインドして呼び出すことができるようにします。
client.properties	bank クライアントを設定するためのプロパティファイル。このサンプルでは、クライアントと Gatekeeper の間に双方向の接続を確立するために、クライアントが監視ポイントを公開するように設定します。接続を一方向にするには、vbroker.orb.enableBiDir プロパティを削除するか、値を none に設定します。サーバーの場合と同様に、クライアントが Gatekeeper を通過するために必要なファイアウォールライブラリをロードする vbroker.orb.dynamicLibs プロパティを設定します。
GateKeeper.properties	Gatekeeper を設定するためのプロパティファイル。このサンプルでは、Gatekeeper は、監視ポイントの公開と受け付けの両方を行うように設定します。したがって、クライアントと Gatekeeper 間の接続と Gatekeeper とサーバー間の接続はいずれも双方向になります。これらの接続を一方向に変更するには、vbroker.orb.enableBiDir プロパティを削除するか、値を none に設定します。

セキュリティに関する注意

注意 双方向 IIOP を使用すると、セキュリティに関する重大な問題が発生する可能性があります。特にセキュリティメカニズムが設定されていない場合は、悪意のあるクライアントがホストとポートを任意に選択して、双方向接続を要求する可能性があります。また、自分のホストにはない、セキュリティ上重要なオブジェクトのホストとポートをクライアントが指定する場合もあります。さらにセキュリティメカニズムが設定されていないと、着信接続を受け付けたサーバーは、接続要求元のクライアントの ID を識別したり、クライアントの完全性を検査できません。また、サーバーが双方向接続を介してほかのオブジェクト

にアクセスできる可能性があります。以上のことから、コールバックオブジェクトごとに独立した双方向 SCM を使用してください。クライアントの完全性に疑問がある場合は、双方向 IIOP を使用しないでください。

Java セキュリティ上の理由から、VisiBroker for Java を実行するサーバーは、双方向 IIOP を使用するように明示的に設定されていない限り、双方向 IIOP を使用しません。プロパティ vbroker.se.<sename>.scm.<scmname>.manager.importBiDir を使用すると、SCM 単位で双方向性を制御できます。たとえば、SSL を使ってクライアントを認証するサーバーエンジンだけで双方向 IIOP を有効にし、その他の通常の IIOP 接続は双方向で使用できないように選択することもできます。詳細については、「Gatekeeper のプロパティ」を参照してください。さらに、クライアントファイアウォール外でコールバックを行うサーバーとの双方向接続だけをクライアント側で有効にすることもできます。クライアントとサーバー間に高度なセキュリティを確立するには、相互認証（クライアントとサーバーの両方で vbroker.security.peerAuthenticationMode を REQUIRE_AND_TRUST に設定）の SSL を使用します。

アクセスコントロール

GateKeeper には、ルールベースのアクセスコントローラが組み込まれています。このコントローラは、次の条件に基づいてアクセスを許可または拒否します。

- 操作
- 署名
- サーバーのホスト／ポート
- サーバーのサブネット
- クライアントのホスト／ポート
- クライアントのサブネット

すべてのルールは指定した順序で評価され、最初に一致したルールで処理が実行されます。一致する規則がない場合はユーザーが指定したデフォルトの動作が実行されます。規則の構文については、付録 A「GateKeeper のプロパティ」を参照してください。

GateKeeper におけるカスタムアクセスコントロール

GateKeeper を使用すると、カスタマイズしたアクセスコントロールメカニズムを組み込むことができます。Access Control Manager は、GateKeeper プロパティを使って指定されているすべてのアクセスコントローラを呼び出します。Access Control Manager は、次のインターフェイスを使ってアクセスコントローラを実装します。

```
package com.inprise.vbroker.gatekeeper.security;
public interface AccessController {
  public void init(org.omg.CORBA.ORB orb, String prefix);
}
```

アクセスコントローラは、TcpConnectionInfo インターフェイスを使用して、クライアントに関する詳細な情報を取得します。

```
package com.inprise.vbroker.orb;
public interface TcpConnectionInfo {
  public String getLocalHostName();
  public int    getLocalPortNumber();
  public String getHostName();
  public int    getPortNumber();
  public long   getTotalBytesRead();
  public long   getTotalBytesWrote();
  public String name();
  public java.io.InputStream getInputStream();
  public java.io.OutputStream getOutputStream();
}
```

Access Control Manager は、init メソッドを呼び出してアクセスコントローラを初期化します。GateKeeper は、次の種類のアクセスコントローラインターフェイスをサポートします。

- **ObjectAccessController:** クライアントが GateKeeper にサーバーオブジェクトへのプロキシチャネル（通信パス）を設定するように要求すると、isObjectAccessible() メソッドが呼び出されます。このメソッドは、オブジェクトにアクセスできる場合に true を戻します。

```
package com.inprise.vbroker.gatekeeper.security;
import  com.inprise.vbroker.orb.TcpConnectionInfo;
import  com.inprise.vbroker.IOP.ServiceContext;
public interface ObjectAccessController extends AccessController {
  public boolean  isObjectAccessible(
    TcpConnectionInfo     clientInfo,
    org.omg.CORBA.Object  server,
    ServiceContext[]      contexts,
    byte[]                principal);
}
```

- **OperationAccessController:** クライアントが GateKeeper を介して要求を送信すると、isOperationAccessible() メソッドが呼び出されます。このメソッドは、指定されたオペレーションにアクセスできる場合に true を戻します。

```
package com.inprise.vbroker.gatekeeper.security;
import  com.inprise.vbroker.orb.TcpConnectionInfo;
import  com.inprise.vbroker.IOP.ServiceContext;
public interface OperationAccessController extends AccessController{
  public boolean isOperationAccessible(
    TcpConnectionInfo         clientInfo,
    TcpConnectionInfo         serverInfo,
    org.omg.CORBA.Object      server,
    String                    operation,
    ServiceContext[]          services);}
```

アクセスコントローラ（たとえば、myAC）をプログラミングし、次のプロパティを使って GateKeeper にインストールできます。

```
vbroker.gatekeeper.security.accessControllers=myAC

vbroker.gatekeeper.security.acl.myAC.type=com.inprise.vbroker.gatekeeper.securi
ty.myACImpl
  vbroker.gatekeeper.security.acl.myAC.rules=
  vbroker.gatekeeper.security.acl.myAC.default=grant
```

アクセスコントローラは、次のように実装できます。

```
package com.inprise.vbroker.gatekeeper.security;
import java.util.*;
import java.io.*;
import com.inprise.vbroker.orb.TcpConnectionInfo;
import com.inprise.vbroker.orb.ORB;
import com.inprise.vbroker.IOP.ServiceContext;
public class myACImpl implements
ObjectAccessController, OperationAccessController {
  public void init(org.omg.CORBA.ORB orb, String prefix) {
  }
  public boolean isObjectAccessible(
TcpConnectionInfo clt, org.omg.CORBA.Object svr,
ServiceContext[] contexts, byte[] principal) {
    return true;
  }
  public boolean isOperationAccessible(
TcpConnectionInfo clt, TcpConnectionInfo svr,
org.omg.CORBA.Object server, String operation,
ServiceContext[] services) {
    return true;
```

```
    }
  }
```

アクセスコントロールのメソッドや規則は、インプリメンテーションによって定義できます。

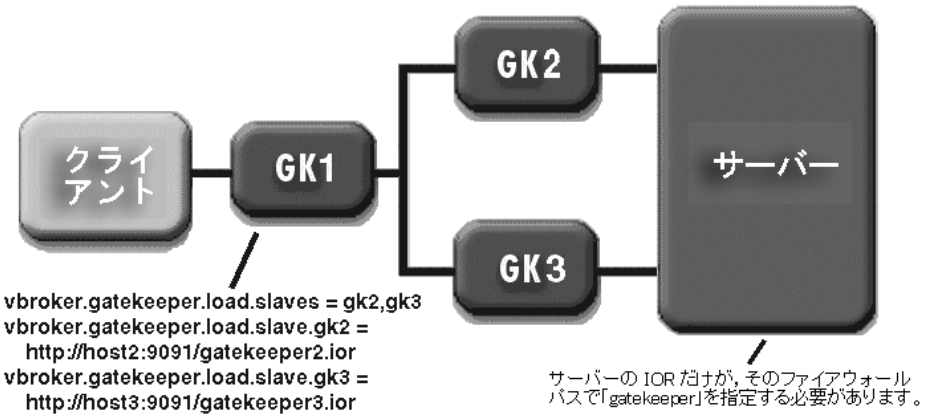
負荷分散とフォールトトレランス

GateKeeper は、内部ネットワークへの単一のアクセスポイントを提供するために使用されることがほとんどなので、非常に混雑したり、エラーの元になる可能性があります。Gatekeeper クラスタリングを利用して、ある程度のスケーラビリティとフォールトトレランスを実現すれば、これらの問題は解決できます。

負荷分散

マスター GateKeeper と 1 つ以上のスレーブ GateKeeper を組み合わせてクラスタを構成できます。マスター GateKeeper はスレーブ GateKeeper 間の負荷分散を担当します。サーバーがエキスポートするのは、マスター GateKeeper オブジェクトリファレンスだけです。

図 4.6　GateKeeper を使用した負荷分散



上の図は、GateKeeper1 とサーバーのプロパティ設定を示したものです。マスター GateKeeper は、スレーブ GateKeeper 間の負荷を、オブジェクトレベル単位で分散できます。オブジェクトレベルで、各クライアントは、負荷分散ポリシーに基づいてスレーブ GateKeeper のいずれかにリダイレクトされます。一般には、この方法で負荷の分散状態は公平になりますが、リソースの使用量が増えて速度が低下するおそれがあります。

デフォルトの負荷分散方式は、ラウンドロビン（総当たり）方式です。ただし、このポリシーをカスタマイズして標準パッケージとして利用することもできます。詳細については、Borland までお問い合わせください。

さらに GK2 と GK3 にも、独自のスレーブ Gatekeeper を設定できます。この設定では、マスターとスレーブの階層は相互にスタックできます。

GateKeeper におけるカスタム負荷分散

ORB のデフォルトの負荷分散インプリメンテーションでは、**ラウンドロビンアルゴリズム**を使用します。このアルゴリズムでは、クライアントの要求をサーバーと GateKeeper が順番に共有します。次のサンプルコードは、分散クラスのインプリメンテーションの例です。

```
package com.inprise.vbroker.gatekeeper.ext;
import java.util.Enumeration;
import org.omg.Firewall.GIOPProxy;
import com.inprise.vbroker.orb.*;
import com.inprise.vbroker.util.*;

public class MyDistributor implements Distributor {
       private Enumeration _enum;
       private UnGuardedVector _servers;
       public void init(ORB orb, UnGuardedVector v) {
              _servers = v;
              _enum = servers.elements();
       }
       public synchronized GIOPProxy next() {
       if (!_enum.hasMoreElements()) {
       _enum = _servers.elements();
       }
       return (GIOPProxy)_enum.nextElement();
       }
}
```

サーバーマネージャは、マスター／スレーブ設定内のほかの GateKeeper インスタンスの現在の負荷関連情報を収集できます。リアルタイムで利用できるサーバーマネージャからの情報に基づいて、マスター GateKeeper は、クライアントの要求をほかの GateKeeper に割り当て直すことができます。また、GateKeeper のフェデレーションは、負荷を分散するための負荷統計情報を交換できます。

フォールトトレランス

マスター GateKeeper と 1 つ以上のバックアップ GateKeeper でクラスタを構成して、見かけ上は 1 つの GateKeeper としてクライアントに提供することができます。GateKeeper をクラスタに構成する方法を次に紹介します。

- サーバーまでのさまざまなファイアウォールパスとして GateKeeper をクラスタに構成します。

  この構成は、Gatekeeper の動的チェイン化と同様の方法で設定できます。GateKeeper の設定を変更する必要はなく、サーバーリスナーにすべてのバックアップ GateKeeper をファイアウォールパスとして組み込むようにサーバーを設定するだけで済みます。ただし、この方法では、サーバー設定が複雑になります。

- すべてのバックアップ GateKeeper のオブジェクトリファレンス（プロファイル）をマスター GateKeeper のオブジェクトリファレンスに折りたたみます。マスター GateKeeper が失敗すると、クライアントはほかのバックアップ GateKeeper のいずれかに自動的にリバインドされます。この方法では、GateKeeper のオブジェクトリファレンスが非常に大きくなる可能性があります。Gatekeeper の負荷分散機能はこの方法を使用しています。

スケーラビリティとパフォーマンスに関するガイドライン

GateKeeper のパフォーマンスの評価では、GateKeeper 通信方式（クライアント - GateKeeper - サーバー）と直接通信方式（クライアント - サーバー）の比較が有効です。

メモ　この場合、パフォーマンスは応答時間で示され、スケーラビリティはスループットで表しています。

GateKeeper 通信方式では、接続が 2 本必要なので、呼び出しも 2 か所必要です。そのため、次のような影響が出ます。

- **スループットが小さい。**直接通信と比較して、50% ほど減少する可能性があります。
- **応答時間が遅い。**直接通信と比較して、応答時間は長くなります。場合によっては、200% まで長くなる可能性があります。

GateKeeper パフォーマンスの調整

GateKeeper では、パフォーマンスやスループットに関する新しいしきい値は発生しません。GateKeeper のパフォーマンスとスループットの特性は VisiBroker ORB と同じです。GateKeeper は CORBA アプリケーションなので、ORB の基本機能を継承します。したがって、ORB 固有のパフォーマンスチューニングパラメータは、すべて GateKeeper にも適用されます。ただし、次の各項で説明する機能は、GateKeeper のパフォーマンスに影響を与えます。

ビッドメカニズム

ユーザーによって設定された制約に基づいて特定のビッドを選択するようにクライアント側 ORB をプログラムできます。ビッド選択の順序を指定することで、接続の確立処理を高速化できます。

- **ビッドポートフォリオの制約**。GateKeeper を静的にチェイン化する際に排他的ビッドを設定する場合は、次のプロパティが有効です。

  ```
  vbroker.orb.alwaysProxy
  vbroker.orb.alwaysTunnel
  vbroker.orb.alwaysSecured
  ```

  たとえば、特定の GateKeeper が別の GateKeeper に静的にチェイン化されている場合は、vbroker.orb.alwaysProxy が有効です。GateKeeper をチェイン化する際に HTTP トンネリングだけを使用することが確実である場合は、vbroker.orb.alwaysTunnel プロパティを設定すると、不要なビッドを避けることができます。vbroker.orb.alwaysSecured プロパティが設定されている場合、GateKeeper は、チェイン化の際にだけセキュリティで保護された通信パスを使用します。これらのプロパティは外部 GateKeeper に設定されることに注意してください。

- **ビッド選択の順序**。ビッドの順序は、特定のビッドが選択される速さに影響します。たとえば、特定の GateKeeper で許可される接続のほとんどがセキュリティで保護されていることが確実な場合は、次に示すように、文字列の最初のエントリに SSL を入れることができます。

  ```
  vbroker.orb.bidOrder=inprocess:liop:ssl:iiop:proxy:hiop:locator
  ```

- **優先順位が高いビッドの指定**。最高の優先順位を持つビッドには、次のプロパティを設定できます。デフォルトでは、ORB の inprocess に設定されます。

  ```
  vbroker.orb.bids.critical=inprocess
  ```

キャッシュ管理

次のプロパティは、GateKeeper のキャッシュサイズを設定します。

```
vbroker.gatekeeper.cache.size=100
```

メッセージマーシャリング

ストリームのチャンクサイズを設定することで、GateKeeper とクライアント／サーバーアプリケーションの間で交換されるメッセージのサイズを大きくすることができます。チャンクサイズは、特に、HTTP トンネリングを使用するアプリケーションのパフォーマンスに大きく影響する場合があります。

```
vbroker.orb.streamChunkSize=4096
```

4096、8192、16384 などの値を使用してみてください。アプリケーションのパフォーマンスは、ネットワーク上のパケットの最大サイズに応じて変化します。

スレッド管理

GateKeeper の応答の必要性に応じて、スレッドプーリング、セッションごとのスレッドなど、さまざまなスレッド管理手法を適用できます。デフォルトでは、要求転送 IIOP サービスは ThreadPool を使用し、HIOP サービスは ThreadSession を使用します。

```
vbroker.se.exterior.scm.ex-iiop.dispatcher.type=ThreadPool
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMax=100
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMin=0
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMaxIdle=300

vbroker.se.interior.scm.in-iiop.dispatcher.type=ThreadPool
vbroker.se.interior.scm.in-iiop.dispatcher.threadMax=100
vbroker.se.interior.scm.in-iiop.dispatcher.threadMin=0
vbroker.se.interior.scm.in-iiop.dispatcher.threadMaxIdle=300

vbroker.se.exterior.scm.ex-hiop.dispatcher.type=ThreadSession
```

接続管理

双方向 GIOP では、前後両方向の通信で同じ通信パスを使用できます。したがって、コールバックを使用する場合は、vbroker.orb.enableBiDir プロパティ設定を使用することをお勧めします。次のプロパティを使用すると、接続リソースの使用を最適化できます（詳細については、付録 A を参照）。

```
vbroker.se.exterior.scm.ex-iiop.manager.connectionMax
vbroker.se.exterior.scm.ex-iiop.manager.connectionMaxIdle

vbroker.se.interior.scm.in-iiop.manager.connectionMax
vbroker.se.interior.scm.in-iiop.manager.connectionMaxIdle
```

GateKeeper は、潜在的に多数のクライアントに対する中間サービスであり、必要な数だけのサーバーに接続できる必要があるため、GateKeeper のコンテキスト内では vbroker.ce.iiop.ccm.connectionMax を使用しないでください。GateKeeper は、開くことができる発信接続の数が制限されていても、すでに接続されているクライアントがサーバーと接続できないようにすることはできません。そのかわり、GateKeeper は、次のプロパティを使用して、サービスを提供するクライアントの数を制限できます。

```
vbroker.se.exterior.scm.ex-iiop.manager.connectionMax
```

ただし、vbroker.ce.iiop.ccm.connectionMaxIdle を使用して、アイドル状態のサーバー接続を削除することができます。これは、GateKeeper が接続するサーバー数が多いと考えられる場合、接続するクライアントの数が少ない場合、およびクライアントが主に少数のサーバーだけをターゲットにする場合に特に有効です。

GateKeeper の非同期呼び出しの影響

GateKeeper の非同期呼び出しは、パフォーマンスやスケーラビリティに対してそれほど大きな影響はありません。

GateKeeper のパフォーマンスプロパティ

Gatekeeper のパフォーマンスに関係がある多くのプロパティがあります。ここでは、接続、スレッドの種類、オペレーションモード、呼び出しの種類に関連するプロパティについて説明します。

調整によってパフォーマンスを向上させることができるその他のプロパティについては、69 ページの「パフォーマンスと負荷分散」を参照してください。

接続の設定

GateKeeper の接続関連のプロパティを次に示します。

```
vbroker.se.<xxx>.scm.<yyy>.manager.connectionMax
vbroker.se.<xxx>.scm.<yyy>.manager.connectionMaxIdle
```

ここで、<xxx> と <yyy> は、「exterior、ex-hiop」、「exterior、ex-iiop」、「exterior、ex-hiops」、「exterior、ex-ssl」、「interior、in-iiop」、または「interior、in-ssl」です。

最初のプロパティは、アクティブな接続の最大許容数を指定します。接続を制限すると、GateKeeper のリソースを節約できますが、クライアントのパフォーマンスが低下する場合があります。デフォルトは、制限なしです。

2 番めのプロパティでは、非アクティブな接続を切断するまでに、どれだけの時間待機するかを指定します。デフォルトの 0 は、非アクティブな接続を切断しないことを表します。

スレッド関連の設定

ディスパッチャタイプが ThreadPool の場合、次の Gatekeeper のプロパティを調整できます。

```
vbroker.se.<xxx>.scm.<yyy>.dispatcher.threadMin
vbroker.se.<xxx>.scm.<yyy>.dispatcher.threadMax
vbroker.se.<xxx>.scm.<yyy>.dispatcher.threadMaxIdle
```

<xxx> と <yyy> の組み合わせの値としては、「exterior、ex-iiop」、「exterior, ex-ssl」、「interior, in_iiop」、または「interior, in_ssl」があります。

最初のプロパティ「threadMin」には、要求にすみやかに応答するための、事前に作成しておくスレッド数を指定します。デフォルトは 0 です。

2 番めのプロパティ「threadMax」では、過剰なスレッド数でシステムに負担がかかるのを避けるため、作成するスレッドの最大数を指定します。デフォルトは 100 です。スレッド数の不足で処理できなかった要求は、利用可能な次のスレッドを待機します。

3 番めのプロパティ「threadMaxIdle」では、スレッドを破棄するまでの待機時間を指定します（秒単位）デフォルトは 300 秒です。

GateKeeper のモード

Gatekeeper は通常モードでも、パススルーモードでも実行できます。パススルーモードではパフォーマンスが低下します。パケットのコンテンツはチェックされなくても Gatekeeper のリソースを消費するからです。実際、各パススルー接続には接続が続く限り排他的なポートが必要とされます。クライアントプロセスは、プログラム的にポリシーを使って排他的な接続を要求できます。

通常モードのオペレーションの方が、Gatekeeper のパフォーマンスが高くなります。

プロパティ vbroker.GateKeeper.enablePassthru=true を設定してパススルーを有効にしない限り、通常モードが使用されます。

呼び出しの種類

呼び出しには、次の 3 種類があります。

- 通常の一方向呼び出し
- 双方向コールバック
- VisiBroker 3.x スタイルのコールバック

双方向コールバックは、一方向の呼び出しとコールバックの両方で 1 つの接続を使用します。これは VisiBroker 3.x スタイルのコールバックより効率的です。

双方向コールバックの効率性は、通常の一方向呼び出しと同程度です。

GateKeeper ガイドと SSL

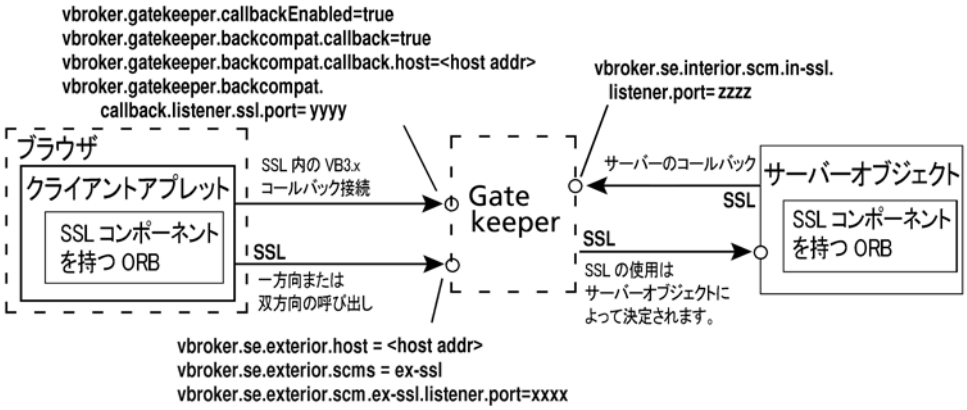
メモ SSL はオプションパッケージで使用できる独立したプロダクトです。そのため、アプレットやサーバーオブジェクトを SSL モードで実行する場合は ORB ランタイムに SSL コンポーネントを組み込む必要があります。

SSL で実行する GateKeeper ガイドには、次のようなセキュリティ機能があります。

- クライアントとサーバー間で IIOP/SSL 接続を中継する。
- HTTPS トンネリングをサポートする。
- IIOP/SSL コールバックを有効にする（VisiBroker 3.x スタイルおよび双方向）。
- サーバーのかわりに認証を実行する。
- 認証情報を転送する。

SSL を設定するためのその他のプロパティについては、付録 A「GateKeeper のプロパティ」を参照してください。

図 4.7 Gatekeeper との SSL 接続



Gatekeeper との SSL 接続

サーバーは、SSL 接続または通常の IIOP 接続のどちらを使用するかを決定します。SSL モードで実行されているクライアントでは、任意で SSL に接続するように要求できます。ただし、SSL モードで実行されているサーバーには、SSL モードでクライアントを接続する必要があります。

クライアントのプロパティファイルで vbroker.orb.alwaysSecure=true と設定されている場合、クライアントは常に SSL モードでサーバーや GateKeeper に接続され、ほかの種類の接続は行われません（ただし、サーバーや GateKeeper がほかの種類の接続を受け入れない場合はエラーになることがあります）。これで接続時間が短縮されます。

同様にこのプロパティを設定すると、Gatekeeper がサーバーに接続するときにも役立ちます。

送信呼び出しと双方向呼び出し用の SSL

次の GateKeeper プロパティを設定すると、クライアント（アプレット）からサーバーオブジェクトへの呼び出しで（Gatekeeper を介して）SSL を有効にできます。

```
vbroker.se.exterior.host = <host address>
vbroker.se.exterior.scms = ex-iiop, ex-hiops, ex-ssl
vbroker.se.exterior.scm.ex-ssl.listener.port = <port address>
```

アプレットクライアントは Gatekeeper への SSL 接続を開きます。
クライアントと Gatekeeper 間の通信チャネルは SSL モードです。ただし、Gatekeeper とサーバー間の通信チャネルのモードはサーバーによって決まります。サーバーの scm を SSL モードに設定すると Gatekeeper とサーバー間の通信チャネルは SSL モードになります。

双方向の呼び出しでは、同じ一方向の通信パスを使用します。ただし、双方向のコールバックには、さらに設定の必要なプロパティがあります。

GateKeeper におけるセキュリティサービスの有効化

セキュリティはデフォルトでオンになりますが、この機能は、セキュリティサービスのライセンスにだけ適用されます。ただし、セキュリティサービスをオンにするためのライセンスチェックはありません。

Borland VisiBroker では、次のプロパティで指定されように、セキュリティはデフォルトでオフになります。

```
vbroker.security.disable=true
```

VisiBroker で次のプロパティを設定すると、アプリケーションは、認証のためのユーザー名とパスワードの入力を求めます。

```
vbroker.security.login=true
```

*.config ファイル（サンプルは下記参照）を作成し、認証および領域関連のパラメータを指定する必要があります。

次に示すプロパティでは、セキュリティが有効の GateKeeper の一般的な例として、IIOP、IIOP/SSL、HIOP、および HIOPS のリスナーが有効にされています。

gatekeeper.config

```
System
  com.borland.security.provider.authn.HostLoginModule required REALM=myrealm
PRIMARYIDENTITY=true;

  com.borland.security.provider.authn.ClientSideDataCollection required
REALM=testrealm;

  };
```

```
myrealm
com.borland.security.provider.authn.HostLoginModule required;
};

anotherrealm {
com.borland.security.provider.authn.HostLoginModule required;
};
```

gatekeeper.properties

```
vbroker.security.disable=false
vbroker.security.peerAuthenticationMode=none
vbroker.security.secureTransport=false
vbroker.security.trustpointsRepository=Directory:./trustpoints
vbroker.gatekeeper.referenceStore=./gkclnt.ior
vbroker.orb.enableBiDir=both
vbroker.orb.dynamicLibs=com.borland.security.hiops.Init

  vbroker.se.exterior.scms=ex-iiop,ex-hiop,ex-ssl,ex-hiops
  vbroker.se.exterior.host=143.186.142.21
  vbroker.se.exterior.scm.ex-iiop.listener.port=25000
  vbroker.se.exterior.scm.ex-hiop.listener.port=25001
  vbroker.se.iiop_tp.scm.hiop_ts.listener.port=25002
  vbroker.se.exterior.scm.ex-ssl.listener.port=25003
  vbroker.se.exterior.scm.ex-hiops.listener.port=25004

  vbroker.se.interior.scms=in-iiop,in-hiop,in-ssl
  vbroker.se.interior.host=143.186.139.226
  vbroker.se.interior.scm.in-iiop.listener.port=15001

  vbroker.se.interior.scm.in-hiop.listener.port=15002

  vbroker.se.interior.scm.in-ssl.listener.port=15003

  # この GateKeeper を使ってコールバックを有効にします。

  vbroker.gatekeeper.callbackEnabled=true

  # VBJ3.x（古いスタイル）コールバックも有効にします。
  vbroker.gatekeeper.backcompat.callback=true
  vbroker.gatekeeper.backcompat.callback.host=143.186.142.21
  vbroker.gatekeeper.backcompat.callback.listeners=iiop,ssl
  vbroker.gatekeeper.backcompat.callback.listener.iiop.port=16001
  vbroker.gatekeeper.backcompat.callback.listener.iiop.type=IIOPCallback
  vbroker.gatekeeper.backcompat.callback.listener.ssl.port=16002
  vbroker.gatekeeper.backcompat.callback.listener.ssl.proxyPort=0
  vbroker.gatekeeper.backcompat.callback.listener.ssl.type=SSLCallback

  # オプション：GateKeeper 固有のアクセスコントロールプロパティを有効にします。
  vbroker.gatekeeper.security.accessControllers=myAC
  vbroker.gatekeeper.security.acl.myAC.default=grant
  vbroker.gatekeeper.security.acl.myAC.rules=rule1
  vbroker.gatekeeper.security.acl.myAC.rule1=grant [operation="*"]

  # オプション：GateKeeper の ID

  vbroker.security.wallet.identity=<username>
  vbroker.security.wallet.password=<password>
  vbroker.security.wallet.type=Directory:<path-to-identities>
```

次のサンプルのプロパティ設定では、クライアントが GateKeeper を使ってセキュリティで保護された転送を特別に要求しています。クライアントアプリケーションは、ユーザー名とパスワードを収集し、それを GateKeeper を介してサーバーに送信します。

client.config

```
System {
  com.borland.security.provider.authn.ClientSideDataCollection required
REALM=myrealm;
  };

  Client {
  com.borland.security.provider.authn.ClientSideDataCollection required;
  };
```

client.properties

```
vbroker.security.disable=false
  vbroker.security.login=true

vbroker.security.authentication.callbackHandler=com.borland.security.provider.a
uthn.HostCallbackHandler
  vbroker.security.authentication.config=client.config

  vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
  vbroker.se.iiop_tp.scms=iiop_tp,ssl
  vbroker.orb.alwaysProxy=true
  vbroker.orb.alwaysSecure=true
```

次のサンプルのプロパティ設定は、IIOP リスナーを無効にし、サーバーは、SSL トランスポートだけを使用するセキュリティで保護されたアプリケーションであるとみなされます。

server.config

```
System {
  com.borland.security.provider.authn.HostLoginModule required REALM=myrealm
PRIMARYIDEHostITY=true;
  com.borland.security.provider.authn.ClientSideDataCollection required
REALM=testrealm;
  };
myrealm {
  com.borland.security.provider.authn.HostLoginModule required;
  };
anotherrealm {
  com.borland.security.provider.authn.HostLoginModule required;
  };
```

server.properties

```
vbroker.security.disable=false
vbroker.security.login=true
vbroker.security.authentication.callbackHandler=com.borland.security.provider.a
uthn.HostCallbackHandler
vbroker.security.authentication.config=server.config
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.host=143.186.142.21
vbroker.se.iiop_tp.scm.iiop_tp.listener.type=Disabled-IIOP
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=25000
vbroker.se.iiop_tp.scm.ssl.listener.port=25005
vbroker.se.iiop_tp.firewallPaths=intranet
vbroker.firewall-path.intranet=first,second
vbroker.firewall-path.internet=first
vbroker.firewall.first.type=PROXY
vbroker.firewall.first.ior=http://localhost:16085/gatekeeper.ior
vbroker.firewall.second.type=TCP
vbroker.firewall.second.host=192.75.11.14
vbroker.firewall.second.iiop_port=32000
vbroker.firewall.second.hiop_port=32001
vbroker.firewall.second.ssl_port=32005
```

GateKeeper を介したネーミングサービスへのアクセスの有効化

固定された IP アドレスとポートでネーミングサービスを開始するには、次のプロパティを設定する必要があります。次のサンプルのネーミングサービスは、IP ホストアドレス 143.186.142.21 とリスナーポート 32101 で実行されています。32101:

namingservice.properties

```
vbroker.agent.addr=143.186.142.21
vbroker.agent.port=25873
vbroker.orb.logger.output=ns_debug.log

vbroker.naming.logLevel=7
vbroker.naming.iorFile=ns.ior

vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.host=143.186.142.21
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=32010
```

gatekeeper.properties

```
vbroker.agent.addr=143.186.142.21
vbroker.agent.port=25873
vbroker.agent.enableLocator=false

vbroker.orb.initRef=NameService=corbaloc::143.186.142.21:32010/NameService
vbroker.gatekeeper.referenceStore=gkclnt.ior
vbroker.se.exterior.host=143.186.142.21
vbroker.se.interior.host=143.186.139.226
vbroker.se.exterior.scm.ex-iiop.listener.port=25000
vbroker.se.exterior.scm.ex-hiop.listener.port=25001
vbroker.se.iiop_tp.scm.hiop_ts.listener.port=25002
```

第 5 章

GateKeeper のトラブルシューティング

ここでは、Gatekeeper とそのクライアントおよびサーバーから、デバッグ用の情報を取得する方法について説明します。また、不正な環境とレジストリの設定などで起こりうる問題と、Gatekeeper のトラブルシューティングによく使用されるツールについて説明します。

トラブルシューティングの準備

ここでは、GateKeeper のトラブルシューティングの前に実行または確認する必要がある事項について説明します。

デバッグ情報の取得

クライアント、サーバー、および Gatekeeper のプロパティを設定することで、包括的なデバッグ情報を取得できます。下の表に、関連する設定と、それらがクライアント、サーバー、および Gatekeeper のどれに適用されるかを示します。プロパティは、それぞれのプロパティファイルで設定します。

ログレベルには、数値とそれに対応するテキスト値があり、どちらかを使って状況が説明されます。レベル 4 以上は、デバッグの対象になります。次の表に、ログレベルの値を示します。

ログレベル値	説明
0 もしくは EMER	システムを使用できません。異常な状態です。
1 もしくは ALERT	すぐに修復が必要な状態です。たとえば、システムのデータベースが損傷した場合です。
2 もしくは CRIT	重大な状態です。たとえば、ハードデバイスにエラーが発生した場合です。
3 もしくは ERR	エラーが発生しています。
4 もしくは WARNING	認証の問題による接続障害などの警告状態です。たとえば、不正なパスワードを入力したり、承認が失敗した場合です。これはデフォルトです。
5 もしくは NOTICE	重大な状態ではありませんが、環境設定の変更が必要になる場合があります。

ログレベル値	説明
6 or INFO	サーバーに対して要求された特定のメソッドに対するすべてのユーザーアクセスなどの情報です。
7 or DEBUG	開発者だけが理解していればよいデバッグ情報です。これらのメッセージは国際化されません。

次の表に、GateKeeper のデバッグに役立つプロパティ設定を示します。

プロパティ	説明	プロパティファイルでの設定対象
vbroker.orb.logger.output=<filename>	**ログファイル** ログが記録されるファイル名を指定します。 指定しない場合、デフォルトは gkdebugfile.log になります。 ログ情報を stdout に送ることもできます。	サーバー、クライアント、Gatekeeper
vbroker.orb.bufferDebug=true	**ORB** ORB が使用するバッファを表示するように内部バッファマネージャに指示します。	サーバー、クライアント、Gatekeeper
vbroker.orb.debug=true	**ORB** ORB のデバッグ情報を表示します。	サーバー、クライアント、Gatekeeper
vbroker.orb.warn=<warning level>	**ORB** ORB が表示する警告の特定のレベルを指定します。0、1、2 のいずれかを指定できます。レベル 2 では、すべてのレベルの警告が表示されます。	サーバー、クライアント、Gatekeeper
vbroker.orb.logLevel=<loglevel>	**ORB** ログレベルは、上記のいずれかの値になります。	サーバー、クライアント、Gatekeeper
vbroker.orb.logger.appName=<Application Name>	**ORB** ログに表示されるアプリケーション名を指定します。	サーバー、クライアント、Gatekeeper
vbroker.events.debug=true	**イベントサービス** イベントサービスの診断メッセージを表示します。	イベントサービス
vbroker.orb.dynamicLibs=com.inprise.vbroker.gatekeeper.trace.Initvbroker.gatekeeper.trace.demo=true	**GateKeeper** GateKeeper の組み込みのトレース機能によるトレース情報を表示します。	GateKeeper
vbroker.agent.debug=true	**Gatekeeper とスマートエージェント** Gatekeeper とスマートエージェントとの間の対話のデバッグ情報を表示します。	GateKeeper
vbroker.locationservice.debug=true	**ロケーションサービス** ロケーションサービスのデバッグ情報を表示します。	サーバー、クライアント、Gatekeeper
vbroker.URLNaming.debug=true	**URLNaming** ORB ランタイムにロードされた URLNaming サービスのデバッグ情報を表示します。この設定は、正しい IOR が取得されたかどうかを検出するときに使用されます。	サーバー、クライアント、Gatekeeper
vbroker.poa.logLevel=emerg	**POA** POA のデバッグ情報を表示します。GateKeeper には、外部、内部、および iiop_tp の POA があります。	サーバーおよび Gatekeeper のサーバー側

プロパティ	説明	プロパティファイルでの設定対象
vbroker.gatekeeper.passthru.logLevel=<emerg>	**パススルー** Gatekeeper のパススルーモードのデバッグ情報を表示します。	GateKeeper
vbroker.security.logLevel=<logLevel>	**セキュリティサービス** GateKeeper の SSL などのセキュリティサービスのログ情報を表示します。	GateKeeper

デバッグモードの GateKeeper の起動

上記のプロパティに加えて、gatekeeper や vbj のコマンドラインユーティリティを使用して、起動時にほかの環境設定情報やパラメータ設定情報を出力できます。-VBJdebug オプションを使用すると、この情報が出力されます。次の表に、デバッグコマンドの例を示します。

コンポーネント	コマンドラインオプションの例
サーバー	vbj -VBJdebug -DORBpropStorage=server.prop Server
GateKeeper	gatekeeper -VBJdebug -J-Dvbroker.orb.debug=true -J-Dvbroker.orb.logLevel=7 -props gk.prop
クライアント	vbj -VBJdebug -DORBpropStorage=client.prop Client

メモ -VBJdebug オプションは、gatekeeper、vbj の各コマンドにだけ作用します。このオプションは、上記の診断プロパティの設定とは関係ありません。-VBJdebug オプションを使用するかどうかに関係なく、診断プロパティを使用すると同じ内容が出力されます。

環境設定

Gatekeeper は、起動時に環境変数を読み取ります。Windows では、レジストリの設定も読み取られます。UNIX および Windows での設定の優先順位は、次のとおりです。

1. コマンドライン
2. プロパティファイル
3. 環境設定
4. レジストリ設定（Windows のみ）
5. システムデフォルト設定（Windows のみ）

次の表は、Gatekeeper でよく使用される環境変数の一覧です。

環境変数	説明
CLASSPATH	CLASSPATH には、Java 開発者キットと Java サーブレット開発者キットのディレクトリを指定してください。具体的には、CLASSPATH に servlet.jar を追加します。たとえば、Windows NT の DOS プロンプトで、次のように入力します。 set CLASSPATH=C:¥visibroker¥lib¥tomcat¥common¥servlet.jar;c:¥bes¥jdk¥jdk1.4.1¥lib クラスパスに複数のバージョンの JDK が含まれている場合は、互換性の問題が発生する可能性があります。
JAVA_HOME	Java のホームディレクトリを指定します。CLASSPATH 環境変数に Java ディレクトリが定義されていない場合、システムはこのディレクトリから Java ライブラリを検索します。 **メモ:** UNIX システムでは、この変数を設定する必要があります。Windows システムでは、レジストリに事前に設定されます。

環境変数	説明
JDK_HOME	JDK のホームディレクトリを指定します。CLASSPATH 環境変数に Java ディレクトリが定義されていない場合、システムはこのディレクトリから Java ライブラリを検索します。 **メモ:** UNIX システムでは、この変数を設定する必要があります。Windows システムでは、レジストリに事前に設定されます。
PATH	PATH 環境変数は VisiBroker のインストール時に自動的に設定されます。この変数に、Gatekeeper が存在するディレクトリを追加する必要があります。たとえば、Windows のコマンドプロンプトで、次のように入力します。 `set PATH = c:¥bes¥vbroker¥bin`
VBROKERDIR	Gatekeeper 配布のホームディレクトリを指定します。VBROKERDIR は、VisiBroker の bin ディレクトリではなく、VisiBroker ディレクトリを指すことに注意してください。PATH 環境変数に %VBROKERDIR%¥bin が含まれていることを確認します。
BES_HOME	使用されていません。BES_HOME に変更されました。
OSAGENT_PORT	スマートエージェントとの通信に使用するポートを指定します。適切なスマートエージェントがこのポートを監視し、Gatekeeper がこのポートとホストに接続できるように設定してください。
BES_LIC_DIR	ライセンスデータファイルが存在するディレクトリのパスを指定します。

トラブルシューティングツール

次の表で、Gatekeeper のトラブルシューティングに役立つツールについて説明します。

名前	説明
osfind（Windows および UNIX）	VisiBroker に付属します。指定されたスマートエージェントドメインに登録されているすべてのオブジェクトを検索し、エージェントに通知されている情報を表示します。通常、スマートエージェントはサブネットに制限されます。
printior（Windows および UNIX）	VisiBroker に付属します。IOR にエンコードされているすべての情報を人が読み取ることができる形式で出力します。
ping（Windows および UNIX）	TCP/IP ネットワークツールセットの一部として、通常はオペレーティングシステムに付属しています。パケットがリモートホストから現在のホストに戻される場所の判定に使用します。また、ファイアウォール構成を検証する際も使用できます。
tracert (Windows) traceroute（UNIX）	TCP/IP ネットワークツールセットの一部として、一部のオペレーティングシステムと一緒に配布されます。データパケットが送信先ホストに到達するまでのルートまたはパスを出力します。これは、ルーターによるパケットの転送中にエラーが発生した場所を示すため、ファイアウォールに関する問題の特定に有効です。
route（Windows および UNIX）	TCP/IP ネットワークツールセットの一部として、一部のオペレーティングシステムと一緒に配布されます。ルーティングテーブルを出力します。ファイアウォールを構成する際に便利です。
netstat （Windows および UNIX）	TCP/IP ネットワークツールセットの一部として、一部のオペレーティングシステムと一緒に配布されます。これは、プロトコルの統計情報と TCP/IP ネットワークの接続状況を表示します。接続の状態とポートの有効性を確認する際に便利です。
nslookup （Windows および UNIX）	TCP/IP ネットワークツールセットの一部として、一部のオペレーティングシステムと一緒に配布されます。これは、インターネットドメインのネーミングサーバーにホスト名マッピングを照会します。
vregedit (Windows)	Borland から提供されています。これは、Borland 製品に関連する Windows レジストリのエントリを編集します。
regedit (Windows)	Microsoft Windows オペレーティングシステムに付属しています。Windows レジストリを編集できます。

コンピュータネットワークに関する情報の取得

Gatekeeper を正しく設定するには、コンピュータネットワークについての知識が必要です。Gatekeeper やファイアウォールまたはネットワーク自体の誤った設定によって発生する問題を特定するには、ネットワーク管理者との協力が必要です。ルーターやファイアウォールの設定が正しくないために問題が発生することはよくあります。

最初に、ネットワーク図、ファイアウォールポリシー、ルーティングテーブル、パケットフィルタ、および基本 TCP/IP スタックサーバーの場所と設定について理解する必要があります。通常、ネットワーク内の物理的な配線とコンポーネントが記載された論理ネットワーク図は、ネットワーク管理者が保管しています。GateKeeper のデプロイメント プランを作成する際は、このネットワーク図を分析および理解することから始めることをお勧めします。

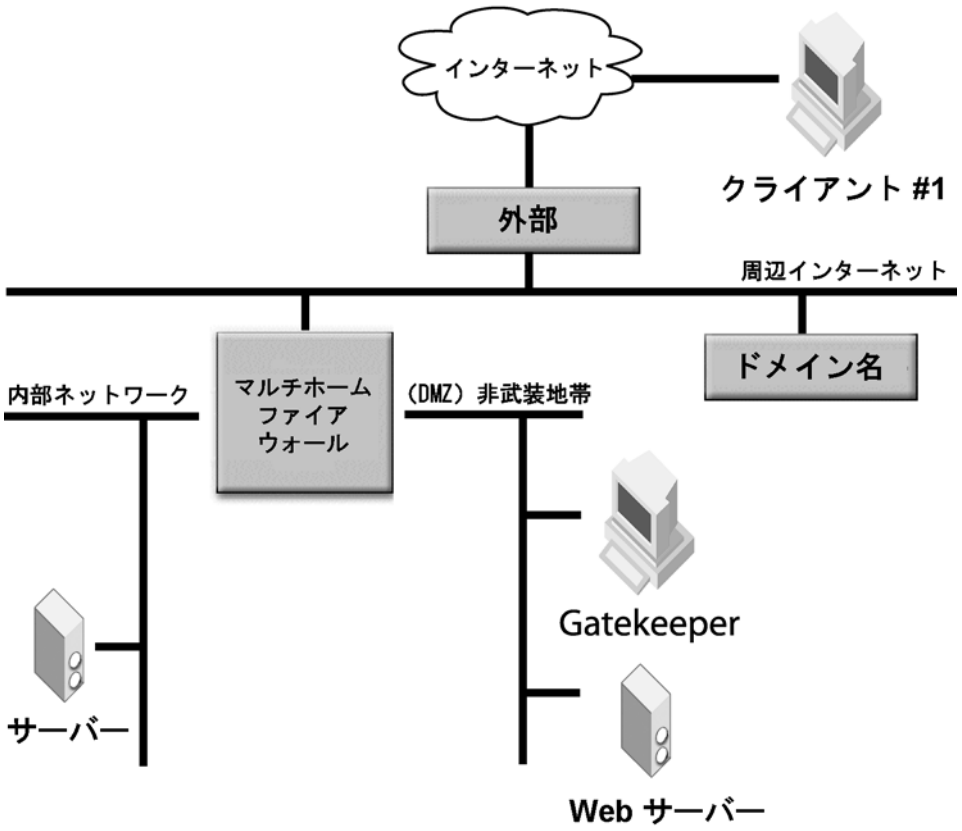
次に、設定されているファイアウォールポリシーを理解する必要があります。ファイアウォールポリシーと物理ネットワーク図を理解することで、クライアントアプリケーションからサーバーに、またはその逆方向にメッセージがさまざまなネットワークを通過して到達できるかどうかを判断できます。また、この情報から、GateKeeper をデプロイメントする必要がある場所を特定できます。これにより、GateKeeper の設定に関するトラブルシューティングに必要な時間が大幅に短縮されます。

外部ルーターは、インターネットと周辺ネットワークの間でパケットを転送します。また、外部ルーターのプログラミングにより、インターネットから周辺ネットワークに入るプロトコルを制限することもできます。この追加情報は、ファイアウォールポリシーにだけあります。ルートが正しく設定されていない場合、パケットは、間違った送信先に転送されるか無視されます。ルーティングテーブルやファイアウォールポリシーに変更があった場合は、ネットワーク管理者が開発者に通知する必要があります。

マルチホームファイアウォールは、パケットをフィルタリングし、周辺ネットワークから内部ネットワークや DMZ にパケットを転送します。また、内部ネットワーク内の真の IP アドレスを偽の IP アドレスに変換／逆変換する、ネットワークアドレス変換も行います。

次の図は、3 つのサブネット（周辺インターネット、DMZ、内部ネットワーク）の物理的な配線を示すネットワーク図の例です。

図 5.1　典型的なネットワーク図の例



- 外部ルーターは、インターネットと周辺インターネットとの間でパケットを転送します。また、外部ルーターのプログラミングにより、インターネットから周辺インターネットに入るプロトコルを少数に制限することもできます。この追加情報は、ファイアウォールポリシー（ルーティングテーブル）にだけあります。
- マルチホームファイアウォールは、パケットをフィルタリングし、周辺インターネットから 2 つのサブネット（内部ネットワークと DMZ）にパケットを転送します。また、内部ネットワーク内の真の IP アドレスを偽の IP アドレスに変換／逆変換する、ネットワークアドレス変換も行います。

メモ　上の図は、ネットワーク設定の 1 つの例にすぎません。そのため、GateKeeper をデプロイメントする前に、このような情報をどこから入手できるかを調べることが重要です。

基本的な確認項目

Gatekeeper はプロキシのように動作し、問題はクライアント、Gatekeeper、またはサーバーで発生します。Gatekeeper が適切に動作しない場合の基本的な確認項目を次に示します。以下の確認項目は包括的なものではなく、重要度や実用度の順には並んでいません。これらは、基本的なトラブルシューティングのガイドラインとして提供されています。

スマートエージェントのチェック

Gatekeeper は、スマートエージェントを使ってサーバーオブジェクトを検索します。また、ネットワーク上のスマートエージェントを自動的に検索できます。スマートエージェントがサーバーオブジェクトを検出できなかったり、GateKeeper がスマートエージェントを自動的に検索できなかった場合は、次のいずれかの方法でスマートエージェントのトラブルシューティングを実行します。

- 前述の環境変数設定を確認します。
- 次のように、デバッグモードでスマートエージェントを起動します。
  `osagent -v`
- GateKeeper がインストールされている場所から到達できるすべてのスマートエージェントを探します。`osfind` コマンドを使用できます。
- IP アドレスとポートがクライアント、Gatekeeper、およびサーバーで正しく設定されているかどうかを確認します。

プロパティファイルのチェック

クライアント、サーバー、および Gatekeeper のプロパティファイルの中の設定を確認します。よくある失敗は、ポートアドレスとホストアドレスの設定の誤りです。

ルーティングテーブルのチェック

マルチホームホストにより、接続されたネットワーク間で通信を行うことができます。マルチホームホストがデータパケットをあるネットワークから別のネットワークにルーティングするには、ホストでルーティングテーブルを正しく設定する必要があります。ルーティングテーブルがデータを正しく送信できない場合は、次の方法でこのプログラムのトラブルシューティングを実行できます。

- `route print` と `traceroute` を使用して、ルーティングテーブルを確認します。通信障害の場所を特定し、ルーティングテーブルを正しく設定します。
- `ping` や `tracert` などのツールを使用して、通信パスを調査および確認します。

パススルー接続のチェック

次のいずれかの方法で、パススルー接続が正しく設定されているかどうかを確認できます。

- Gatekeeper をパススルーモードで使用している場合は、Gatekeeper に対して次のプロパティを正しく設定する必要があります。

  ```
  vbroker.gatekeeper.passthru.inPortMin
  vbroker.gatekeeper.passthru.inPortMax
  vbroker.gatekeeper.passthru.outPortMin
  vbroker.gatekeeper.passthru.outPortMax
  ```

  inPortMin プロパティと *inPortMax* プロパティを使用して、クライアントが GateKeeper との接続に使用するポートの範囲を指定します。したがって、クライアントは、ファイアウォールを通過してこれらのポートに接続できる必要があります。

同様に、*outPortMin* プロパティと *outPortMax* プロパティを使用して、GateKeeper がサーバー側ネットワークとの接続に使用するポートの範囲を指定します。したがって、Gatekeeper は、ファイアウォールを通過してサーバー上のこれらのポートに接続できる必要があります。

- 着信先に接続できるかどうかを確認するには、ping、tracert、traceroute、route などのツールを使用します。

Java ポリシーのチェック

クライアントが Java プラグインを使用するアプレットの場合は、次のプロパティを java.policy ファイルに追加してください。これらの設定を JRE の java.policy ファイルに指定しないと、セキュリティ例外が発生することがあります。これらのプロパティはクライアントの設定であり、「111.222.333.444:25001」は、GateKeeper のホストと HIOP ポートの IP アドレスとポートです。

```
grant codeBase "http://192.73.8.25:25001/*" {
   permission java.lang.reflect.ReflectPermission"suppressAccessChecks";
   permission java.io.SerializablePermission "enableSubclassImplementation";
   permission java.lang.RuntimePermission "accessDeclaredMembers";
};
```

SSL のチェック

SSL を使用する場合は、クライアント（Web ブラウザ）、サーバー、および Gatekeeper に証明書が正しくインストールされているかどうかを確認します。セキュリティの詳細と Borland Security Service の使用方法については、Borland Enterprise Server の『開発者ガイド』を参照してください。

IOR ファイルのチェック

IOR ファイルの内容を確認するには、次の操作を実行します。

- クライアント、GateKeeper、またはサーバーで vbroker.URLNaming.debug プロパティを設定して、どの IOR ファイルが取得されるかをトレースします。
- printior コマンドで IOR ファイルの内容を出力します。

ファイアウォール設定のチェック

ファイアウォール設定は、問題が最も発生しやすい設定です。

- 14 ページの「ファイアウォールの設定」および第 5 章「GateKeeper のトラブルシューティング」を参照してください。
- ファイアウォールの制約について理解するために、ネットワーク管理者と緊密に協力して作業してください。
- NAT（ネットワークアドレス変換）の設定を確認します。

一般的なエラーと FAQ

1 よく寄せられる質問（FAQ）は、Borland Web サイトで「VisiBroker Gatekeeper FAQs」として包括的に管理されています。詳細については、このリストを参照してください。

2 プロパティの設定時のエラーは、プロパティ名のスペルミスが原因でよく発生します。たとえば、「vbroker」を「vroker」と入力してしまうなどです。また、Windows の一部のワードプロセッサは、行の最初の文字を自動的に大文字に変更してしまいます。そのため、vbroker が Vbroker になり、無効になることがあります。

3 IP アドレスやポートが不正またはすでに使用されている場合は、Socket Binding エラーが発生することがあります。次の表に、一般的なエラーを示します。

エラーメッセージ	エラーの種類	ソリューション
"Bind Exception: Cannot assign requested address"	Wrong IP Address	IP アドレスまたはポートを修正します。
"Bind Exception: Address in use"	IP port already in use	IP アドレスまたはポートを修正します。
"Invalid GIOP Proxy ior: Communication Problem"	Wrong IP Address	IP アドレスまたはポートを修正します。
"Invalid GIOP Proxy ior: Connect Exception"	Wrong Port Address	IOR ファイル（プロパティファイル）内の IP アドレスを修正します。
"Invalid GIOP Proxy ior: Invalid Object Ref, File Not Found Exception"	Wrong IOR Name	間違っているリファレンス IOR ファイル名を修正します。

プロキシサーバーと GateKeeper

GateKeeper を HTTP プロキシサーバーと組み合わせて使用できます。これらのプロキシサーバーは、GateKeeper に HTTP トンネリング機能を提供するために HIOP プロトコルによって使用されます。

通常、最新のファイアウォール製品には、HTTP 通信を処理する機能が組み込まれています。一部のファイアウォールには HTTP プロキシサーバーが組み込まれています（Microsoft の ISA Server など）。ほかのファイアウォールは、HTTP メッセージを HTTP プロキシサーバーに転送します。これらのプロキシサーバーは、専用のメカニズムで負荷分散を実行できます。また、キャッシュ技術を使ってパフォーマンスを向上させている HTTP プロキシサーバーもあります。GateKeeper は、そのメッセージに対して HTTP プロキシサーバーのキャッシュ機能を無効にするように要求します。

HTTP プロキシサーバーを GateKeeper と組み合わせて使用する場合、HTTP プロキシサーバーはパケットを転送するため、GateKeeper に対して NAT デバイスのように動作します。GateKeeper は HTTP プロキシサーバーの背後に隠されます。そのため、プロキシホストのプロパティまたは TCP ファイアウォールのプロパティを設定して、HIOP の偽ホスト／ポートを指定することが重要です。

付録 A

GateKeeper のプロパティ

この付録では、GateKeeper に設定できるプロパティを説明します（74 ページの「サーバーのファイアウォール仕様に関するプロパティ」は例外でサーバー上に設定します）。

メモ　以下の表の「デフォルト / オプション」列では、次の表記規則を使用します。

- オプションは太字で表示（例：**gatekeeper.ior**）
- < 空 > は空白または空の文字列
- 山括弧（< >）で囲まれたオプションには、ユーザーが値を指定（例：< ポート番号 >、< 整数値 >）

一般的なプロパティ

次の表は、GateKeeper でよく使用されるプロパティの一覧です。

プロパティ	デフォルト / オプション	説明
vbroker.gatekeeper.name	**null** - 名前の定義なし。< ユーザー定義名 >	他の GateKeeper インスタンスと区別するために、GateKeeper インスタンスの名前を指定します。
vbroker.gatekeeper.referenceStore	**gatekeeper.ior** - GateKeeper の現在のディレクトリ内。< 相対パス名 > < フル パス名 >	GateKeeper IOR ファイルの名前を指定します。このファイルが GateKeeper の現在のディレクトリに格納されていない場合は、そのファイルが存在する場所のフルパスを定義します。
vbroker.gatekeeper.locationService	**true** - 有効 **false** - 無効	GateKeeper を使用するロケーション サービスを有効または無効にします。このサービスは、スマート エージェント（OSAgent）と通信してバインドを行うことができないクライアント（アプレットなど）に提供されます。このプロパティが false の場合、クライアントは GateKeeper を介したバインド操作時に NO_PERMISSION 例外を受け取ります。
vbroker.gatekeeper.cache.size	**64**（デフォルト） **1** - キャッシュが無効 **0** - キャッシュ サイズが無制限	GateKeeper のキャッシュ サイズを定義します。

プロパティ	デフォルト / オプション	説明
vbroker.gatekeeper.cache.timeout	**900** < 整数値 >	キャッシュに情報を保存しておく時間を秒単位で指定します。この時間が経過しても使用されない情報は、不要な情報として回収されます。
vbroker.gatekeeper.asynchronizedIO	**false**（デフォルト）- 有効 **true** - 無効	GateKeeper の非同期化 IO 機能を有効または無効にします。この機能は、サーバー上のメソッドの呼び出しに時間がかかるため、新しい着信クライアントが多数ある場合にのみ効果的です。通常は、この機能を有効にしても利点はありません。この機能は、主に古いバージョンで使用されていたために現在も設定できるようになっていますが、使用自体はお勧めできません。

外部サーバー エンジン

次の表は、GateKeeper のクライアント側またはインターネット側にある外部サーバー エンジンで使用するプロパティの一覧です。ただし、重要なプロパティのほとんどは、各サーバー接続マネージャ（SCM）で定義します。SCM のプロパティについては、この後に説明します。

プロパティ	デフォルト	説明
vbroker.se.exterior.scms	**ex-iiop,ex-hiop** - 使用中の ex-iiop および ex-hiop サーバー接続マネージャ。ex-iiop、ex-hiop、ex-ssl、ex-hiops をカンマで区切ったリストも使用可能。	外部サーバー エンジンのサーバー接続マネージャを定義します。
vbroker.se.exterior.host	**null** - プライマリ ネットワーク インターフェイス カード（NIC）の IP アドレスであるプライマリ ホストを使用。< ホスト アドレス >	外部ホストのホスト アドレス。プライマリ NIC は外部 NIC です。このプロパティは、GateKeeper の <Basic Properties> パネルで設定できます。
vbroker.se.exterior.proxyHost	**< 空 >** - プロキシ ホストなし。< 偽ホスト アドレス >	ネットワーク アドレス変換（NAT）デバイスは、IP パケット内の IP アドレスやポート番号を変更することで、ネットワーク内の実際の IP アドレスやポート番号を隠します。この値には、NAT によって定義された値が設定されます。コールバックを有効にし、GateKeeper が NAT の後ろにある場合は、コールバック プロキシ ホスト（vbroker.gatekeeper.backcompat.callback.proxyHost）でもこのプロパティを同様に設定してください。このプロパティは、GateKeeper が NAT の後ろにある場合に使用します。また、設定は VisiBroker コンソールでも実行できます。
vbroker.se.exterior.type	gatekeeper	この設定により、外部サーバー エンジンが「プロキシ」の役割を果たします。つまり、このサーバー エンジン上でパケット / メッセージ転送機能を使用できます。この設定は変更しないでください。

ex-hiop サーバー接続マネージャ（SCM）

Java ex-hiop サーバー接続マネージャは、外部サーバー エンジンにおける HTTP リクエストへの応答を管理します。リスナーおよびディスパッチャのプロパティ設定には、どちらも vbroker.scm.exterior.ex-hiop から始まるプロパティを使用します。

次の vbroker.se.exterior.scm.ex-hiop プロパティは、ex-hiop リスナーの動作を指定します。ex-hiop リスナーは HIOP リスナーです。デフォルト ポートには 8088 を使用します。スレッド ポリシーは ThreadSession に設定します。

メモ SCM 関連のプロパティは、いずれも先頭に vbroker.se.<サーバー エンジン名>.scm.<サーバー接続マネージャ > を付けて定義されます。

一部の SCM には追加のプロパティが定義されていますが、特にスレッドや接続に関連するプロパティは、すべての SCM で同じ名前を使用します。

プロパティ	デフォルト	説明
vbroker.se.exterior.scm.ex-hiop.root	.< ファイル ディレクトリのフルパス >	デフォルトのルート ディレクトリを設定します。デフォルト ディレクトリは、GateKeeper が起動するディレクトリです。
vbroker.se.exterior.scm.ex-hiop.dispatcher.type	ThreadSession	ex-hiop scm 要求のディスパッチャの種類を指定します。この種類は、必ず「ThreadSession」に設定してください。
vbroker.se.exterior.scm.ex-hiop.listener.port	**8088** < ポート番号 >	GateKeeper のクライアント側（外部）HIOP リスナーのデフォルト リスナー ポートを設定します。クライアントのアプリケーションやアプレットは、このポートを主に HTTP トンネリングのサポートに使用します。また、それ以外にも GateKeeper IOR ファイルの取得など、標準の HTTP 要求のために限定的に使用することもあります。
vbroker.se.exterior.scm.ex-hiop.listener.proxyPort	< 空 > < 偽ポート番号 >	proxyPort プロパティは、一般にはサーバー エンジンの proxyHost プロパティと組み合わせて使用し、このリスナーの監視対象ポートを隠します。このプロパティを設定した場合、GateKeeper IOR ファイルのこのリスナーのエンドポイント情報に、proxyPort の値が追加されます。その後、外部 NAT デバイスが、proxyPort をリスナーの真のポートに対応付けます。デフォルトは < 空 > で、その場合、機能は無効です（リスナー ポートのマスクなし）。
vbroker.se.exterior.scm.ex-hiop.listener.type	HIOP	ex-hiops SCM のリスナーの種類を指定します。HIOP は、現在のリスナーが、HTTP プロトコルだけでなく、HTTP トンネリングに使用する Borland 独自の HIOP プロトコルもサポートすることを意味します。ex-hiop など、あらかじめ設定されているサーバー接続マネージャのリスナーの種類は変更しないでください。
vbroker.se.exterior.scm.ex-hiop.manager.connectionMax	**0** - 着信接続は無制限 < 整数値 >	GateKeeper の外部 HIOP リスナーが受け取ることができる着信接続の最大数を定義します。
vbroker.se.exterior.scm.ex-hiop.manager.connectionMaxIdle	**0** < 整数値 >	非アクティブな接続を閉じるまでの時間を秒単位で指定します。
vbroker.se.exterior.scm.ex-hiop.manager.type	Socket	サーバー接続マネージャの種類を指定します。
vbroker.se.exterior.scm.ex-hiop.servletList	orb	使用するサーブレット クラスの種類です。
vbroker.se.exterior.scm.ex-hiop.servlet.orb.GET	**true** - 有効 **false** - 無効	サーブレットの GET オペレーションを有効または無効にします。ORB は定義済みのプロパティであることに注意してください。
vbroker.se.exterior.scm.ex-hiop.servlet.orb.PUT	**true** - 有効 **false** - 無効	サーブレットの PUT オペレーションを有効または無効にします。ORB は定義済みのプロパティであることに注意してください。

プロパティ	デフォルト	説明
vbroker.se.exterior.scm.ex-hiop.servlet.orb.class	com.inprise.vbroker.HIOP.servlets.ORB.Servlet	GateKeeper をサーブレットとして使用する場合に HIOP 用にロードされる Java クラスを指定します。このプロパティは別の値に変更しないでください。
vbroker.se.exterior.scm.ex-hiop.servlet.orb.load	**false** - 無効 **true** - 有効	サーブレットの Load オペレーションを有効または無効にします。ORB は定義済みのプロパティであることに注意してください。

ex-iiop サーバー接続マネージャ（SCM）

ex-iiop サーバー接続マネージャは、外部サーバー エンジンにおける IIOP 要求への応答を管理します。リスナーおよびディスパッチャのプロパティ設定には、vbroker.se.exterior.scm.ex-iiop から始まるプロパティを使用します。次の vbroker.se.exterior.scm.ex-iiop プロパティは、ex-iiop リスナーの動作を指定します。ex-iiop リスナーは IIOP リスナーです。

プロパティ	デフォルト	説明
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMax	100 < 整数値 >	サーバー接続マネージャが作成できるスレッドの最大数を指定します。
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMaxIdle	300 < 整数値 >	アイドル状態のスレッドが破棄されるまでの時間を指定します。デフォルトは 300 です。
vbroker.se.exterior.scm.ex-iiop.dispatcher.threadMin	0 < 整数値 >	サーバー接続マネージャが作成できるスレッドの最小数を指定します。
vbroker.se.exterior.scm.ex-iiop.dispatcher.type	ThreadPool	ex-iiop scm のディスパッチャの種類を指定します。
vbroker.se.exterior.scm.ex-iiop.listener.giopVersion	1.2	GIOP メッセージによって使用される GIOP バージョンを設定します。このプロパティを使用すると、GIOP の未知のマイナー バージョンを正しく処理できない古いバージョンの ORB で発生する相互運用性の問題を解決できます。
vbroker.se.exterior.scm.ex-iiop.listener.port	683 < ポート番号 >	GateKeeper のクライアント側 IIOP リスナーのデフォルト リスナー ポートを設定します。ポート 683 は配布されたアプリケーションの推奨設定です（IIOP の OMG 標準であり、IANA に登録されています）。**UNIX :** UNIX プラットフォームでのデフォルトのリスナー ポート番号には、特権ユーザー用に予約されている 0 から 1024 の範囲を指定します。非特権ユーザーとして使用する場合は、必要に応じてリスナー ポートを 1024 より大きな値に設定できます。
vbroker.se.exterior.scm.ex-iiop.listener.proxyPort	**< 空 >** - プロキシポート機能が無効。 これは、機能が無効であることを示します（リスナー ポートのマスクなし）。 < 偽ポート番号 >	プロキシ ホスト名のプロパティとともに使用されるプロキシ ポート番号を定義します。
vbroker.se.exterior.scm.ex-iiop.listener.type	IIOP	ex-iiop scm のリスナーの種類を指定します。
vbroker.se.exterior.scm.ex-iiop.manager.connectionMax	**0** - 着信接続は無制限 < 整数値 >	GateKeeper の外部 IIOP リスナーが受け取ることができる着信接続の最大数を定義します。

プロパティ	デフォルト	説明
vbroker.se.exterior.scm.ex-iiop.manager.connectionMaxIdle	0 < 整数値 >	非アクティブな接続を閉じるまでの時間を秒単位で指定します。
vbroker.se.exterior.scm.ex-iiop.manager.type	Socket	サーバー接続マネージャの種類を指定します。現在は、「Socket」のみを指定できます。

ex-hiops サーバー接続マネージャ（SCM）

ex-hiops サーバー接続マネージャは、外部サーバー エンジンにおける HTTP 要求への応答を管理します。リスナーおよびディスパッチャのプロパティ設定には、どちらも vbroker.scm.exterior.ex-hiops から始まるプロパティを使用します。

次の vbroker.se.exterior.scm.ex-hiops プロパティは、ex-hiops リスナーの動作を指定します。ex-hiops リスナーは HIOPS リスナーです。デフォルト ポートには 8089 を使用します。スレッド ポリシーは、常に ThreadSession です。

プロパティ	デフォルト	説明
vbroker.se.exterior.scm.ex-hiops.root	.< ファイル ディレクトリのフルパス >	デフォルトのルート ディレクトリを設定します。デフォルト ディレクトリは、GateKeeper が起動するディレクトリです。
vbroker.se.exterior.scm.ex-hiops.dispatcher.type	ThreadSession	ex-hiop scm 要求のディスパッチャの種類を指定します。この種類は必ず「ThreadSession」に設定してください。
vbroker.se.exterior.scm.ex-hiops.listener.port	8089 < ポート番号 >	GateKeeper のクライアント側（外部）HIOPS リスナーのデフォルト リスナー ポートを設定します。クライアントのアプリケーションやアプレットは、HTTPS トンネリングのサポートなどにこのポートを使用します。また、このポートを GateKeeper IOR ファイルの取得など、標準の HTTPS 要求のために限定的に使用することもあります。
vbroker.se.exterior.scm.ex-hiops.listener.proxyPort	**< 空 >** < 偽ポート番号 >	proxyPort プロパティは、一般にはサーバー エンジンの proxyHost プロパティと組み合わせて使用し、このリスナーの監視対象ポートを隠します。このプロパティを設定した場合、GateKeeper IOR ファイルのこのリスナーのエンドポイント情報に、proxyPort の値が追加されます。その後、外部 NAT デバイスが、proxyPort をリスナーの真のポートに対応付けます。デフォルトは < 空 > で、その場合、機能は無効です（リスナー ポートのマスクなし）。
vbroker.se.exterior.scm.ex-hiops.listener.type	HIOPS	ex-hiops SCM のリスナーの種類を指定します。HIOPS は、現在のリスナーが、HTTPS プロトコルだけでなく、HTTP トンネリングに使用する Borland 独自の HIOPS プロトコルもサポートすることを意味します。ex-hiops など、あらかじめ設定されているサーバー接続マネージャの種類は変更しないでください。
vbroker.se.exterior.scm.ex-hiops.manager.connectionMax	0 - キャッシュ接続が無制限 < 整数値 >	GateKeeper の外部 HIOPS リスナーが使用できるキャッシュ接続の最大数を定義します。
vbroker.se.exterior.scm.ex-hiops.manager.connectionMaxIdle	0 < 整数値 >	非アクティブな接続を閉じるまでの時間を秒単位で指定します。

プロパティ	デフォルト	説明
vbroker.se.exterior.scm.ex-hiops.manager.type	Socket	サーバー接続マネージャの種類を指定します。
vbroker.se.exterior.scm.ex-hiops.servletList	orb	使用するサーブレット クラスの種類です。
vbroker.se.exterior.scm.ex-hiops.servlet.orb.GET	**true** - 有効 **false** - 無効	サーブレットの GET オペレーションを有効または無効にします。ORB は定義済みのプロパティであることに注意してください。
vbroker.se.exterior.scm.ex-hiops.servlet.orb.PUT	**true** - 有効 **false** - 無効	サーブレットの PUT オペレーションを有効または無効にします。ORB は定義済みのプロパティであることに注意してください。
vbroker.se.exterior.scm.ex-hiops.servlet.orb.class	com.inprise.vbroker.HIOP.servlets.ORB.Servlet	GateKeeper をサーブレットとして使用する場合に HIOPS 用にロードされる Java クラスを指定します。このプロパティは別の値に変更しないでください。
vbroker.se.exterior.scm.ex-hiops.servlet.orb.load	**false** - 無効 **true** - 有効	サーブレットの Load オペレーションを有効または無効にします。ORB は定義済みのプロパティであることに注意してください。

ex-ssl サーバー接続マネージャ（SCM）

ex-ssl サーバー接続マネージャは、外部サーバー エンジンにおける SSL 要求への応答を管理します。リスナーおよびディスパッチャのプロパティ設定には、vbroker.se-exterior.scm.ex-ssl から始まるプロパティを使用します。

次の vbroker.se.exterior.scm.ex-ssl プロパティは、ex-ssl リスナーの動作を指定します。ex-ssl リスナーは SSL リスナーです。

プロパティ	デフォルト	説明
vbroker.se.exterior.scm.ex-ssl.dispatcher.threadMax	100 < 整数値 >	サーバー接続マネージャが作成できるスレッドの最大数を指定します。
vbroker.se.exterior.scm.ex-ssl.dispatcher.threadMaxIdle	300 < 整数値 >	アイドル状態のスレッドが破棄されるまでの時間を指定します。デフォルトは 300 です。
vbroker.se.exterior.scm.ex-ssl.dispatcher.threadMin	0 < 整数値 >	サーバー接続マネージャが作成できるスレッドの最小数を指定します。
vbroker.se.exterior.scm.ex-ssl.dispatcher.type	ThreadPool	ex-iiop scm のディスパッチャの種類を指定します。
vbroker.se.exterior.scm.ex-ssl.listener.port	684 < ポート番号 >	GateKeeper のクライアント側 SSL リスナーのデフォルト リスナー ポートを設定します。ポート 684 は配布されたアプリケーションの推奨設定です（IIOP の OMG 標準であり、IANA に登録されています）。 **UNIX :** UNIX プラットフォームでのデフォルトのリスナー ポート番号には、特権ユーザー用に予約されている 0 から 1024 の範囲を指定します。非特権ユーザーとして使用する場合は、必要に応じてリスナー ポートを 1024 より大きな値に設定できます。
vbroker.se.exterior.scm.ex-ssl.listener.proxyPort	**< 空 >** - プロキシポート機能が無効。これは、機能が無効であることを示します（リスナーポートのマスクなし）。 < 偽ポート番号 >	プロキシ ホスト名のプロパティとともに使用されるプロキシ ポート番号を定義します。

プロパティ	デフォルト	説明
vbroker.se.exterior.scm.ex-ssl.listener.type	SSL	ex-ssl scm のリスナーの種類を指定します。
vbroker.se.exterior.scm.ex-ssl.manager.connectionMax	0 - キャッシュ接続が無制限。 < 整数値 >	GateKeeper の外部 SSL リスナーが使用できるキャッシュ接続の最大数を定義します。
vbroker.se.exterior.scm.ex-ssl.manager.connectionMaxIdle	0 < 整数値 >	非アクティブな接続を閉じるまでの時間を秒単位で指定します。
vbroker.se.exterior.scm.ex-ssl.manager.type	Socket	サーバー接続マネージャの種類を指定します。現在は、「Socket」のみを指定できます。

内部サーバー エンジン

次の表は、GateKeeper のサーバー側またはイントラネット側にある内部サーバー エンジンが使用するプロパティの一覧です。

GateKeeper をデュアル ホームマシンで稼動させている場合や、GateKeeper とサーバーの間にネットワーク アドレス変換（NAT）を使用している場合など、特殊な環境では内部サーバー エンジンのプロパティの一部を設定する必要があります。

プロパティ	デフォルト	説明
vbroker.se.interior.scms	in-iiop in-hiop in-ssl in-hiops < 上記の値をカンマ区切りで組み合わせ >	このプロパティは、サーバー エンジンのサーバー接続マネージャを定義します。デフォルト値は IIOP scm です。ただし、個々の scm を指定することで、SSL など、別の種類のプロトコルを選択することもできます。
vbroker.se.interior.host	Null - プライマリ ネットワーク インターフェイス カード（NIC）の IP アドレスであるプライマリ ホストを使用。 < ホスト アドレス >	内部サーバー エンジンのホスト アドレス。このプロパティは、VisiBroker コンソールの Basic Properties パネルでも設定できます。
vbroker.se.interior.proxyHost	< 空 > - プロキシポート機能が無効。 < 偽ホストアドレス >	GateKeeper とサーバーの間で NAT を実行しており、サーバー ホストのアドレスを不可視にする場合に、このプロパティを使用します。このプロパティは、VisiBroker コンソールの Basic Properties パネルでも設定できます。

in-iiop サーバー接続マネージャ（SCM）

in-iiop サーバー マネージャは、内部サーバー エンジンにおける IIOP 要求への応答を管理します。リスナーおよびディスパッチャの設定には、vbroker.se.interior.in-iiop から始まるプロパティを使用します。

次の vbroker.se.interior.scm.in-iiop プロパティは、in-iiop サーバー接続マネージャの動作を指定します。

プロパティ	デフォルト	説明
vbroker.se.interior.scm.in-iiop.dispatcher.threadMax	100 < 整数値 >	サーバー接続マネージャが作成できるスレッドの最大数を指定します。
vbroker.se.interior.scm.in-iiop.dispatcher.threadMaxIdle	300 < 整数値 >	アイドル状態のスレッドが破棄されるまでの時間を指定します。デフォルトは 300 です。
vbroker.se.interior.scm.in-iiop.dispatcher.threadMin	0 < 整数値 >	サーバー接続マネージャが作成できるスレッドの最小数を指定します。
vbroker.se.interior.scm.in-iiop.dispatcher.type	ThreadPool	in-iiop scm のディスパッチャの種類を指定します。この種類は必ず「ThreadPool」に設定してください。
vbroker.se.interior.scm.in-iiop.listener.giopVersion	1.2	GIOP メッセージに使用するプロトコル バージョン番号
vbroker.se.interior.scm.in-iiop.listener.port	**0** - ランダムに番号を選択 < ポート番号 >	ホスト名のプロパティで使用するポート番号を定義します。
vbroker.se.interior.scm.in-iiop.listener.proxyPort	**< 空 >** - プロキシポート機能が無効。 < 偽ポート番号 >	プロキシ ホスト名のプロパティとともに使用されるプロキシ ポート番号を定義します。
vbroker.se.interior.scm.in-iiop.listener.type	IIOP	in-iiop scm のリスナーの種類を指定します。
vbroker.se.interior.scm.in-iiop.manager.connectionMax	**0** - キャッシュ接続が無制限。 < 整数値 >	GateKeeper IIOP リスナーに使用できるキャッシュ接続の最大数を定義します。
vbroker.se.interior.scm.in-iiop.manager.connectionMaxIdle	0 < 整数値 >	非アクティブな接続を閉じるまでの時間を秒単位で指定します。
vbroker.se.interior.scm.ex-iiop.manager.type	Socket	サーバー接続マネージャの種類を指定します。現在は、「Socket」のみを指定できます。

in-ssl サーバー接続マネージャ（SCM）

in-ssl サーバー マネージャは、内部サーバー エンジンにおける SSL 要求への応答を管理します。リスナーおよびディスパッチャの設定には、vbroker.se.interior.in-ssl から始まるプロパティを使用します。

次の vbroker.se.interior.scm.in-ssl プロパティは、in-ssl サーバー接続マネージャの動作を指定します。

プロパティ	デフォルト	説明
vbroker.se.interior.scm.in-ssl.dispatcher.threadMax	100 < 整数値 >	サーバー接続マネージャが作成できるスレッドの最大数を指定します。
vbroker.se.interior.scm.in-ssl.dispatcher.threadMaxIdle	300 < 整数値 >	アイドル状態のスレッドが破棄されるまでの時間を指定します。デフォルトは 300 です。
vbroker.se.interior.scm.in-ssl.dispatcher.threadMin	0 < 整数値 >	サーバー接続マネージャが作成できるスレッドの最小数を指定します。
vbroker.se.interior.scm.in-ssl.dispatcher.type	ThreadPool	in-iiop scm のディスパッチャの種類を指定します。この種類は必ず「ThreadPool」に設定してください。
vbroker.se.interior.scm.in-ssl.listener.port	**0** - ランダムに番号を選択 < ポート番号 >	ホスト名のプロパティで使用するポート番号を定義します。
vbroker.se.interior.scm.in-ssl.listener.proxyPort	**< 空 >** - プロキシポート機能が無効。 < 偽ポート番号 >	プロキシ ホスト名のプロパティとともに使用されるプロキシ ポート番号を定義します。
vbroker.se.interior.scm.in-ssl.listener.type	SSL	in-ssl scm のリスナーの種類を指定します。
vbroker.se.interior.scm.in-ssl.manager.connectionMax	**0** - 着信接続は無制限 < 整数値 >	GateKeeper の内部 SSL リスナーが受け取ることができる着信接続の最大数を定義します。
vbroker.se.interior.scm.in-ssl.manager.connectionMaxIdle	0 < 整数値 >	非アクティブな接続を閉じるまでの時間を秒単位で指定します。
vbroker.se.interior.scm.ex-ssl.manager.type	Socket	サーバー接続マネージャの種類を指定します。現在は、「Socket」のみを指定できます。

管理

Java　次の表は、管理プロパティの一覧です。デフォルトのリスナー ポート番号は 9091 です。

プロパティ	デフォルト / オプション	説明
vbroker.se.iiop_tp.host	**null** - システムのホスト アドレスを使用。 < ホスト アドレス >	このサーバー エンジンが使用できるホスト アドレスを指定します。
vbroker.se.iiop_tp.ProxyHost	**< 空 >** - システムのホスト アドレスを使用。 < プロキシ ホスト アドレス >	このサーバー エンジンが使用できるプロキシ ホスト アドレスを指定します。
vbroker.se.iiop_tp.scms	**iiop_tp、hiop_ts**	サーバー接続マネージャの名前の一覧を指定します。
vbroker.se.iiop_tp.scm.iiop_tp.listener-port	**< 空 >** < ポート番号 >	IIOP の管理リスナー ポートを指定します。
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort	**< 空 >** < ポート番号 >	IIOP 管理リスナー ポートのプロキシ ポートを指定します。
vbroker.se.iiop_tp.scm.hiop_ts.listener.port	**9091** < ポート番号 >	GateKeeper の管理リスナー ポートを指定します。
vbroker.se.iiop_tp.scm.hiop_ts.listener.proxyPort	**< 空 >** - プロキシ ポート機能が無効。 < 偽ポート番号 >	HIOP 管理リスナー ポートのプロキシ ポート番号を指定します。
vbroker.se.iiop_tp.type	normal	管理サーバー エンジンに指定できます。
vbroker.se.iiop_tp.scm.hiop_ts.servletList	orb	サーブレット クラスに与えられる仮想名。PUT、GET、LOAD などの他のプロパティの指定時に使用します。
vbroker.se.iiop_tp.scm.hiop_ts.servlet.orb.GET	**true** - 有効 **false** - 無効	サーブレットの GET オペレーションを有効または無効にします。
vbroker.se.iiop_tp.scm.hiop_ts.servlet.orb.PUT	**true** - 有効 **false** - 無効	サーブレットの PUT オペレーションを有効または無効にします。
vbroker.se.iiop_tp.scm.hiop_ts.servlet.orb.class	com.inprise.vbroker.HIOP.servlets.ORBServlet	サーブレット クラスの名前。
vbroker.se.iiop_tp.scm.hiop_ts.servlet.orb.load	**true** - 有効 **false** - 無効	サーブレットの Load オペレーションを有効または無効にします。

アクセス コントロール

次の表は、GateKeeper のセキュリティ制御の設定に使用するプロパティの一覧です。

プロパティ	デフォルト	説明
vbroker.gatekeeper. security. accessControllers	default	アクセス コントローラの名前の一覧を指定します。
vbroker.gatekeeper. security.acl. <controllerName>.default	**null** - デフォルトアクションなし **deny** - エントリを拒絶 **grant** - 許可を付与	コントロール リストのデフォルトの処理を指定します。 <controllerName> は上記のプロパティで指定します。
vbroker.gatekeeper. security.acl. <controllerName>.<xx> この <xx> には指定の規則名が入ります。	**null** - アクションの指定なし。追加オプションについては、説明を参照してください。	指定の規則に対する特定のプロパティのアクションを定義します。定義は次のとおりです。 <deny|grant> [operation="<**operation name**>" [signer by="<**signer's company name**>"] [server host="<**hostname**>"] [client host="<**hostname**>"] [server ip=**aa.bb.cc.dd**|<**sub-mask**>] [client ip=**aa.bb.cc.dd**|<**sub-mask**>] [object interface="<repId>"]] <deny | grant> は、各規則に対するアクションを定義します。 operation="<**operation name**>" は、IDL に基づくオペレーション名を定義します。 signer by="<**signer's company name**>" は、署名者の会社名を定義します。 server host="<**hostname**>" は、サーバーのホスト名を指定します。 client host="<**hostname**>" は、クライアントのホスト名を指定します。 server ip="<**aa.bb.cc.dd**>" は、サーバーが存在するマシンの IP アドレスを指定します。 client ip="<**aa.bb.cc.dd**>" は、クライアントが存在するマシンの IP アドレスを定義します。 object interface="<repId>" は、オブジェクトの種類を定義します。 例： vbroker.gatekeeper.security.accessControllers=default vbroker.gatekeeper.security.acl.default.rules=rule1,rule2,rule3 vbroker.gatekeeper.security.acl.default.rule1=*grant* [operation=\"*"\ [server host=\"*borland*"\]] vbroker.gatekeeper.security.acl.default.rule2=*deny* [operation=\"*"\ [client ip=*192.168.100.40*|*255.255.255.0*]] vbroker.gatekeeper.security.acl.default.rule3=*deny* [operation=\"*"\ [server host=\"*inprise*"\] [client ip=*192.168.100.88*|*255.255.255.0*]] **メモ：** **太字**の変数はユーザー定義。
vbroker.gatekeeper. security.acl. <controllerName>.type	com.inprise.vbroker.gatekeeper.security.ACImpl	GateKeeper によってロードされるアクセス コントロールの実装クラスを指定します。 **メモ：**この値は変更しないでください。
vbroker.gatekeeper. security.acl. <controllerName>.rules	**null** - 規則の指定なし(説明を参照)。	規則のセットに対する名前を指定します。例を次に示します。 vbroker.gatekeeper.security.accessControllers=default vbroker.gatekeeper.security.acl.default.rules=rule1,rule2,rule3 この **rule1、rule2、rule3** は、ユーザーが定義する名前です。

VisiBroker 3.x スタイルのコールバック

次の表は、VisiBroker 3.x スタイルのコールバックを使用する場合に、VisiBroker 5.x で設定可能なプロパティの一覧です。

プロパティ	デフォルト / オプション	説明
vbroker.gatekeeper.callbackEnabled	**false** - 無効 **true** - 有効	GateKeeper を介したコールバック機構を有効または無効にします。デフォルトは**無効**に設定されています。
vbroker.gatekeeper.backcompat.callback	**false** - 無効 **true** - 有効	VisiBroker 3.x スタイルのコールバックを有効または無効にします。古いスタイルのコールバックを使用する場合は、上記のプロパティとこのプロパティの両方を **true** に設定してください。
vbroker.gatekeeper.backcompat.callback.listeners	iiop ssl	リスナーの一覧を指定します。
vbroker.gatekeeper.backcompat.callback.listener.iiop.type	IIOPCallback	IIOP のコールバックリスナーの種類を指定します。
vbroker.gatekeeper.backcompat.callback.host	**< 空 >** - 値をプライマリホストの IP アドレスに設定。これは、プライマリ NIC の IP アドレスです。 < ホスト アドレス >	コールバック リスナーにバインドするホスト IP アドレスを定義します。
vbroker.gatekeeper.backcompat.callback.proxyHost	< 空 > - プロキシ ホストを使用せず。 < 偽ホスト アドレス >	このプロパティは、一般にはサーバー エンジンの proxyPort プロパティと組み合わせて使用し、このリスナーの監視対象ポートを隠します。このプロパティを設定した場合、コールバック IOR は、proxyHost の値をこのリスナーのエンドポイント情報に追加します。その後、ネットワークアドレス変換 (NAT) デバイスが、proxyHost をリスナーの真のポートにマッピングします。デフォルトは **< 空 >** であり、その場合機能は無効です（コールバック IOR でリスナー ポートのマスクなし）。
vbroker.gatekeeper.backcompat.callback.listener.ssl.type	SSLCallback	SSL リスナーの種類を指定します。
vbroker.gatekeeper.backcompat.callback.listener.ssl.port	**0** - ポートをランダムに選択 < ポート番号 >	コールバック リスナーが SSL の監視に使用するポートを定義します。
vbroker.gatekeeper.backcompat.callback.listener.ssl.proxyPort	< 空 > - プロキシ ポートを使用しない。 < 偽ポート番号 >	このプロパティは、クライアントと GateKeeper の間に NAT が存在し、NAT が実際の GateKeeper ホスト アドレスを隠す場合に使用します。

プロパティ	デフォルト / オプション	説明
vbroker.gatekeeper.backcompat.callback.listener.iiop.port	**0** - ポート番号をランダムに選択。 < ポート番号 >	コールバック リスナーが IIOP を監視するポートを指定します。
vbroker.gatekeeper.backcompat.callback.listener.iiop.proxyPort	**< 空 >** - プロキシ ポートを使用しない。 < 偽ポート番号 >	proxyPort プロパティは、一般にはサーバー エンジンの proxyHost プロパティと組み合わせて使用し、このリスナーの監視対象ポートを隠します。このプロパティを設定した場合、コールバック IOR は、proxyPort の値をこのリスナーのエンドポイント情報に追加します。その後、ネットワークアドレス変換 (NAT) デバイスが、proxyPort をリスナーの真のポートにマッピングします。デフォルトは **< 空 >** であり、その場合機能は無効です (コールバック IOR でリスナー ポートのマスクなし)。

パフォーマンスと負荷分散

次の表は、クライアントとサーバー間の負荷を分散させ、負荷を監視するためのパフォーマンス プロパティと負荷分散プロパティをまとめたものです。

プロパティ	デフォルト	説明
vbroker.gatekeeper.load.distributor	com.inprise.vbroker.gatekeeper.ext.RoundRobinDistributor	負荷分散マネージャが使用する分散クラスを指定します。ディストリビュータ インターフェイスの内容をよく理解したユーザーが独自のディストリビュータを実装する場合を除き、このプロパティを変更しないでください。デフォルトのディストリビュータとして、次の実装クラスが使用されます。(Round Robin):com.inprise.vbroker.gatekeeper.ext.RoundRobinDistributor
vbroker.ce.iiop.ccm.connectionMax	**0** - 古いアクティブ接続やキャッシュ接続を閉じない。 < 整数値 >	クライアントごとの接続総数の最大数を指定します。この値は、アクティブな接続の数とキャッシュされた接続の数の合計です。
vbroker.orb.gcTimeout	30 < 整数値 >	使用されていない重要なリソースが消去されるまでの時間を秒単位で指定します。
vbroker.orb.fragmentSize	0	GIOP のフラグメント サイズを指定します。GIOP ストリームのチャンク サイズの倍数を指定する必要があります。
vbroker.orb.streamChunkSize	**4096** <2 の累乗数 >	GIOP メッセージのチャンク サイズを指定します。2 の累乗数を指定してください。
vbroker.orb.bufferCacheTimeout	**6000** < 整数値 >	メッセージ チャンクをキャッシュしてから破棄するまでの時間をミリ秒単位で指定します。

プロパティ	デフォルト	説明
vbroker.gatekeeper.load.slaves	< 空 > - スレーブ GateKeeper なし < スレーブのリスト > - 説明を参照	このマスター GateKeeper の負荷分散で、クラスタリングに使用するスレーブ GateKeeper の一覧を指定します。名前の一覧はカンマで区切ります。一覧に含まれる名前ごとに、「vbroker.gatekeeper.load.slave.<slave_name>」などのプロパティがプロパティ ファイルに追加されます。このプロパティにデフォルト値はありません。このプロパティを使用するのは、マスター GateKeeper プロパティのみです。たとえば、次のようになります。 vbroker.gatekeeper.load.slaves=abc,xyz. **メモ：**このプロパティを設定した場合は、対応するライブラリをロードする必要があります。75 ページの「その他の ORB プロパティ」の vbroker.orb.dynamicLibs の説明を参照してください。
vbroker.gatekeeper.load.slave.<slave_name>	< 空 > < 特定のスレーブ IOR> - 説明を参照	このマスター GateKeeper の負荷分散で、クラスタリングに使用する特定のスレーブ GateKeeper を指す IOR または URL を指定します。このプロパティを使用するのはマスター GateKeeper プロパティのみです。たとえば、次のようになります。 vbroker.gatekeeper.load.slave.abc=http:// host1:9091/GateKeeper.ior vbroker.gatekeeper.load.slave.xyz=http:// host2:9091/GateKeeper.ior. **メモ：**このプロパティを設定した場合は、対応するライブラリをロードする必要があります。75 ページの「その他の ORB プロパティ」の vbroker.orb.dynamicLibs の説明を参照してください。
vbroker.gatekeeper.load.balancer	< 空 > < マスター >	マスター GateKeeper は負荷分散のみを目的にし、クライアントにサービスを提供しないことを指定します。デフォルト モードでは、マスター自体もスレーブです。つまり、マスターも利用可能な GatrKeeper のリストに含まれ、ほかのスレーブと交代に動作します。特定の GateKeeper が利用できなくなると、クライアントはマスターに戻り、かわりに次のスレーブ GateKeeper（スレーブ群の 1 つまたはマスター自身）を取得します。 **メモ：**このプロパティを設定した場合は、対応するライブラリをロードする必要があります。75 ページの「その他の ORB プロパティ」の vbroker.orb.dynamicLibs の説明を参照してください。

双方向通信のサポート

次の表は、双方向通信をサポートするプロパティの一覧です。これらのプロパティは、SCM の作成時に一度だけ評価されます。すべての場合で、SCM の exportBiDir プロパティと importBiDir プロパティは、enableBiDir プロパティより優先します。つまり、両方のプロパティに相反する値を設定した場合、SCM 固有のプロパティが適用されます。そのため enableBiDir プロパティをグローバルに設定したあと、SCM で個別に双方向をオフにすることが可能です。

プロパティ	デフォルト	説明
vbroker.orb.enableBiDir	サーバーとクライアントの両方 none 32 ページの「Gatekeeper の双方向サポートによるコールバック」を参照してください。	双方向接続は、それぞれ選択できます。クライアントが vbroker.orb.enableBiDir=client と定義し、サーバーが vbroker.orb.enableBiDir=server と定義している場合は、GateKeeper の vbroker.orb.enableBiDir の値によって接続状態が決定されます。 **メモ：** SCM 単位で双方向通信を選択して有効にできるのと同様に、GateKeeper でも双方向通信を選択して有効にできます。たとえば、vbroker.se.exterior.scm.ex--iiop.manager.importBiDir プロパティを **true** に設定すると、GateKeeper はクライアントからの双方向接続を受け付けます。vbroker.se.exterior.scm.ex--iiop.manager.exportBiDir プロパティを **true** に設定すると、GateKeeper はサーバーとの双方向接続を要求します。

パススルー接続のサポート

メモ パススルー接続をサポートするプロパティは、vbroker.gatekeeper.enablePassthru のみです。

プロパティ	デフォルト	説明
vbroker.gatekeeper.enablePassthru	**false** - 無効 **true** - 有効	GateKeeper のパススルー モードを有効または無効にします。
vbroker.gatekeeper.passthru.blockSize	**16384** < 整数値 >	チャネルがそれぞれの読み取り / 書き込みオペレーションで使用するバッファ サイズを指定します。値を大きくすると、一度の読み取り / 書き込みで大きなメッセージを処理できますが、1 つのチャネルで使用されるリソースが増加します。値を小さくすると、リソースの使用を最適化できますが、何度も読み取り / 書き込みが繰り返されるためパフォーマンスが低下します。
vbroker.gatekeeper.passthru.connectionTimeout	**300000 ミリ秒** (5 分) < 整数値 >	指定したチャネルが接続待機を終了してチャネルを停止するまでの待ち時間をミリ秒単位で指定します。
vbroker.gatekeeper.passthru.inPortMin	**1024** < ポート番号 >	vbroker.gatekeeper.passthru.inPortMax と組み合わせて使用します。パススルー着信接続用の内部ポートの範囲の先頭を指定します。
vbroker.gatekeeper.passthru.inPortMax	**65535** < ポート番号 >	vbroker.gatekeeper.passthru.inPortMin と組み合わせて使用します。パススルー着信接続用のポートの範囲の末尾を指定します。
vbroker.gatekeeper.passthru.logLevel	**0** - ログなし < 整数値 >	パススルー コンポーネントのログのレベルを有効にします。
vbroker.gatekeeper.passthru.outPortMin	**0** < ポート番号 >	vbroker.gatekeeper.passthru.outPortMax と組み合わせて使用します。パススルー発信接続用の外部ポートの範囲の先頭を指定します。

プロパティ	デフォルト	説明
vbroker.gatekeeper.passthru.outPortMax	**65535** < ポート番号 >	vbroker.gatekeeper.passthru.outPortMin と組み合わせて使用します。パススルー発信接続用の外部ポートの範囲の末尾を指定します。
vbroker.gatekeeper.passthru.streamTimeout	**2000** < 整数値 >	確立したチャネルが閉じられるまでのメッセージの待機時間をミリ秒単位で指定します。

セキュリティ サービス（SSL）

次の表は、セキュリティ サービスで使用されるプロパティの一覧です。

メモ このプロパティを設定した場合は、対応するライブラリをロードしてください。75 ページの「その他の ORB プロパティ」の vbroker.orb.dynamicLibs の説明を参照してください。

プロパティ	デフォルト	説明
vbroker.orb.alwaysSecure	**false** - 無効 **true** - 有効	GateKeeper にセキュリティで保護された接続をサーバーと確立させるかどうかを指定します。
vbroker.security.peerAuthenticationMode	説明を参照	詳細については、**『VisiBroker セキュリティ ガイド』**の「セキュリティ プロパティ（Java）」または「セキュリティ プロパティ（C++）」を参照してください。
vbroker.security.trustpointsRepository	説明を参照	詳細については、**『VisiBroker セキュリティ ガイド』**の「セキュリティ プロパティ（Java）」または「セキュリティ プロパティ（C++）」を参照してください。
vbroker.security.wallet.identity	説明を参照	詳細については、**『VisiBroker セキュリティ ガイド』**の「セキュリティ プロパティ（Java）」または「セキュリティ プロパティ（C++）」を参照してください。
vbroker.security.wallet.password	説明を参照	詳細については、**『VisiBroker セキュリティ ガイド』**の「セキュリティ プロパティ（Java）」または「セキュリティ プロパティ（C++）」を参照してください。

ロケーション サービス（スマート エージェント）

次の表は、サーバー オブジェクトを検出するためにロケーション サービスで使用するスマート エージェント（OSAgent）のプロパティの一覧です。

プロパティ	デフォルト	説明
vbroker.agent.addr	null - 説明を参照	スマート エージェント（OSAgent）が稼動しているホストの IP アドレスまたはホスト名を指定します。デフォルト値の null は、VisiBroker アプリケーションに OSAGENT_ADDR 環境変数の値を使用するように指示します。OSAGENT_ADDR 変数が設定されていない場合は、スマート エージェントがローカル ホストで稼動しているとみなされるか、ブロードキャスト メッセージによって検索されます。詳細については、**『VisiBroker for C++ 開発者ガイド』**または**『VisiBroker for Java 開発者ガイド』**の「スマート エージェントの使い方」を参照してください。
vbroker.agent.port	null	使用しているネットワーク上のドメインを定義するポート番号を指定します。VisiBroker アプリケーションとスマート エージェント（OSAgent）のポート番号が同じ場合は、相互が連動して機能します。このプロパティは、OSAGENT_PORT 環境変数と同じです。 詳細については、**『VisiBroker for C++ 開発者ガイド』**または**『VisiBroker for Java 開発者ガイド』**の「スマート エージェントの使い方」を参照してください。
vbroker.agent.addrFile	null	スマート エージェントの IP アドレスまたはホスト名の情報を格納するファイルを指定します。詳細については、**『VisiBroker for C++ 開発者ガイド』**または**『VisiBroker for Java 開発者ガイド』**の「スマート エージェントの使い方」を参照してください。
vbroker.agent.failOver	true false	true に設定すると、VisiBroker アプリケーションを他のスマート エージェントにフェイルオーバーさせることができます。詳細については、『VisiBroker for C++ 開発者ガイド』または**『VisiBroker for Java 開発者ガイド』**の「スマート エージェントの使い方」を参照してください。
vbroker.agent.enableCache	true false	true に設定すると、VisiBroker アプリケーションにオブジェクト リファレンスをキャッシュさせることができます。また、true に設定した場合、サーバー検索時のパフォーマンスは向上しますが、スマート エージェントのラウンドロビン アクティビティは無効になります。詳細については、**『VisiBroker for C++ 開発者ガイド』**または**『VisiBroker for Java 開発者ガイド』**の「スマート エージェントの使い方」を参照してください。

VisiBroker 4.x 以前との下位互換性

GateKeeper バージョン 5.x は、デフォルトの状態では、VisiBroker 4.x 以前のバージョンで開発されたプログラムと互換性がありません。GateKeeper バージョン 5.x を、VisiBroker 4.x 以前のバージョンで開発されたプログラムで正しく動作させるには、次のプロパティを **true** に設定してください。

メモ GateKeeper の古いバージョンは、VisiBroker 4.x 以前のバージョンで開発された古いプログラムとデフォルトの状態で互換性があります。ただし、GateKeeper 5.x では、このプロパティを明示的に設定する必要があります。

プロパティ	デフォルト	説明
vbroker.orb.enableVB4backcompat	true false	GateKeeper に VisiBroker の旧バージョンとの互換性を持たせるかどうかを指定します。このプロパティを **false** に設定した場合、VisiBroker 4.5.x 以降のバージョンで開発したプログラムに対しては互換性が保たれます。このプロパティを **true** に設定した場合、VisiBroker 4.5.x 以前のバージョンとの互換性も保たれます （詳細については、付録 B を参照してください）。 **メモ：**GateKeeper では、このプロパティはデフォルトで **true** に設定されています。ただし、クライアントとサーバーでは、この値はデフォルトで **false** に設定されています。

サーバーのファイアウォール仕様に関するプロパティ

メモ これらのプロパティは、サーバーのプロパティ ファイルでのみ設定する必要があります。これらのプロパティを設定した場合は、対応するライブラリをロードしてください。また、75 ページの「その他の ORB プロパティ」の vbroker.orb.dynamicLibs の説明を参照してください。

次のプロパティは、クライアントからサーバーへの通信経路を指定します。使用方法（例文付き）については、27 ページの「サーバーまでの通信パスの指定」を参照してください。

プロパティ	デフォルト	説明
vbroker.se.iiop_tp.firewallPaths	**< 空 >** < パス一覧 >	クライアントからサーバーへの通信経路の一覧を指定します。< パス一覧 > は、ユーザーが定義するパス名で、パスはカンマで区切ります。< パス一覧 > の例を次に示します。 vbroker.se.iiop_tp.firewallPaths=x,y
vbroker.firewall-path.<pathname>	**< 空 >** < コンポーネント一覧 >	ファイアウォールパス < パス名 > 内のコンポーネントの一覧を指定します。たとえば、次のようにします。 vbroker.firewall-path.x=a,b vbroker. firewall-path.y=c
vbroker.firewall.<component>.type	**< 空 >** PROXY TCP	コンポーネントの種類を指定します。たとえば、次のようになります。 vbroker.firewall.a.type = PROXY vbroker.firewall.b.type = TCP
vbroker.firewall.<component>.ior	**< 空 >** <ior ファイルのファイル名 > <ior ファイルの URL> IOR：<GateKeeper の文字列化 ior>	コンポーネントの ior を指定します。これは、vbroker.firewall.<component>.type=PROXY と組み合わせて指定します。次に値の例を示します。 1. file:C:/GateKeeper/GateKeeper.ior 2. http://www.inprise.com/GK GateKeeper.ior 3. IOR:239840284172907342349723423423 4
vbroker.firewall.<component>.host	**< 空 >** < 偽ホスト名 >	サーバーの偽ホストを指定します。これは、vbroker.firewall.<component>.types=TCP と組み合わせて指定し、コンポーネントは NAT 使用の TCP ファイアウォールです。

プロパティ	デフォルト	説明
vbroker.firewall.<component>.iiop_port	**< 空 >** < 偽 IIOP ポート >	サーバーの偽 IIOP ポートを指定します。これは、vbroker.firewall.<component>.types=TCP と組み合わせて指定し、コンポーネントは NAT 使用の TCP ファイアウォールです。
vbroker.firewall.<component>.ssl_port	**< 空 >** < 偽 SSL ポート >	サーバーの偽 SSL ポートを指定します。これは、vbroker.firewall.< コンポーネント >.type=TCP とともに指定し、コンポーネントは NAT 使用の TCP ファイアウォールです。
vbroker.firewall.<component>.hiop_port	**< 空 >** < 偽 HIOP ポート >	サーバーの偽 HIOP ポートを指定します。これは、vbroker.firewall.< コンポーネント >.type=TCP とともに指定し、コンポーネントは NAT 使用の TCP ファイアウォールです。

その他の ORB プロパティ

これらのプロパティは ORB でよく使用されており、直接的および間接的に GateKeeper に関連します。GateKeeper のプロパティ ファイルで設定されていない場合もあるため、説明文には注意して目を通してください。

プロパティ	デフォルト	説明
vbroker.orb.gatekeeper.ior	**< 空 >** <ior ファイル名 >	GateKeeper IOR ファイルの URL を指定します。このプロパティによってアプレット ビューアの動作が変化するため、アプレットを含む HTML ファイルで設定する必要があります。
vbroker.orb.alwaysProxy	**false** - 無効 **true** - 有効	クライアントがサーバーに接続するときに、必ず GateKeeper を経由するかどうかを指定します。このプロパティは、クライアントまたは GateKeeper で設定できます。クライアントで設定すると、クライアントは必ず GateKeeper を経由してサーバーに接続します。GateKeeper で設定すると、クライアントは必ず別の GateKeeper を介してサーバーに接続します。詳細については、**『VisiBroker プログラマーズ リファレンス』**を参照してください。
vbroker.locator.ior.ior	**< 空 >** <ior ファイル名 >	GateKeeper の IOR ファイルの URL を指定します。このプロパティは、通常、クライアント アプレットで設定しますが、アプリケーションでも設定できます。 **メモ：**GateKeeper が提供するロケーション サービスには制限があります。ロケーション要求を別の GateKeeper に転送することはできません。これとは対照的に、スマート エージェントでは、利用可能な別のスマート エージェントに要求を転送できます。
vbroker.orb.alwaysTunnel	**false** - 無効 **true** - 有効	クライアントがサーバーに接続するときに、必ず HTTP トンネル（IIOP ラッパー）を使用するかどうかを指定します。このプロパティは、クライアントまたは GateKeeper で設定できます。詳細については、**『VisiBroker プログラマーズ リファレンス』**を参照してください。
vbroker.orb.dynamicLibs	**< 空 >** < ライブラリの一覧 > 説明を参照	ライブラリの一覧を指定します。ここでは、GateKeeper と関連のあるライブラリについてのみ説明します。 ファイアウォール コンポーネントを指定する場合は、クライアントとサーバーのプロパティで、この値を次のように設定する必要があります。 com.inprise.vbroker.firewall.Init 負荷分散コンポーネントを指定する場合は、GateKeeper のプロパティ ファイルで、この値を次のように設定する必要があります。 com.inprise.vbroker.gatekeeper.ext.Init

付録 B

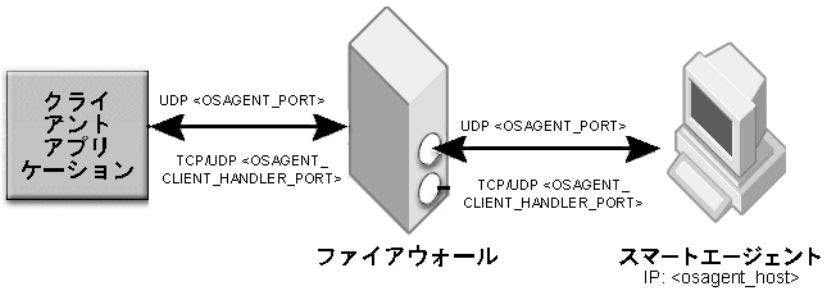
GateKeeper のデプロイメント例

この付録では、Gatekeeper を使用する場合と使用しない場合について、マルチネットワーク環境の一般的なデプロイメント構成をいくつか示します。

TCP ファイアウォール（Gatekeeper を使用しない）

構成例 1.1：ファイアウォール背後にスマートエージェント

この構成は、クライアントオブジェクトを設定して、ファイアウォールの背後にあるスマートエージェントにアクセスする方法を示します。



クライアントの環境設定（環境変数または Windows レジストリを使用）：

```
OSAGENT_PORT
```

クライアントのプロパティ：

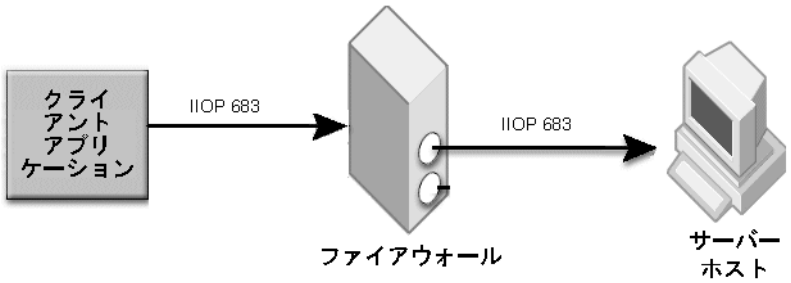
```
vbroker.agent.addr=<osagent_host>
vbroker.agent.port=<OSAGENT_PORT>
```

ファイアウォールの設定：

ポート <OSAGENT_PORT> で、クライアントホストとスマートエージェントホスト間の双方向の UDP パケットを有効にします。

ポート <OSAGENT_CLIENT_HANDLER_PORT> で、クライアントホストとスマートエージェントホスト間の双方向の TCP パケットと UDP パケットを有効にします。

構成例 1.2：IIOP 通信の使用



クライアントのプロパティ：不要

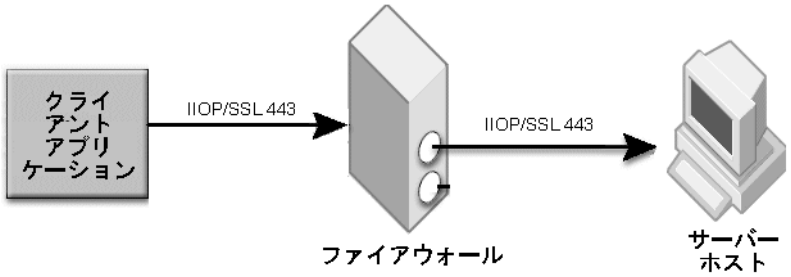
サーバーのプロパティ：

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
```

ファイアウォールの設定：

ポート 683 で、クライアントホストからサーバーホストへの TCP パケットを有効にします。

構成例 1.3：IIOP/SSL 通信の使用



クライアントのプロパティ：

```
# セキュリティサービスの有効化
vbroker.security.disable=false

# セキュリティで保護された転送をクライアント側で適用
vbroker.security.alwaysSecure=true

# peerAuthenticationMode を設定
vbroker.security.peerAuthenticationMode=REQUIRE_AND_TRUST
vbroker.security.trustpointsRepository=Directory:./trustpoints
```

サーバーのプロパティ：

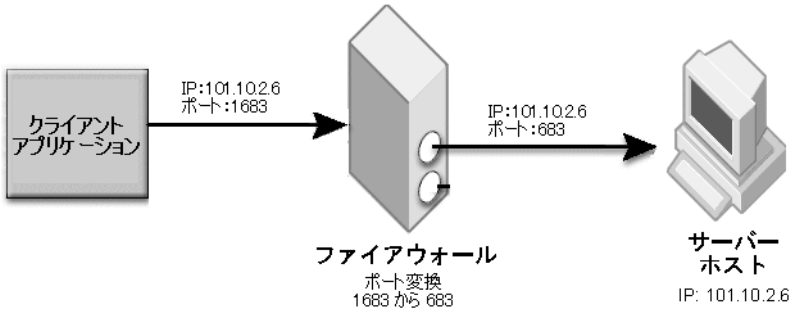
```
# セキュリティサービスの有効化
vbroker.security.disable=false

# SSL 層の属性を設定
vbroker.security.peerAuthenticationMode=REQUIRE_AND_TRUST
vbroker.security.trustpointsRepository=Directory:./trustpoints

# 443 の SSL リスナーポートを設定
vbroker.se.iiop_tp.scms=iiop_tp,ssl
vbroker.se.iiop_tp.scm.ssl.listener.port=443
vbroker.se.iiop_tp.scm.iiop_tp.listener.type=Disabled-IIOP
```

メモ このサンプルプロパティでは、<install_dir>/examples/vbroker/security/bank_ssl のサンプルのようなセキュリティで保護されたクライアントとサーバーに、有効な証明書情報がすでに読み込まれていることを前提としています。

ファイアウォールの設定：

ポート 443 で、クライアントホストからサーバーホストへの SSL パケットを有効にします。

構成例 1.4：ファイアウォールでアドレス変換のみ実行



ファイアウォールの設定：

アドレス変換：199.10.9.6 → 101.10.2.6

サーバーのプロパティ：次の 2 つの方法のどちらかを使用します。

方法 1：IIOP プロファイルの使用

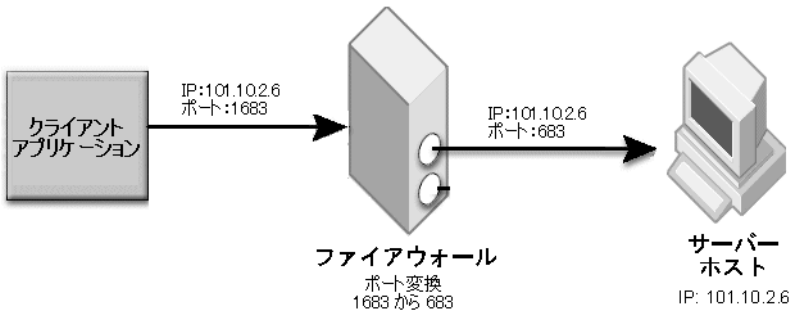
```
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.proxyHost=199.10.9.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
```

方法 2：ファイアウォールコンポーネントの使用

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw
vbroker.firewall.fw.type=TCP
vbroker.firewall.fw.host=199.10.9.6
vbroker.firewall.fw.iiop_port=683
vbroker.firewall.fw.hiop_port=0
```

メモ　ポート変換を行わない場合は、実ポートを指定してください。リスナーポートが無効になっている場合は 0 を指定します。

構成例 1.5：ファイアウォールでポート変換のみ実行



ファイアウォールの設定：

ポート変換：1683 → 683

サーバーのプロパティ：次の 2 つの方法のどちらかを使用します。

方法 1：IIOP プロファイルの使用

```
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=1683
```

方法 2：ファイアウォールコンポーネントの使用

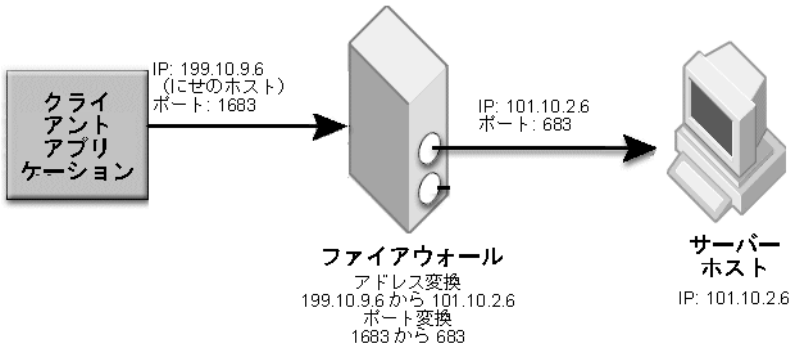
```
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw
vbroker.firewall.fw.type=TCP
vbroker.firewall.fw.host=101.10.2.6
vbroker.firewall.fw.iiop_port=1683
vbroker.firewall.fw.hiop_port=0
```

メモ　アドレス変換を行わない場合は実ホストを指定してください。

方法 2 では、次をクライアントのプロパティに追加してください。

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
```

構成例 1.6：ファイアウォールがアドレス変換とポート変換の両方を実行



ファイアウォールがアドレス変換とポート変換の両方を実行する場合は、構成 1.4 と 1.5 の設定を組み合わせてください。

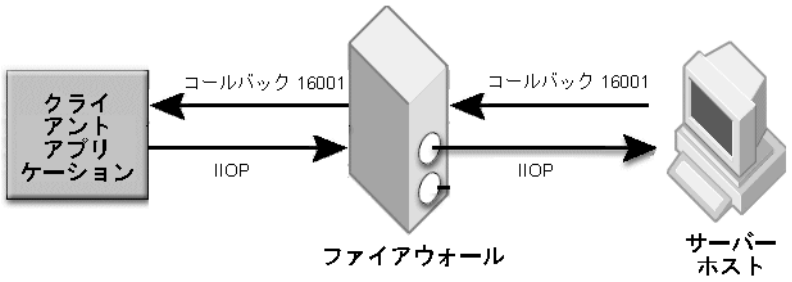
メモ　ファイアウォールコンポーネントを使用する方法では、次のように偽ホストと偽ポートを同じファイアウォールエントリに組み合わせた後でファイアウォールを指定してください。

```
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw
vbroker.firewall.fw.type=TCP
vbroker.firewall.fw.host=199.10.9.6
vbroker.firewall.fw.iiop_port=1683
vbroker.firewall.fw.hiop_port=0
```

メモ　NAT（ネットワークアドレス変換）とセキュリティで保護された接続を行うには、構成 1.3 のセキュリティプロパティの設定を使用してください。

構成例 1.7 : NAT なしのコールバック

転送通信設定については、構成 1.2 を参照してください。



クライアントのプロパティ：

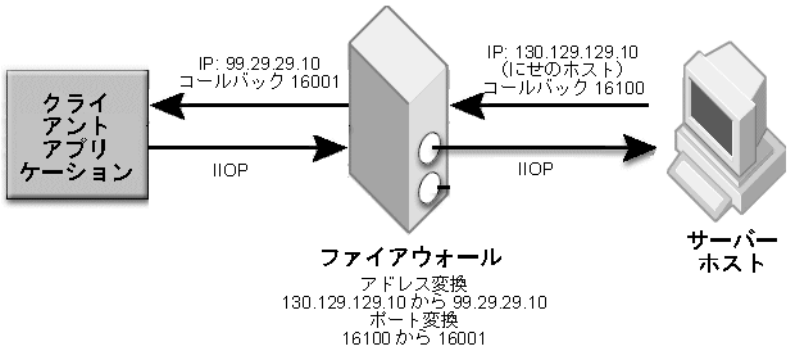
```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=16001
```

ファイアウォールの設定：

ポート 16001 で、サーバーホストからクライアントホストへの TCP パケットを有効にします。

構成例 1.8 : NAT ありコールバック

転送通信設定については、構成 1.2 を参照してください。



ファイアウォールの設定：

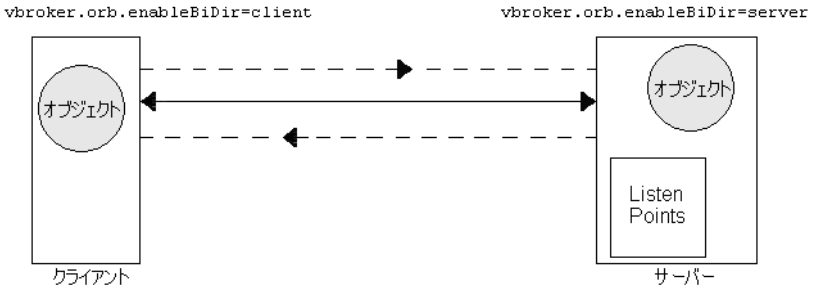
アドレス変換 : 130.129.129.10 → 99.29.29.10（サーバーネットワークからクライアントネットワークへのパケット）。ポート変換 : 16100 → 16001（サーバーネットワークからクライアントネットワークへのパケット）

クライアントのプロパティ：

```
vbroker.se.iiop_tp.host=99.29.29.10
vbroker.se.iiop_tp.proxyHost=130.129.129.10
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=16001
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=16100
```

構成例 1.9：双方向通信

双方向通信を有効にするには、構成 1.2、1.3、1.4、1.5、または 1.6 の設定に次の設定を追加します。



上の図では、前方向と後方向の両方の通信パスで同じ接続が使用されます。

クライアントのプロパティ：

```
vbroker.orb.enableBiDir=client
```

サーバーのプロパティ：

```
vbroker.orb.enableBiDir=server
```

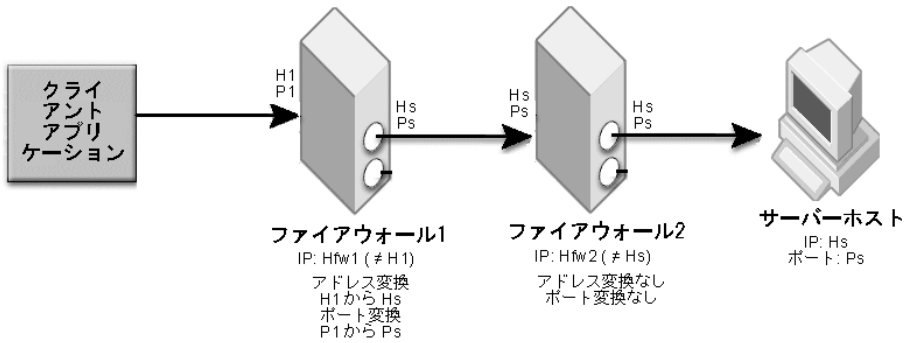
構成例 1.10：サーバー前面の複数のファイアウォール

この構成は、サーバーホストの前に 2 つのファイアウォールが存在する状況を示します。この構成は、3 つ以上のファイアウォールが存在する状況にも当てはまります。

両方のファイアウォールが NAT を実行しない

両方のファイアウォールが NAT を実行しない場合は、IIOP 通信に関してポート Ps で TCP パケットを許可するように両方のファイアウォールを設定します。

ファイアウォール 1 だけが NAT を実行する



ファイアウォール 1 が次の NAT を実行します。

- アドレス変換：H1 → Hs。ポート変換：P1 → Ps
- ファイアウォール 2 は、ポート Ps で TCP パケットを有効に設定する必要があります。
- クライアントは、ポート P1 でホスト H1 に IIOP パケットを送信します。

サーバーのプロパティ：次の 2 つの方法のどちらかを使用します。

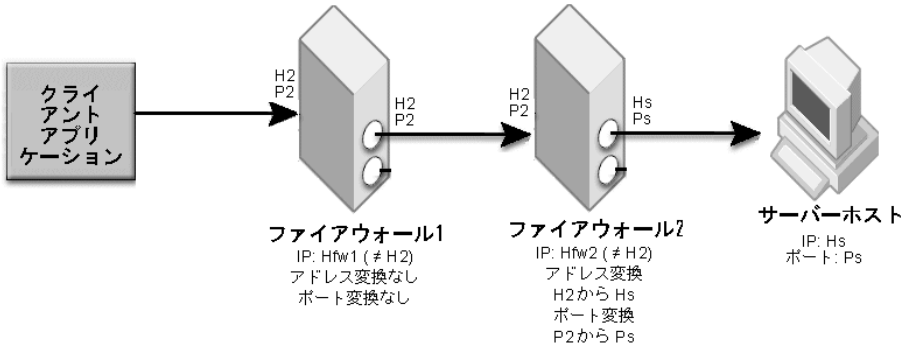
方法 1：IIOP プロファイルの使用

```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.proxyHost=<H1>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=<P1>
```

方法 2：ファイアウォールコンポーネントの使用

```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw1
vbroker.firewall.fw1.type=TCP
vbroker.firewall.fw1.host=<H1>
vbroker.firewall.fw1.iiop_port=<P1>
vbroker.firewall.fw1.hiop_port=0
```

ファイアウォール 2 だけが NAT を実行



ファイアウォール 2 が次の NAT を実行します。

- アドレス変換：H2 → Hs。Port 変換：P2 → Ps
- ファイアウォール 1 は、ポート P2 で TCP パケットを許可するように設定する必要があります。
- クライアントは、ポート P2 で、ホスト H2 に IIOP パケットを送信します。

サーバーのプロパティ：次の 2 つの方法のどちらかを使用します。

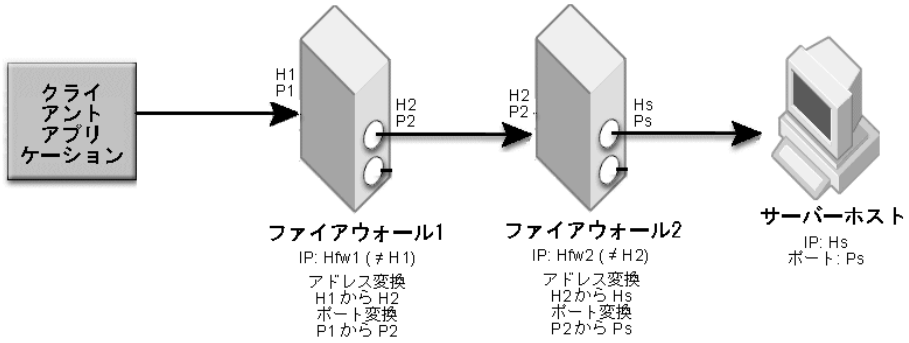
方法 1：IIOP プロファイルの使用

```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.proxyHost=<H2>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=<P2>
```

方法 2：ファイアウォールコンポーネントの使用

```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw2
vbroker.firewall.fw2.type=TCP
vbroker.firewall.fw2.host=<H2>
vbroker.firewall.fw2.iiop_port=<P2>
vbroker.firewall.fw2.hiop_port=0
```

両方のファイアウォールが NAT を実行



ファイアウォール 1 が次の NAT を実行します。

- アドレス変換：H1 → H2。ポート変換：P1 → P2
- ファイアウォール 2 が次の NAT を実行します。
- アドレス変換：H2 → Hs。Port 変換：P2 → Ps
- クライアントは、ポート P1 でホスト H1 に IIOP パケットを送信します。

サーバーのプロパティ：次の 2 つの方法のどちらかを使用します。

方法 1：IIOP プロファイルの使用

```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.proxyHost=<H1>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=<P1>
```

方法 2：ファイアウォールコンポーネントの使用

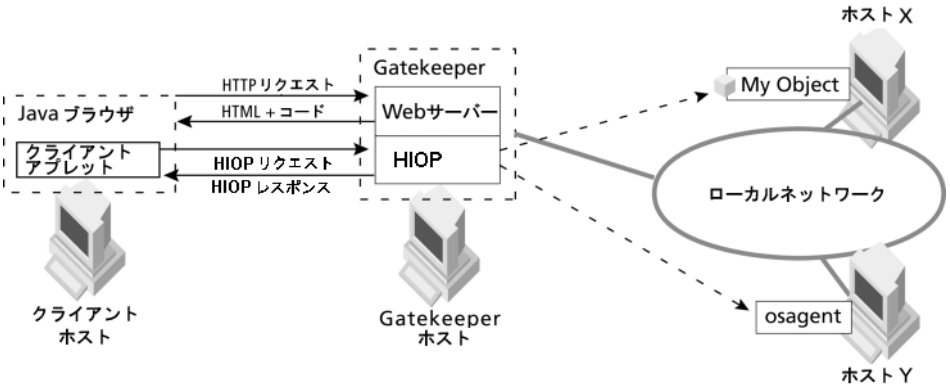
```
vbroker.se.iiop_tp.host=<Hs>
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<Ps>
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw1
vbroker.firewall.fw1.type=TCP
vbroker.firewall.fw1.host=<H1>
vbroker.firewall.fw1.iiop_port=<P1>
vbroker.firewall.fw1.hiop_port=0
```

メモ ファイアウォール 2 の NAT 情報を設定する必要はありません。`proxyHost` と `proxyPort` は、最初の NAT 偽ホストと偽ポートのみ指定します。ファイアウォールコンポーネントとファイアウォール経路については、最初の NAT デバイスだけを指定します。

Gatekeeper のデプロイメント

構成例 2.1 : Web サーバーとしての GateKeeper

GateKeeper は、HTML ページ、クライアントアプレット、および IOR ファイルを処理する Web サーバーとして機能できます。



次の GateKeeper のプロパティを使用して、GateKeeper の HTTP リスナーを設定します。

```
vbroker.se.exterior.scms=ex-iiop,ex-hiop
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
```

クライアントホストの Web ブラウザから、次の作業を行ってください。

- 次の URL を使って HTML ファイルまたはクライアントアプレットをロードする。

  ```
  http://gatekeeper:8088/ClientApplet.html
  ```

- 次の URL を使って Gatekeeper の IOR をロードする。

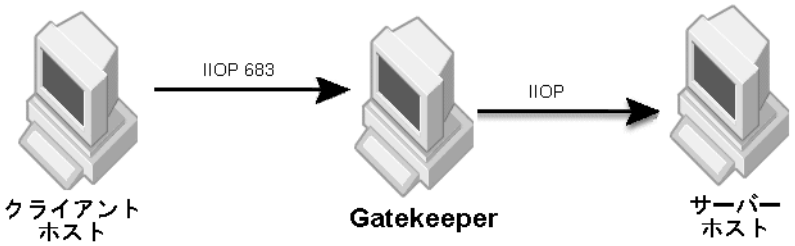
  ```
  http://gatekeeper:8088/gatekeeper.ior
  ```

次の例を使用して、クライアントアプレット（ClientApplet.html）を設定します。

```
<applet archive=vbjorb.jar code="ClientApplet.class" width="200" height="80">
<param name="org.omg.CORBA.ORBClass" value="com.inprise.vbroker.orb.ORB">
<param name="vbroker.orb.alwaysTunnel" value="true">
<param name="vbroker.orb.gatekeeper.ior" value="http://gatekeeper:8088/
gatekeeper.ior">
</applet>
```

その他必要なクライアントプロパティは、パラメータ名と値で同様に設定できます。

構成例 2.2 : IIOP プロキシとしての GateKeeper



クライアントのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.alwaysProxy=true
```

Gatekeeper のプロパティ：

```
vbroker.se.exterior.scm.ex-iiop.listener.port=683
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
```

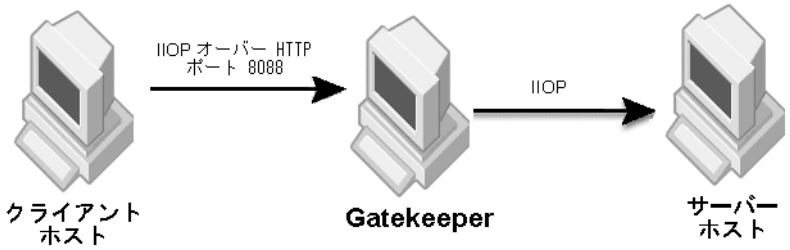
サーバーのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://gatekeeper:8088/gatekeeper.ior
```

クライアントが、HTTP トンネリングではなく IIOP の使用を求めるアプレットの場合、プロパティを設定せず次の設定を使用してください。

```
<param name="vbroker.orb.alwaysTunnel" value="true">:
<applet archive=vbjorb.jar code="ClientApplet.class" width="200" height="80">
<param name="org.omg.CORBA.ORBClass" value="com.inprise.vbroker.orb.ORB">
<param name="vbroker.orb.alwaysProxy" value="true">
<param name="vbroker.orb.gatekeeper.ior" value="http://gatekeeper:8088/
gatekeeper.ior">
</applet>
```

構成例 2.3：HTTP トンネリング接続



クライアントのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init,com.inprise.vbroker.H
IOP.Init
vbroker.orb.alwaysTunnel=true
Vbroker.orb.alwaysProxy=true
vbroker.orb.gatekeeper.ior=http://gatekeeper:8088/gatekeeper.ior
```

Gatekeeper のプロパティ：

```
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
vbroker.se.exterior.scm.ex-iiop.listener.port=683
```

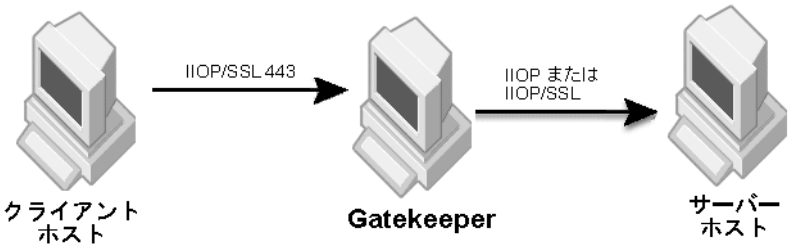
サーバーのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://gatekeeper:8088/gatekeeper.ior
vbroker.orb.exportFirewallPath=true
```

クライアントが HTTP トンネリングの使用を求めるアプレットの場合は、次の設定を使用してください。

```
<applet archive=vbjorb.jar code="ClientApplet.class" width="200" height="80">
<param name="org.omg.CORBA.ORBClass" value="com.inprise.vbroker.orb.ORB">
<param name="vbroker.orb.alwaysTunnel" value="true">
<param name="vbroker.orb.gatekeeper.ior" value="http://gatekeeper:8088/
gatekeeper.ior">
</applet>
```

構成例 2.4：セキュリティで保護された接続（SSL）



クライアントのプロパティ：

```
# ファイアウォール関連のプロパティ
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.alwaysProxy=true

# SSL 関連のプロパティを設定
vbroker.security.disable=false
vbroker.security.wallet.type=Directory:./identities
vbroker.security.wallet.identity= paul
vbroker.security.wallet.password= Paul$$$$
vbroker.security.trustpointsRepository=Directory:./trustpoints
```

Gatekeeper のプロパティ：

```
vbroker.se.exterior.scms=ex-iiop,ex-ssl
vbroker.se.exterior.scms.ex-iiop.listener.type=Disabled-IIOP
vbroker.se.exterior.scms.ex-hiop.listener.port=8088
vbroker.se.exterior.scms.ex-ssl.listener.port=443

# SSL 関連のプロパティを設定
vbroker.security.disable=false
vbroker.security.wallet.type=Directory:./identities
vbroker.security.wallet.identity= kevin
vbroker.security.wallet.password= Kevin$$$$
vbroker.security.trustpointsRepository=Directory:./trustpoints
vbroker.se.iiop_tp.scm.ssl.listener.port=<server SSL: listener port>
```

サーバーのプロパティ：

```
# ファイアウォール関連のプロパティ
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://gatekeeper:8088/gatekeeper.ior

# SSL 関連のプロパティを設定
vbroker.security.disable=false
vbroker.security.wallet.type=Directory:./identities
vbroker.security.wallet.identity= kevin
vbroker.security.wallet.password= Kevin$$$
vbroker.security.trustpointsRepository=Directory:./trustpoints

vbroker.se.iiop_tp.scms=iiop_tp,ssl
vbroker.se.iiop_tp.scm.ssl.listener.port=<server SSL listener port>
vbroker.se.iiop_tp.scm.iiop_tp.listener.type=Disabled-IIOP
```

メモ クライアント、サーバー、および Gatekeeper で、SSL セキュリティ保護サービスをロードしてください。

構成例 2.5：セキュリティで保護された HTTP トンネリング

構成 2.4 のクライアントおよびサーバー設定で、次をクライアントのプロパティに追加してください。

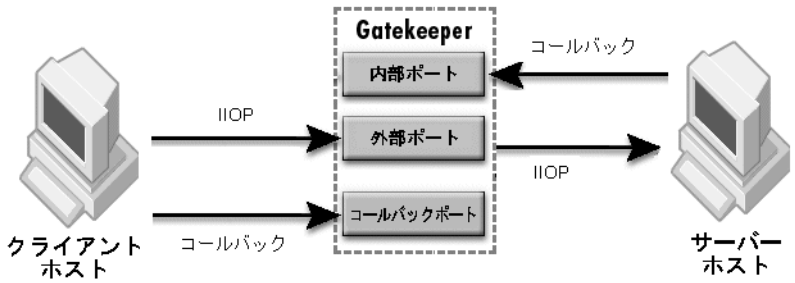
```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init,
                        com.inprise.vbroker.HIOP.Init
                        com.inprise.security.Init,
                        com.inprise.security.hiops.Init
vbroker.orb.alwaysTunnel=true
```

Gatekeeper のプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.security.Init,
                        com.inprise.vbroker.gatekeeper.ssl.Init,
                        com.inprise.security.hiops.Init
vbroker.se.exterior.scms=ex-IIOP,ex-hiop,ex-hiops
vbroker.se.exterior.scm.ex-iiop.listener.type=Disabled-IIOP
vbroker.se.exterior.scm.ex-hiops.listener.port=443
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
vbroker.security.wallet.type=Directory:./identities
vbroker.security.wallet.identity=Kevin
vbroker.security.wallet.password=Kevin$$$
vbroker.security.secureTransport=true
vbroker.security.trustpointsRepository=Directory:./trustpoints
vbroker.security.peerAuthenticationMode=none
```

構成例 2.6：コールバック接続（VisiBroker 3.x スタイル用）

転送通信設定については、構成 2.2 を参照してください。



次のクライアントのプロパティを設定します。

```
vbroker.se.iiop_tp.scm.iiop_tp.type=Callback-IIOP
vbroker.se.iiop_tp.scm.iiop_tp.listener.gatekeeper=http://gk_host:8088/
gatekeeper.ior
```

次の Gatekeeper プロパティで Gatekeeper のコールバック（VisiBroker 3.x スタイル）を有効にしてください。

```
vbroker.gatekeeper.callbackEnabled=true
vbroker.gatekeeper.backcompat.callback=true
vbroker.gatekeeper.backcompat.callback.listeners=iiop
vbroker.gatekeeper.backcompat.callback.listener.iiop.port=<exterior callback
port>
vbroker.gatekeeper.backcompat.callback.listener.iiop.type=IIOPCallback
```

内部ポート in-iiop は、コールバックを有効にすると自動的に有効になります。セキュリティで保護されたコールバックの場合にだけ、必要に応じて in-ssl、ex-ssl、および ex-hiops に対する SCM を追加する必要があります。

構成例 2.7：双方向通信

双方向通信を有効にするには、構成 2.2、2.3、2.4、または 2.5 の設定に次の設定を追加します。

クライアントのプロパティ：

```
vbroker.orb.enableBiDir=client
```

サーバーのプロパティ：

```
vbroker.orb.enableBiDir=server
```

Gatekeeper のプロパティ：

```
vbroker.orb.enableBiDir=both
```

構成例 2.8：パススルー接続

パススルー通信を有効にするには、構成 2.2 または 2.4 の設定に次の設定を追加します。

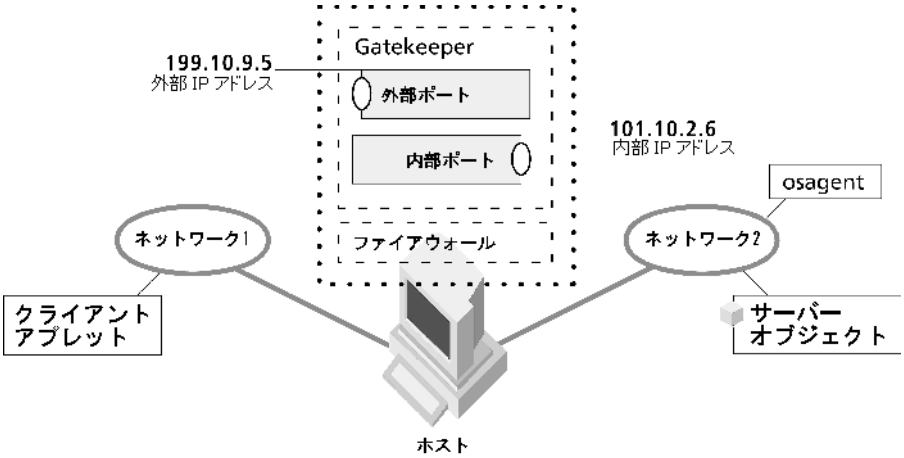
クライアントのプロパティ：

```
vbroker.orb.proxyPassthru=true
```

Gatekeeper のプロパティ：

```
vbroker.gatekeeper.enablePassthru=true
```

構成例 2.9：デュアルホームホスト設定の GateKeeper



Gatekeeper のプロパティを次のように設定します。

- 外部ホストおよび内部ホストアドレス

```
vbroker.se.exterior.host=199.10.9.5
vbroker.se.interior.host=101.10.2.6
```

- 外部リスナーポート

```
vbroker.se.exterior.scm.ex-iiop.listener.port=<exterior IIOP port>
vbroker.se.exterior.scm.ex-hiop.listener.port=<exterior HIOP port>
```

- 内部リスナーポート（VisiBroker 3.x スタイルのコールバックに使用）

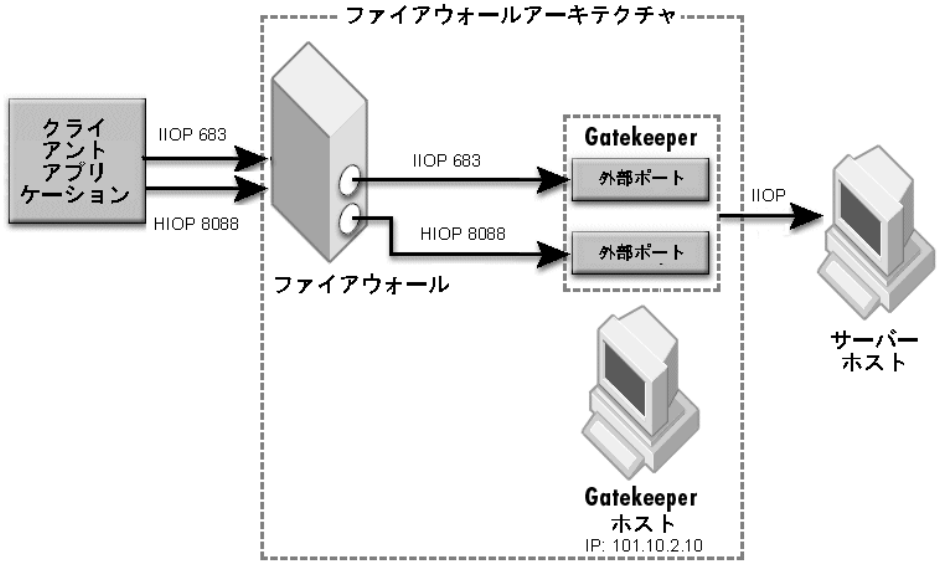
```
vbroker.se.interior.scm.in-iiop.listener.port=<interior IIOP port>
```

サーバー側にファイアウォールがある Gatekeeper

メモ　ルーターも、ファイアウォールの機能を実行できます。

Gatekeeper 前面のファイアウォール

構成例 3.1：ファイアウォールが NAT なしでパケットフィルタリングを実行



Gatekeeper のプロパティ：

```
vbroker.se.exterior.scm.ex-iiop.listener.port=683
vbroker.se.exterior.scm.ex-hiop.listener.port=8088
```

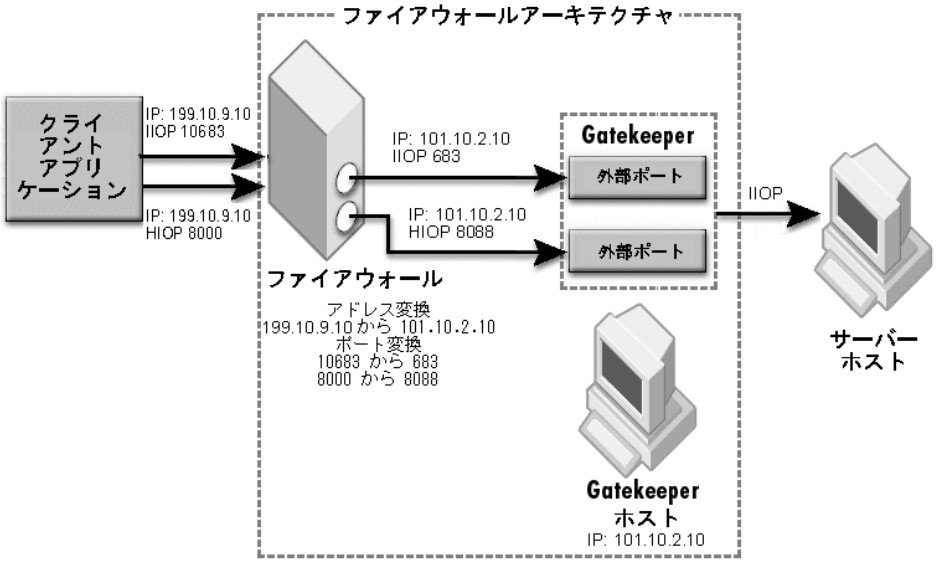
ファイアウォールの設定：

ポート 683 で TCP パケットの転送を有効にし、ポート 8088 で外部ネットワークから内部ネットワークへの HTTP パケットを有効にしてください。

サーバーのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://101.10.2.10:8088/gatekeeper.ior
```

構成例 3.2：ファイアウォールで NAT を実行



ファイアウォールの NAT 設定

アドレス変換：199.10.9.10 → 101.10.2.10。ポート変換：10683 → 683 および 8000 → 8088

Gatekeeper 前面のファイアウォールで NAT を指定する方法は 2 つあります。次のどちらかを使用します。

- GateKeeper の `proxyHost` と `proxyPort` の設定を使用する

  Gatekeeper のプロパティ：

  ```
  vbroker.se.exterior.host=101.10.2.10
  vbroker.se.exterior.proxyHost=199.10.9.10
  vbroker.se.exterior.scm.ex-iiop.listener.port=683
  vbroker.se.exterior.scm.ex-iiop.listener.proxyPort=10683
  vbroker.se.exterior.scm.ex-hiop.listener.port=8088
  vbroker.se.exterior.scm.ex-hiop.listener.proxyPort=8000
  ```

  サーバーのプロパティ：

  ```
  vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
  vbroker.se.iiop_tp.firewallPaths=p
  vbroker.firewall-path.p=gk
  vbroker.firewall.gk.type=PROXY
  vbroker.firewall.gk.ior=http://101.10.2.10:8088/gatekeeper.ior
  ```

- サーバーのファイアウォールコンポーネントを使用する

  Gatekeeper のプロパティ：

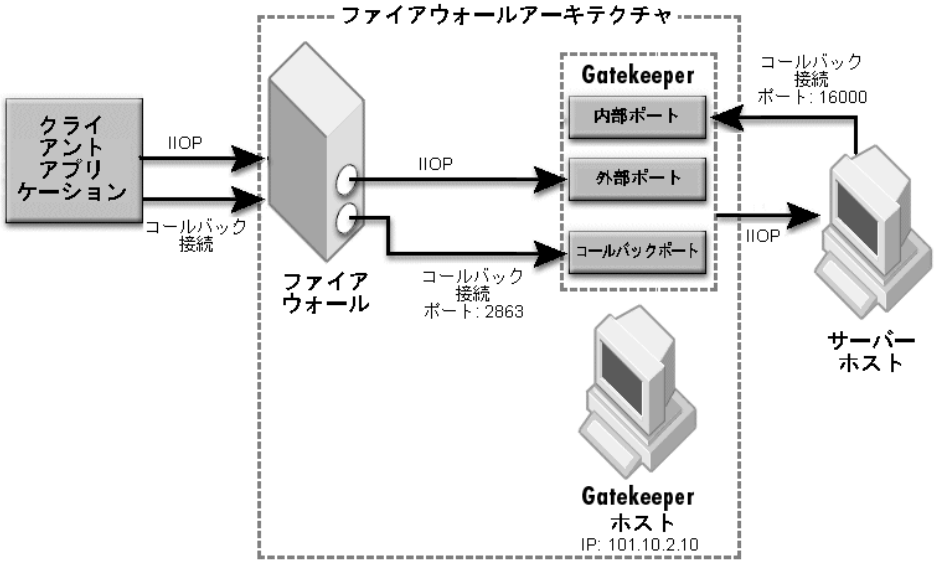
  ```
  vbroker.se.exterior.host=101.10.2.10
  vbroker.se.exterior.scm.ex-iiop.listener.port=683
  vbroker.se.exterior.scm.ex-hiop.listener.port=8088
  ```

サーバーのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=fw,gk
vbroker.firewall.fw.type=TCP
vbroker.firewall.fw.host=199.10.9.10
vbroker.firewall.fw.iiop_port=10683
vbroker.firewall.fw.hiop_port=8000
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://101.10.2.10:8088/gatekeeper.ior
```

構成例 3.3：コールバック接続（VisiBroker 3.x スタイル用）

転送通信設定については、構成 3.1 または 3.2 を参照してください。



次のクライアントのプロパティを設定します。

```
vbroker.se.iiop_tp.scm.iiop_tp.type=Callback-IIOP
vbroker.se.iiop_tp.scm.iiop.listener.gatekeeper=http://gk_host:8088/
gatekeeper.ior
```

次のプロパティで、Gatekeeper のコールバック（VisiBroker 3.x スタイル）を有効にし、コールバックポートを指定します。

```
vbroker.gatekeeper.callbackEnabled=true
vbroker.gatekeeper.backcompat.callback=true
vbroker.gatekeeper.backcompat.callback.host=101.10.2.10
vbroker.gatekeeper.backcompat.callback.listeners=iiop
vbroker.gatekeeper.backcompat.callback.listener.iiop.port=2683
vbroker.gatekeeper.backcompat.callback.listener.iiop.type=IIOPCallback
```

ファイアウォールの設定では、クライアントがポート 2683 経由で Gatekeeper へのコールバック接続（TCP プロトコル）を確立できるようにしておきます。

次の Gatekeeper のプロパティで、内部ポートを設定します。

```
vbroker.se.interior.scm.in-iiop.listener.port=16000
```

ファイアウォールが Gatekeeper のホストおよびコールバックポート（アドレス変換は 199.10.9.10 から 101.10.2.10、ポート変換は 12683 から 2863）で NAT を実行する場合は、次を Gatekeeper のプロパティに追加してください。

```
vbroker.gatekeeper.backcompat.callback.proxyHost=199.10.9.10
vbroker.gatekeeper.backcompat.callback.listener.iiop.proxyPort=12683
```

構成例 3.4：双方向通信

双方向通信を有効にするには、構成 3.1 または 3.2 の設定に次の設定を追加します。

クライアントのプロパティ：

```
vbroker.orb.enableBiDir=client
```

サーバーのプロパティ：

```
vbroker.orb.enableBiDir=server
```

Gatekeeper のプロパティ：

```
vbroker.orb.enableBiDir=both
```

構成例 3.5：パススルー接続

パススルー通信を有効にするには、構成 3.1 または 3.2 の設定に次の設定を追加します。

クライアントのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.proxyPassthru=true
```

Gatekeeper のプロパティ：

```
vbroker.gatekeeper.enablePassthru=true
vbroker.gatekeeper.passthru.inPortMin=<in_min_port>
vbroker.gatekeeper.passthru.inPortMax=<in_max_port>
```

ファイアウォールの設定：

注意 <in_min_port> から <in_max_port> の範囲のポートで、クライアントホストから GateKeeper までの TCP パケットのルーティングを有効にします。ファイアウォールは、このポート範囲でポート変換を実行することはできません。

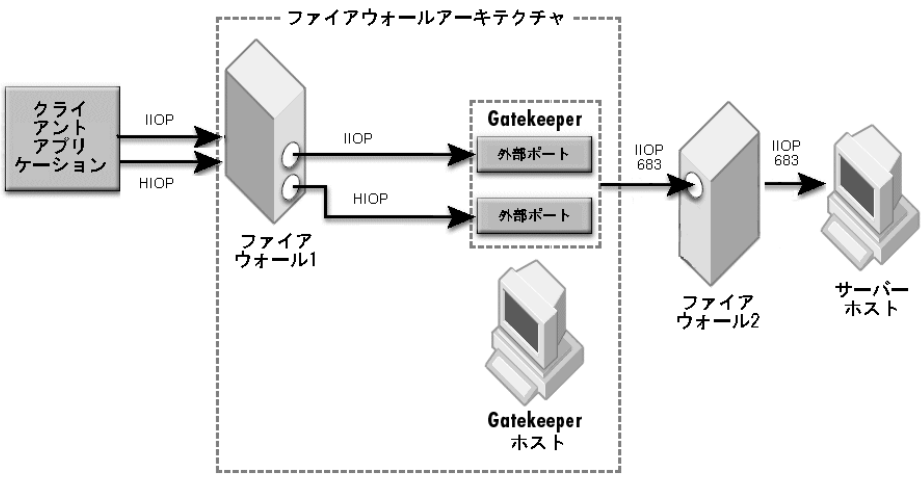
Gatekeeper の前後にファイアウォールが存在

サーバーは内部ネットワークに配置される一方で、Gatekeeper は DMZ（非武装地帯）に配置されます。

メモ GateKeeper の前に位置するファイアウォールに関する設定については、前の節を参照してください。ここでは、GateKeeper とサーバー間に位置するファイアウォールに関する設定について説明します。

構成例 4.1 : GateKeeper 背後のファイアウォールを設定

構成 3.1 または 3.2 の設定を使用して、クライアントと GateKeeper 間の通信を設定します。ここで説明した設定は、構成 3.1 または 3.2 と組み合わせて使用してください。



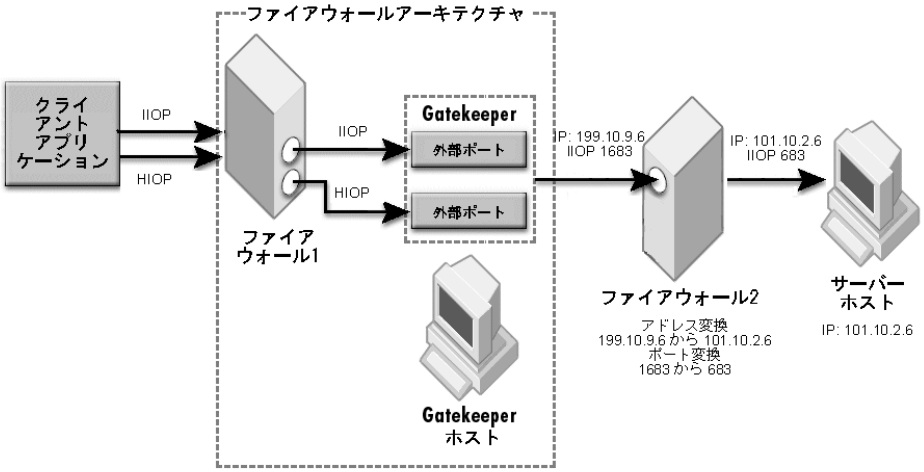
次のサーバーのプロパティを使用して、サーバーの IIOP のリスナーポートを指定します。

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
```

ポート 683 で GateKeeper からサーバーホストへの IIOP パケット (TCP プロトコル) が有効になるようにファイアウォール 2 を設定します。

構成例 4.2 : GateKeeper 背後のファイアウォールが NAT を実行

構成 3.1 または 3.2 の設定を使用して、クライアントと GateKeeper 間の通信を設定します。ここで説明した設定は、構成 3.1 または 3.2 と組み合わせて使用してください。



ファイアウォール 2 の NAT 設定

アドレス変換 : 199.10.9.10 → 101.10.2.10
ポート変換 : 1683 → 683

ファイアウォール 2 で NAT を指定する方法は 2 つあります。次のどちらかを使用します。

サーバーのプロパティ：

方法 1：IIOP プロファイルの使用

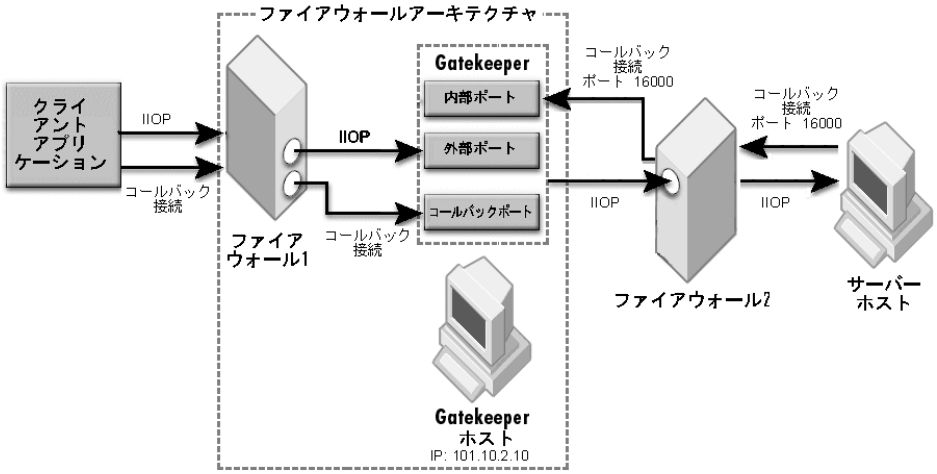
```
vbroker.se.iiop_tp.host=101.10.2.6
vbroker.se.iiop_tp.proxyHost=199.10.9.6
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=683
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=1683
```

方法 2：ファイアウォールコンポーネントの使用

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk,fw2
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://gatekeeper:8088/gatekeeper.ior
vbroker.firewall.fw2.type=TCP
vbroker.firewall.fw2.host=199.10.9.6
vbroker.firewall.fw2.iiop_port=1683
vbroker.firewall.fw2.hiop_port=0
```

構成例 4.3：コールバック接続（VisiBroker 3.x スタイル用）

構成 3.3 の設定を使用して、クライアントと Gatekeeper 間のコールバック接続を設定します。



次の Gatekeeper のプロパティで、内部ポートを設定します。

```
vbroker.se.interior.scm.in-iiop.listener.port=16000
```

ファイアウォール 2 では、ポート 16000 における TCP プロトコルによるサーバーと GateKeeper との通信を有効にしておく必要があります。

サーバーから GateKeeper に転送されるパケットに対して、ファイアウォール 2 が次の NAT を実行する場合があります。

アドレス変換：121.100.2.19 → 101.10.2.10。ポート変換：161000 → 16000

この場合は、Gatekeeper のプロパティに次のプロパティを追加してください。

```
vbroker.se.interior.host=101.10.2.10
vbroker.se.interior.proxyHost=121.100.2.19
vbroker.se.interior.scm.in-iiop.listener.port=16000
vbroker.se.interior.scm.in-iiop.listener.proxyPort=16100
```

構成例 4.4：双方向通信

双方向通信を有効にするには、構成 4.1 または 4.2 の設定に次の設定を追加します。

クライアントのプロパティ：

```
vbroker.orb.enableBiDir=client
```

サーバーのプロパティ：

```
vbroker.orb.enableBiDir=server
```

Gatekeeper のプロパティ：

```
vbroker.orb.enableBiDir=both
```

構成例 4.5：パススルー接続

パススルー通信を有効にするには、構成 4.1 または 4.2 の設定に次の設定を追加します。

クライアントのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.proxyPassthru=true
```

Gatekeeper のプロパティ：

```
vbroker.gatekeeper.enablePassthru=true
vbroker.gatekeeper.passthru.inPortMin=<in_min_port>
vbroker.gatekeeper.passthru.inPortMax=<in_max_port>
vbroker.gatekeeper.passthru.outPortMin=<out_min_port>
vbroker.gatekeeper.passthru.outPortMax=<out_max_port>
```

サーバーのプロパティ：

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<server IIOP port>
```

注意 <server IIOP port> の値は、<out_min_port> から <out_max_port> の範囲とします。

ファイアウォール 1 では、<in_min_port> から <in_max_port> の範囲のポートで、クライアントホストから GateKeeper までの TCP パケットのルーティングを有効にします。ファイアウォール 2 では、<out_min_port> から <out_max_port> の範囲のポートで、GateKeeper からサーバーホストまでの TCP パケットのルーティングを有効にします。ファイアウォールは、これらのポートでポート変換を実行しない設定とします。

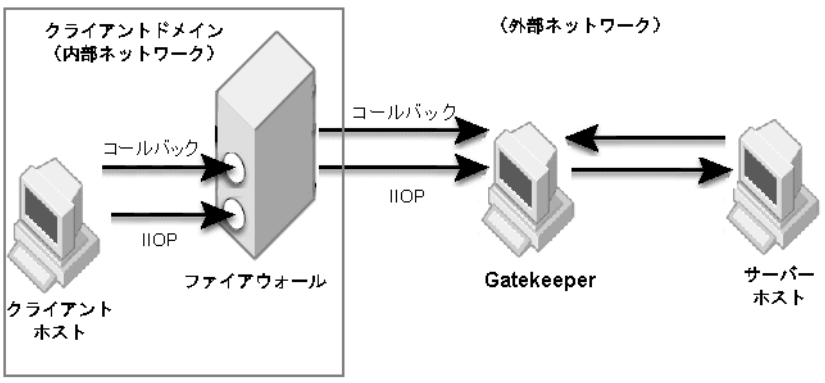
構成例 4.6：内部ネットワークスマートエージェント

Gatekeeper がクライアントであると想定し、構成 1.1 の設定を使用してください。

Gatekeeper とクライアント側のファイアウォール

構成例 5.1：ファイアウォールで IIOP が有効

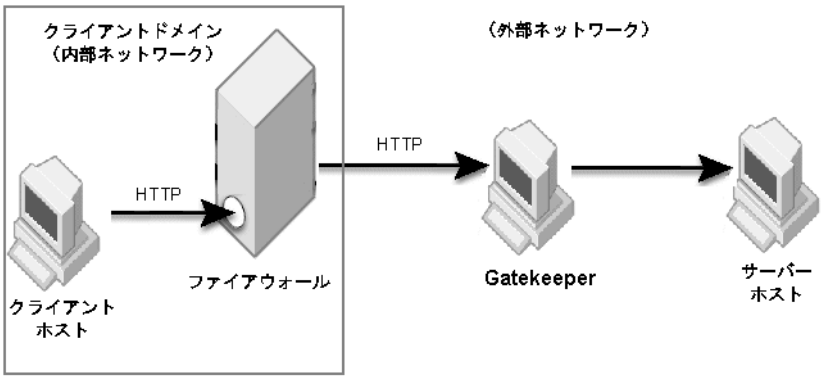
クライアント側のファイアウォールで、内部ネットワークのクライアントから外部ネットワークへの IIOP メッセージ（TCP プロトコル）送信を有効にします。



構成 3.x を参照して、Gatekeeper の前に位置するサーバー側ファイアウォールをクライアント側ファイアウォールと交換してください。Gatekeeper はクライアントのドメイン外にあるので、通常、ファイアウォールを管理するクライアント側の管理者には、Gatekeeper の設定を変更する権限がありません。管理者は、Gatekeeper のリスナーポート情報を収集し、それに応じてクライアント側のファイアウォールを設定する必要があります。

構成例 5.2：ファイアウォールで HTTP のみ有効

クライアント側のファイアウォールでは、内部ネットワークのクライアントから外部ネットワーク向けに HTTP メッセージの送信だけを有効にします。IIOP メッセージは、ファイアウォールによってブロックされます。そのため、クライアントがクライアント側ファイアウォール外の Gatekeeper と通信するには、HTTP トンネリングを使用する必要があります。



クライアントに常に HTTP トンネリングを使用させるには、次のクライアントのプロパティを設定します。

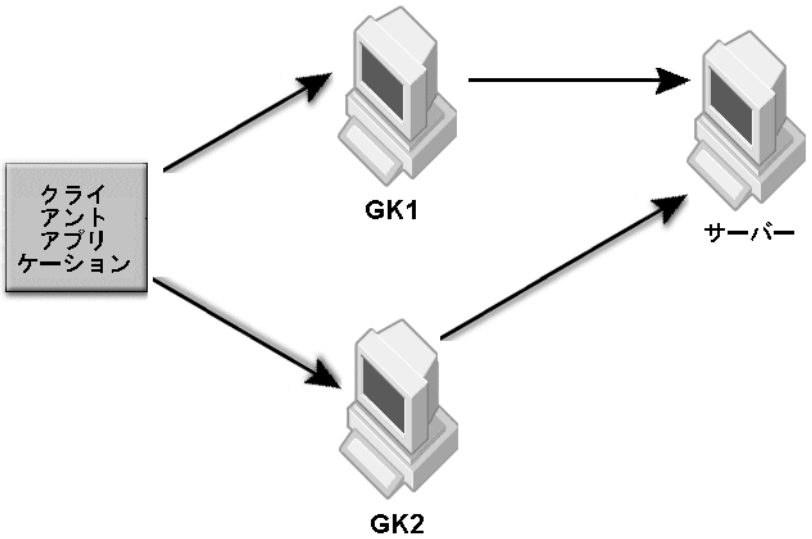
```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.alwaysTunnel=true
```

メモ　HTTP トンネリングは、VisiBroker 3.x スタイルのコールバックをサポートしていません。コールバックが必要な場合は、双方向接続を使用してください。また、HTTP トンネリングを使用する場合は、パススルー接続は使用できません。

Gatekeeper の負荷分散とフォールトトレランス

構成例 6.1：フォールトトレランスのためのマルチ GateKeeper 使用

1 つの GateKeeper だけに依存するかわりに、複数の GateKeeper をフォールトトレランスに配置することができます。冗長性を持たせるには、複数の Gatekeeper をサーバーに割り当ててください。



この例におけるサーバーのプロパティは次のとおりです。

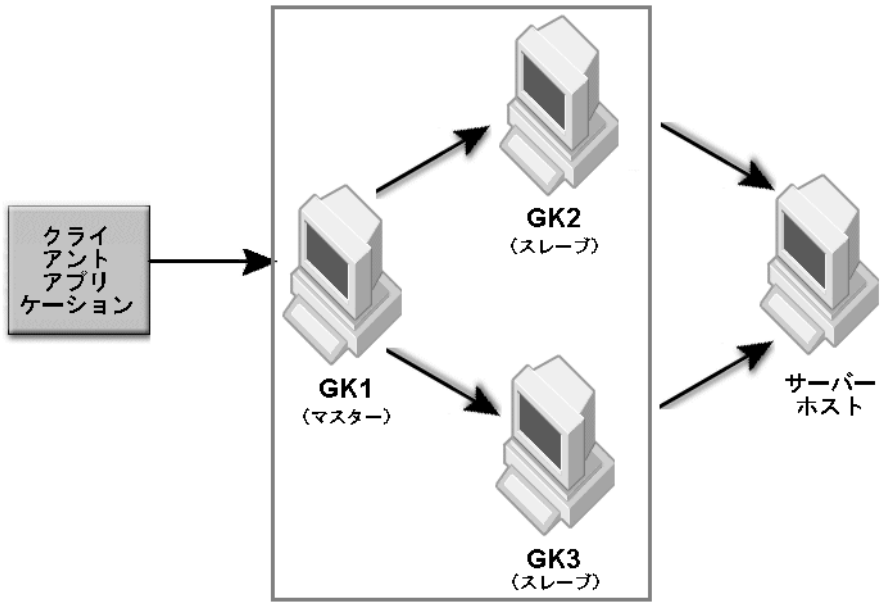
```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p1,p2
vbroker.firewall-path.p1=gk1
vbroker.firewall.gk1.type=PROXY
vbroker.firewall.gk1.ior=http://gk1_host:8088/gatekeeper.ior
vbroker.firewall-path.p2=gk2
vbroker.firewall.gk2.type=PROXY
vbroker.firewall.gk2.ior=http://gk2_host:8088/gatekeeper.ior
```

GK1 と GK2 の両方で次のプロパティが必要です。

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.gatekeeper.ext.Init
```

クライアントは、GK1 と GK2 のどちらかを使ってサーバーと通信できます。一方の Gatekeeper が停止しても、クライアントはもう一方の Gatekeeper を使ってサーバーと通信できます。

構成例 6.2：負荷分散のためのマスター／スレーブ設定



上の図は、GK1 をマスター GateKeeper とし、GK2 および GK3 をスレーブ GateKeeper とする場合のマスター／スレーブ GateKeeper 設定を示します。

GateKeeper GK1 のプロパティ（マスター）：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.gatekeeper.ext.Init
vbroker.gatekeeper.load.slaves=gk2,gk3
vbroker.gatekeeper.load.slave.gk2=http://gk2_host:8088/gatekeeper.ior
vbroker.gatekeeper.load.slave.gk3=http://gk3_host:8088/gatekeeper.ior
```

このほかにプロパティをスレーブ GateKeeper GK1 と GK2 に追加する必要はありません。

サーバーのプロパティ（マスター GateKeeper だけを指定）：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk1
vbroker.firewall.gk1.type=PROXY
vbroker.firewall.gk1.ior=http://gk1_host:8088/gatekeeper.ior
```

クライアントがサーバーの IOR を直接取得できない場合は、クライアントは次のプロパティを使って通信する GateKeeper を指定できます。

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.gatekeeper.ior=http://gk1_host:8088/gatekeeper.ior
```

この設定は、フォールトトレランスも提供できます。各接続クライアントに対して、マスター GateKeeper は次のスレーブ GateKeeper を順にクライアントの処理に割り当てますが、そのスレーブ GateKeeper がダウンした場合、クライアントはマスター GateKeeper に戻り、次の利用可能なスレーブ GateKeeper の割り当てを受けます。これは、クライアントが使用できる GateKeeper を取得するまで続けられます。

マスター GateKeeper は、接続クライアントに割り当てることができる利用可能な GateKeeper のリストを実際に保持しています。このリストには、すべてのスレーブ GateKeepers とマスター GateKeeper 自体が含まれています。したがって、順番が来れば、マスター GateKeeper は自分自身をクライアントに割り当てます。

マスター GateKeeper で次のプロパティが設定されている場合は、マスター GateKeeper がリストに含まれません。

```
vbroker.gatekeeper.load.balancer=master
```

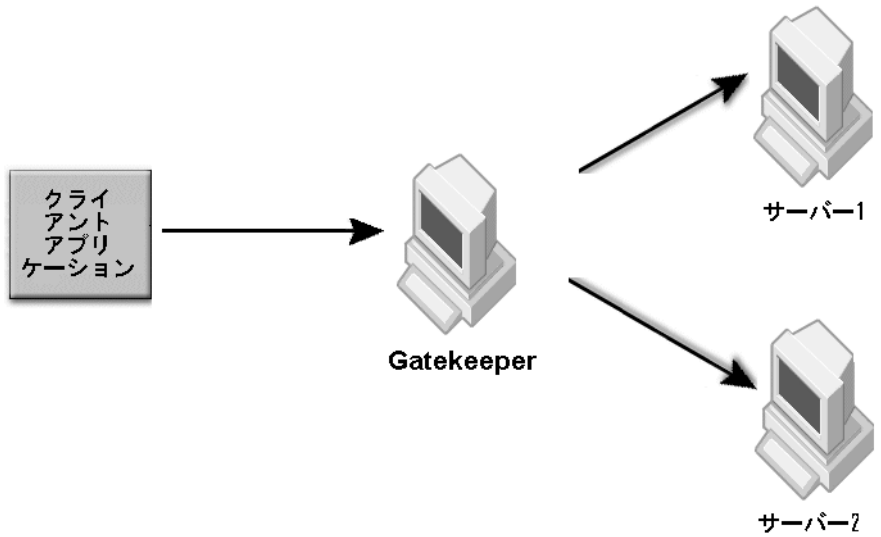
すべてのスレーブ GateKeeper がダウンしている場合に、使用可能な GateKeeper を取得しようとクライアントが無限にマスター GateKeeper に戻り続けないように、クライアント側で次のプロパティを設定する必要があります。

```
vbroker.orb.rebindForward=N
```

この *N* は、スレーブ GateKeeper の数より小さくする必要があります。

マスター GateKeeper 自体がダウンしている場合は、クライアント ORB のリバインドメカニズムが、利用可能な最初のスレーブ GateKeeper を介してすべてのクライアント接続を作成します。この状態では負荷分散は行われません。負荷分散機能はマスター GateKeeper で行われ、そのマスターがダウンしているからです。ただし、クライアントは引き続き動作しており、サーバーに接続されているため、フォールトトレランスは維持されています。

構成例 6.3：負荷分散のために同じサーバー上にマルチインスタンス



サーバーの負荷分散およびフォールトトレランスを行うために、同一のサーバーのインスタンスを複数配置することができます。負荷分散に関しては、Gatekeeper がラウンドロビン方式を使って複数のサーバーに要求を送信します。フォールトトレランスに関しては、1 つのサーバーが停止しても、別のサーバーが同じサービスを引き続き提供できます。

Gatekeeper に次のプロパティを追加してください。

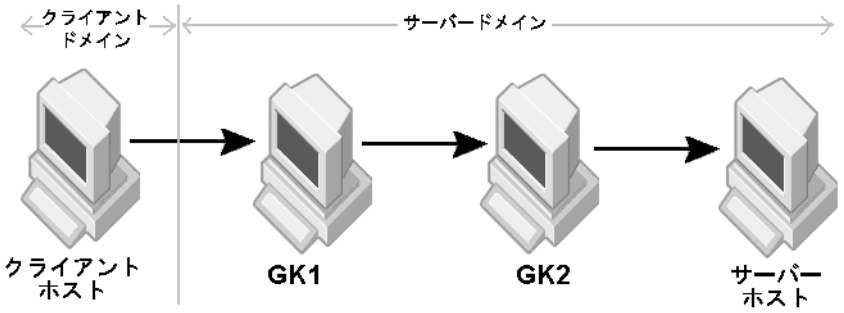
```
vbroker.orb.dynamicLibs=com.inprise.vbroker.gatekeeper.ext.Init
```

サーバー 1 およびサーバー 2 のプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk
vbroker.firewall.gk.type=PROXY
vbroker.firewall.gk.ior=http://gk_host:8088/gatekeeper.ior
```

Gatekeeper のチェイン化

構成例 7.1：サーバー側のチェイン化

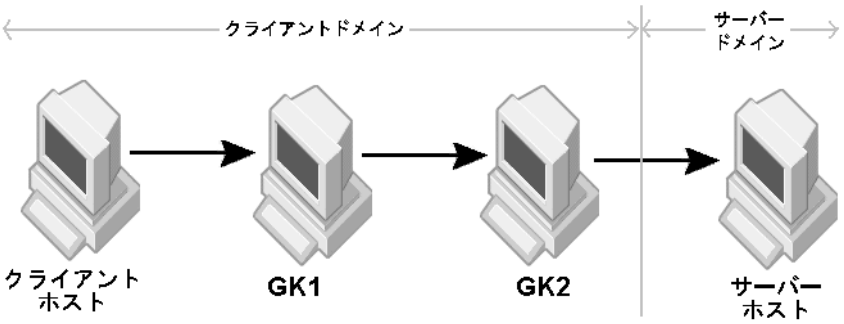


サーバー側 Gatekeeper のチェイン化を指定するには、次のサーバーのプロパティを使用してください。

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk1,gk2
vbroker.firewall.gk1.type=PROXY
vbroker.firewall.gk1.ior=http://gk1_host:8088/gatekeeper.ior
vbroker.firewall.gk2.type=PROXY
vbroker.firewall.gk2.ior=http://gk2_host:8088/gatekeeper.ior
```

クライアントがサーバーの IOR を取得すると、Gatekeeper のチェイン化を実行してサーバーと通信できるようになります。

構成例 7.2：クライアント側のチェイン化



クライアントのプロパティ：

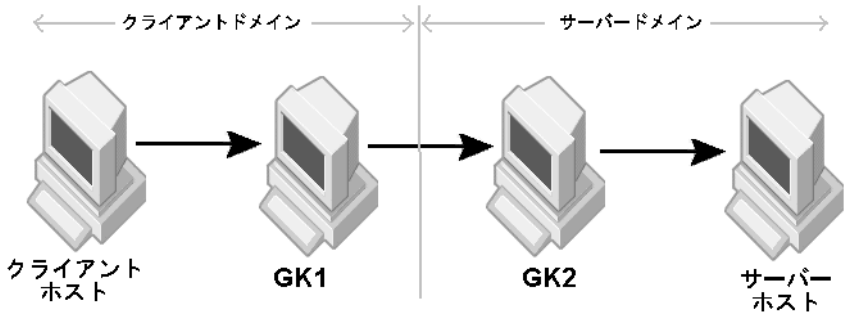
```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.gatekeeper.ior=http://GK1:8088/gatekeeper.ior
```

GK1 のプロパティ：

```
vbroker.orb.gatekeeper.ior=http://GK2:8088/gatekeeper.ior
```

メモ チェイン化された Gatekeeper でクライアントがサーバーと通信するには、チェインの最後尾にある Gatekeeper（GK2）でサーバーの IOR を取得する必要があります。

構成例 7.3：サーバー側とクライアント側両方のチェイン化



クライアントのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.gatekeeper.ior=http://gk_host:8088/gatekeeper.ior
```

サーバーのプロパティ：

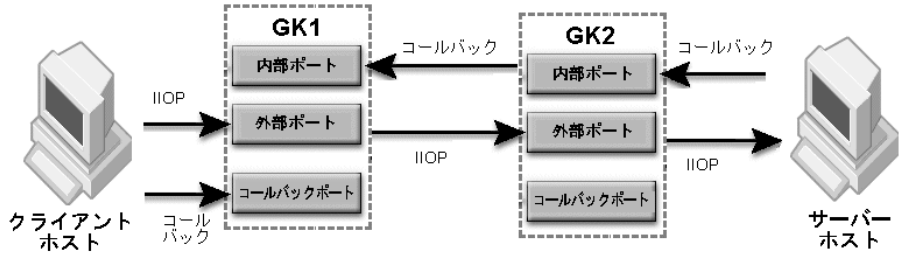
```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.p=gk2
vbroker.firewall.gk2.type=PROXY
vbroker.firewall.gk2.ior=http://gk2_host:8088/gatekeeper.ior
```

GK1 が常時 GK2 と接続している場合は、次の GK1 のプロパティを使用して、GK1 を GK2 と静的にチェイン化できます。

```
vbroker.orb.gatekeeper.ior=http://gk2_host:8088/gatekeeper.ior
```

それ以外の場合は、GK1 が、スマートエージェントまたはネーミングサービスで、サーバーまたは GK2 の IOR を取得する必要があります。

構成例 7.4：コールバック通信（VisiBroker 3.x スタイル）



VisiBroker 3.x スタイルのコールバック通信を有効にするには、次のプロパティを設定します。

クライアントのプロパティ：

```
vbroker.orb.alwaysProxy=true
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.se.iiop_tp.scm.iiop_tp.listener.type=Callback-IIOP
vbroker.se.iiop_tp.scm.iiop_tp.listener.gatekeeper=http://gk1_host:8088/
gatekeeper.ior
```

GK1 および GK2 のプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.gatekeeper.ext.Init
vbroker.gatekeeper.callbackEnabled=true
vbroker.gatekeeper.backcompat.callback=true
vbroker.gatekeeper.backcompat.callback.listeners=iiop
vbroker.gatekeeper.backcompat.callback.listener.iiop.type=IIOPCallback
vbroker.gatekeeper.backcompat.callback.listener.iiop.port=
<exterior callback port>
vbroker.gatekeeper.backcompat.callback.host=<GK exterior IP address>
vbroker.se.interior.scm.in-iiop.listener.port=<interior port>
```

サーバーのプロパティ：

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<IIOP listener port>
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.firewallPaths=p
vbroker.firewall-path.intranet=gk1,gk2
vbroker.firewall.gk1.type=PROXY
vbroker.firewall.gk1.ior=http://gk1_host:8088/gatekeeper.ior
vbroker.firewall.gk2.type=PROXY
vbroker.firewall.gk2.ior=http://gk2_host:8088/gatekeeper.ior
```

構成例 7.5：双方向接続

構成 7.1、7.2、または 7.3 を参照してください。

双方向通信を有効にするには、次の設定を追加します。

クライアントのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.enableBiDir=client
```

サーバーのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.enableBiDir=server
```

GK1 および GK2 のプロパティ：

```
vbroker.orb.enableBiDir=both
```

構成例 7.6：パススルー接続

構成 7.1、7.2 または、7.3 の図を参照してください。

クライアントのプロパティ：

```
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init
vbroker.orb.proxyPassthru=true
```

GK1 および GK2 のプロパティ：

```
vbroker.gatekeeper.enablePassthru=true
vbroker.gatekeeper.passthru.inPortMin=<in_min_port>
vbroker.gatekeeper.passthru.inPortMax=<in_max_port>
vbroker.gatekeeper.passthru.outPortMin=<out_min_port>
vbroker.gatekeeper.passthru.outPortMax=<out_max_port>
```

サーバーのプロパティ：

```
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=<server IIOP port>
```

メモ <server IIOP port> の値は、GK2 の <out_min_port> から <out_max_port> の範囲とします。GK2 の <in_min_port> から <in_max_port> の範囲は、<out_min_port> から <out_max_port> of GK1 の範囲にします。

いずれかのホスト間にファイアウォールが存在する場合は、次の表を参照して、開く必要があるポートを確認してください。

ファイアウォールの場所	開いているポートの範囲
クライアントと GK1 の間	GK1 外部 IIOP および HIOP リスナーポート、GK1 <in_min_port> と <in_max_port>
GK1 と GK2 の間	GK2 外部 IIOP および HIOP リスナーポート、GK2 <in_min_port> と <in_max_port>
GK2 とサーバーの間	GK2 <out_min_port> と <out_max_port>

注意 ファイアウォールは、これらのパススルーポートでポート変換を実行することはできません。

複数のファイアウォール／サブネット環境での VisiBroker の使用

VisiBroker は、複数のファイアウォールが存在する構成で使用できます。一般に VisiBroker では、ファイアウォールを通過する方法が 2 つ提供されています。

第一に、次のプロパティを使用して、ネットワークアドレス変換（TCP ファイアウォール）を設定できます。

```
vbroker.se.iiop_tp.host=www.realdomain.com
vbroker.se.iiop_tp.proxyHost=www.fakedomain.com
vbroker.se.iiop_tp.scm.iiop_tp.listener.port=25000
vbroker.se.iiop_tp.scm.iiop_tp.listener.proxyPort=32000
```

この設定では、実ホスト／ポート情報は IOR で失われます。つまり、IOR で利用できるのは偽ホスト／ポートだけです。また、別の TCP ファイアウォール設定も、サーバー側の設定として一般によくデプロイメントされています。この設定は ORB の組み込みメカニズムなので、すべての種類のサービス（GateKeeper、ネーミングサービスなど）に適用されます。

```
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.firewallPaths =Queen
vbroker.firewall-path.Queen=Atlantic
vbroker.firewall-path.Atlantic.type=TCP
vbroker.firewall-path.Atlantic.host=www.fakedomain.com
vbroker.firewall-path.Atlantic.iiop_port=32000
vbroker.firewall-path.Atlantic.hiop_port=32003
vbroker.firewall-path.Atlantic.ssl_port=32004
```

この設定の利点は、設定情報が失われないことです。内部クライアントは、実 IP ホスト／ポート情報を使ってサーバーに直接接続できます。ただし、この設定には、生成される IOR ファイルで IP ホスト／ポート（実と偽の両方）が公開されるというリスクがあります。

第二に、GateKeeper は、ファイアウォールサーバー上で GIOP プロキシサーバーとして機能できます。GateKeeper では、さまざまな目的で設計されたさまざまなメカニズムを利用できます。次に例を示します。

- **通常モード**。ファイアウォールが GIOP プロキシサーバー（GateKeeper など）に対して少なくとも 1 つのポートを許可する場合に使用されます。必要に応じて HTTP トンネリングに切り替えることができる自動モードです。
- **パススルーモード**。ファイアウォールが一定範囲のポートと、クライアントとサーバーの間で交換される GateKeeper によって解釈されないパケットを許可する場合に使用されます。この場合、GateKeeper は、リソースマネージャとしてのみ機能します。GateKeeper がリソースマネージャとして機能するのは、クライアントが使用する IP ポートを割り当てるためです。
- **HTTP トンネリング**。ファイアウォールが HTTP トラフィックだけを許可する場合に使用されます。この場合、GIOP プロキシサーバーは、ファイアウォール内で実行できません。かわりに、GateKeeper の前面に HTTP プロキシサーバーが存在します。クライアント側 ORB には、GIOP メッセージを HTTP メッセージに変換するためのメカニズムが組み込まれており、変換後の HTTP メッセージが HTTP プロキシサーバーま

たはファイアウォールに送信されます。HTTP プロキシサーバー（または適用可能なファイアウォール）は、HTTP メッセージを GateKeeper に転送します。さらに、GateKeeper は、HTTP メッセージを GIOP メッセージに変換し、それを要求された送信先（サーバーや別の GateKeeper など）に転送します。この設定は、ファイアウォールが発信 HTTP トラフィックは許可するが、ほかの種類の TCP 接続は許可しない場合のクライアント側設定でも使用できます。

メモ 複数のファイアウォールでは、上記の設定方法を組み合わせる必要があります。基本的に、複数のファイアウォールの使用方法はデプロイメントによって異なるため、考えられるすべての組み合わせについてここで説明することはできません。一般的なガイドラインを次に示します。

- スマートエージェントは単一ドメインで動作することを目的に設計されているため、使用しないでください。
- CORBA オブジェクト IOR の格納とルックアップには、CORBA ネーミングサービスを使用してください。
- ドメインネームサービス（DNS）のルックアップは使用しないでください。
- ネットワークアドレス変換（NAT）または TCP ファイアウォール設定は、一連のファイアウォールの最も外側にあるファイアウォールでのみ使用してください。その場合は、内部クライアントであっても、ファイアウォールの外にあるかのように動作することが必要です。
- GateKeeper は、ファイアウォール環境内の実行可能な場所であれば、どこでも使用できます。HTTP トンネリング機能は、TCP 接続がファイアウォールによって許可されない場合に使用できます。
- GateKeeper のチェイン化は、複数のファイアウォールが必要な場合に使用できます。GateKeeper のチェイン化では、複数のホップを設定できます。
- 負荷分散のために複数の GateKeeper を使用してください。

ファイアウォールとスマートエージェント構成

ファイアウォールアーキテクチャ内のスマートエージェントは、ファイアウォールホスト上で動作しません。かわりに、内部ネットワーク内でスマートエージェントを実行できます。通常、スマートエージェントは、セキュリティ上の理由から、外部ネットワークに公開されません。スマートエージェントは、IPv4 UDP ブロードキャストメッセージを使って自分自身を通知します。ファイアウォール／ルーターは、ブロードキャストメッセージがネットワーク内の次のホップに転送されることをブロックできるため、スマートエージェントは、通常、ローカルネットワーク内でのみ可視になります。外部ネットワークからスマートエージェントにアクセスする必要がある場合は、ファイアウォール上の特定のポートを開く必要があります。

スマートエージェントは、次の環境変数を使用します。

- `OSAGENT_PORT`
- `OSAGENT_CLIENT_HANDLER_PORT`

VisiBroker ORB がスマートエージェントを使って CORBA オブジェクトを登録および照会するには、`OSAGENT_PORT` 環境変数が設定されている必要があります。`OSAGENT_PORT` プロパティのデフォルト値は 14000 です。`OSAGENT_PORT` を適切な TCP/IP ポートに設定することで、仮想ドメインを定義できます。指定されたサブネット内でスマートエージェントをいくつでも実行できます。複数の `OSAGENT_PORT` 値を設定すると、複数のドメインが作成されます。つまり、あるスマートエージェントドメインに登録された CORBA オブジェクトは、別のドメインから照会を行う CORBA クライアントには不可視になります。

このスマートエージェントに到達するには、CORBA アプリケーションで次の TCP/IP アドレスまたはポートが使用されるように設定します。

- `vbroker.agent.addr=143.186.142.21`

- vbroker.agent.port=25873

クライアント、サーバー、または GateKeeper がスマートエージェントを使用する必要がない場合は、それぞれのプロパティファイルで次のプロパティを設定してスマートエージェントを無効にします。

```
vbroker.agent.enableLocator=false
```

GateKeeper はマルチホーム（またはファイアウォール）ホスト上で実行します。スマートエージェントは、マルチホームホストまたは内部ネットワークのいずれかで実行できます。GateKeeper が特定のスマートエージェントを使用するように設定するには、たとえば、次のプロパティを使用します。

```
vbroker.agent.addr=143.186.142.21
vbroker.agent.port=25873
```

クライアントプログラムが GateKeeper に（たとえば、次のプロパティを使って）サーバーオブジェクトを照会して IOR を取得するように要求する場合、内部ネットワーク内のサーバーは、自分自身を GateKeeper と同じスマートエージェントに登録する必要があります。

```
vbroker.agent.addr=143.186.142.21
vbroker.agent.port=25873
```

GateKeeper が一度に使用できるスマートエージェントドメインは 1 つだけです。スマートエージェントドメインは、OSAGENT_PORT 値または vbroker.agent.port プロパティを設定することによって決定されます。GateKeeper を介してアクセスできるすべてのサーバーは、同じスマートエージェントドメインまたはネーミングサービスに登録する必要があります。内部ネットワーク内の GateKeeper と同じサブネットでスマートエージェントを実行することをお勧めします。

スマートエージェントでは、次のポートが必要です。

1. OSAGENT_PORT（UDP 型）
2. OSAGENT_CLIENT_HANDLER_PORT（UDP 型）
3. OSAGENT_CLIENT_HANDLER_PORT（TCP 型）

ORB アプリケーションが使用する OSAGENT_PORT は UDP ポートです。ロケーションサービスに割り当てられるのは、スマートエージェントが使用する TCP 型ポート（OSAGENT_CLIENT_HANDLER_PORT）だけです。UDP 型の OSAGENT_CLIENT_HANDLER_PORT は、スマートエージェント自身が使用します。OSAGENT_CLIENT_HANDLER_PORT は、スマートエージェントが実行されているホストに対してのみ設定してください。

ファイアウォール構成でのスマートエージェントの使用

スマートエージェントには、いくつかのフェイルオーバー機能と負荷分散機能が組み込まれています。スマートエージェントのドメインは、使用されている OSAGENT_PORT によって定義されます。ある ORB アプリケーション（サーバーなど）が特定のドメイン（スマートエージェントドメイン）内のいずれかのスマートエージェントに登録されている場合、ほかの ORB アプリケーション（クライアントプログラムなど）は、同じドメイン内にある任意のスマートエージェントからそのドメイン内のサーバーオブジェクトを照会できます。スマートエージェントは、ドメイン内でサーバーオブジェクトを検索しますが、クライアントアプリケーションはこの処理を認識しません。1 つのスマートエージェントでエラーが発生した場合、ORB アプリケーションは、同じドメイン内の別のスマートエージェントを見つけ、自分自身を再登録して処理を続行できます。

各ファイアウォールには固有の動作があるため、スマートエージェントの負荷分散機能は、ファイアウォールにまで及ぶようには設計されていません。たとえば、NAT（ネットワークアドレス変換）デバイスは、IP アドレス／ポートを変更する一種のファイアウォールです。スマートエージェントは、NAT に対応するようには設計されていません。また、ファイアウォールには、特定の種類のパケットだけを許可するもの、セキュリティと暗号化を要求するもの、DNS ルックアップを許可しないものなどあります。したがって、どのよう

な種類のファイアウォールまたは NAT 設定でも、スマートエージェントは使用しないでください。

スマートエージェントはファイアウォール構成内で使用するようには設計されていませんが、アプリケーションがファイアウォールの背後にあるスマートエージェントにアクセスする必要がある場合は、次の手順にしたがいます。ただし、この手順は、部署間のファイアウォールにだけ適用してください。

1 ファイアウォールで OSAGENT_PORT と OSAGENT_CLIENT_HANDLER_PORT を開きます。一部のファイアウォールでは、パケットがスマートエージェントに到達するように、静的転送経路を設定する必要があります。アプリケーション間にあるすべてのファイアウォールで、これらのポートを開く必要があります。ファイアウォールがマルチホームホスト上にある場合のために、localaddr（たとえば、<instal_dir>var/defaults/adm/properties/services/osagentfile フォルダにある）を編集し、OSAGENT_LOCAL_ADDR_FILE を設定して、スマートエージェントが要求パケットを監視するためにバインドする必要があるすべてのインターフェイスを指定します。

2 agentaddr ファイルでスマートエージェント IP アドレスを設定して、1 つのネットワーク上のスマートエージェントが別のネットワーク上のスマートエージェントと通信できるようにします。

3 ORB アプリケーションが起動されるすべてのホストで、OSAGENT_PORT と OSAGENT_CLIENT_HANDLER_PORT を設定します。これらのポートは、ファイアウォールで開いているポートと同じにする必要があります。

メモ 上記の設定でスマートエージェントを使用することは可能ですが、スマートエージェントをこのように使用することは、お勧めしません。このような設定は、一部のファイアウォールでは機能しますが、すべてのファイアウォールで機能するとは限りません。

ファイアウォール構成でのスマートエージェントのエラー時の動作

スマートエージェントでエラーが発生すると、ORB アプリケーションは、同じサブネット内の別のスマートエージェントに切り替える必要があります。スマートエージェントの OSAGENT_PORT はすでに固定されているため、ORB アプリケーションは、UDP ブロードキャストを送信して別のスマートエージェントを検索します。ファイアウォールがある場合、ORB アプリケーションは、スマートエージェントが実行されている別のホストに到達できる必要があります。ORB アプリケーションは、かわりのスマートエージェントの場所を知らない場合があるため、実行できる処理は限られます。スマートエージェントが同じホストで再起動した場合、クライアントはそのスマートエージェントに接続できます。基本的に、スマートエージェントが UDP ブロードキャストベースの技術を使用することの理解が重要です。ファイアウォールやルーターが UDP ブロードキャストを転送しないことがあるため、これがファイアウォールにまたがってスマートエージェントを使用できない原因の 1 つになります。ただし、スマートエージェントでエラーが発生しでも、同じサブネット内のスマートエージェントは使用できます。

スマートエージェントを使用するクライアントの動作

次のプロパティを設定すると、特定の範囲のポートを使ってスマートエージェントにバインドするようにクライアント ORB アプリケーションを設定できます。

```
vbroker.agent.clientPort
vbroker.agent.clientPortRange
```

ポート範囲を指定すると、クライアント ORB は、指定された範囲のローカルポートだけを使用します。Windows/NT では実際にポートを閉じる動作が遅延し、ポート範囲の使用が制限されるため、クライアントのポート範囲の指定が必要です。

GateKeeper とほかの CORBA サービスの使用

クライアントから見ると、GateKeeper は、ほかのすべての CORBA サービスに対して透過的です。通常のサーバーオブジェクトとほかの CORBA サービス（ネーミングサービス、トランザクションサービス、通知サービス、イベントサービスなど）の間に違いはありません。

サーバー側の設定では、クライアント ORB によって識別され、必要な場合にだけ使用される IOR 内のファイアウォールコンポーネントを指定するようにサーバーを設定できます。この場合、クライアントは、直接の接続を確立できなかった場合にだけフェイルオーバーして、GateKeeper によってサーバーにバインドします。サーバーによって使用されるデフォルトのサーバーエンジンが iiop_tp だとすると、ファイアウォール設定の典型的なプロパティは次のようになります。

```
vbroker.orb.exportFirewallPath=true
vbroker.se.iiop_tp.firewallPaths =Queen,King
vbroker.firewall-path.Queen=Atlantic,Pacific
vbroker.firewall-path.King=Indian
vbroker.firewall-path.Atlantic.type=TCP
vbroker.firewall-path.Atlantic.host=www.borland.com
vbroker.firewall-path.Atlantic.iiop_port=25000
vbroker.firewall-path.Atlantic.hiop_port=25003
vbroker.firewall-path.Atlantic.ssl_port=25004
vbroker.firewall-path.Pacific.type=PROXY
vbroker.firewall-path.Pacific.ior=http://www.mygk1domain.com/gatekeeper.ior
vbroker.firewall-path.Indian.type=PROXY
vbroker.firewall-path.Indian.ior=http://www.mygk1domain.com/gatekeeper.ior
```

クライアント側の設定では、GateKeeper IOR をクライアント ORB に提供できます。この場合、クライアントは、すべてのオペレーションを GateKeeper を使って実装します。この場合は、次のプロパティを使用できます。

```
vbroker.orb.alwaysProxy=true
vbroker.orb.gatekeeper.ior=http://www.mydomain.com/gatekeeper.ior
```

クライアントがスマートエージェントにアクセスできない場合に、GateKeeper を使ってサーバーオブジェクトを検索することもできます。その場合は、GateKeeper を介してロケーションサービスを使用することをお勧めします。それには、GateKeeper で次のプロパティを使用します。

```
vbroker.gatekeeper.locationService=true
```

また、オブジェクトを検索するために、クライアント側で次のプロパティを使用します。

```
vbroker.locator.ior=http://www.mydomain.com/gatekeeper.ior
```

HTTP プロキシサーバーを使用した GateKeeper の設定

クライアントと GateKeeper の間で HTTP プロキシサーバーが実行されている場合、GateKeeper は、HTTP プロキシサーバーの IP ホスト／ポートアドレスを IOR で公開する必要があります。それには、次に説明する方法にしたがいます。次の GateKeeper プロパティを設定します。この設定は、ネットワークアドレス変換の設定に似ています。この場合、HTTP プロキシサーバーは NAT として機能します。

```
vbroker.se.exterior.proxyHost=142.186.142.21
vbroker.se.exterior.scm.ex-hiop.listener.proxyPort=32001
```

メモ 上の 2 つのプロパティの設定は、必須ではありません。この設定では、GateKeeper が NAT デバイスの背後にあります。したがって、HTTP トンネリングを使って GateKeeper と通信しようとするすべてのクライアントは、常に HTTP プロキシサーバーを介して要求を渡します。

GateKeeper へのサーバーエンジンの追加

GateKeeper では、次の 3 つの組み込みサーバーエンジンを利用できます。

- iiop_tp
- 外部
- 内部

iiop_tp サーバーエンジンは、管理目的でのみ使用されます。外部サーバーエンジンと内部サーバーエンジンは、それぞれ外部ネットワークと内部ネットワークのために使用されます。TCP/IP ネットワークを使用する場合、各サーバーエンジンは、ネットワーク IP ホストアドレスに関連付けられます。次に例を示します。

```
vbroker.se.exterior.host=142.186.142.21
vbroker.se.interior.host=142.186.182.30
vbroker.se.iiop_tp.host=192.73.8.25
```

このバージョンの GateKeeper では、プロパティファイルを使って新しいサーバーエンジンを追加することはできません。

GateKeeper へのリスナーまたはサーバー接続マネージャの追加

GateKeeper には、特定の種類のサービスに対して複数のサーバー接続マネージャ (SCM) またはリスナーを設定できます。通常、SCM は、IIOP、SSL、HIOP、HIOPS などの特定の種類のサービスを提供します。各 SCM は、外部サーバーエンジンや内部サーバーエンジンなどのサーバーエンジンにバインドされます。SCM を設定するには、論理名（たとえば、myscm）を割り当て、この名前を次のプロパティに追加する必要があります。

```
vbroker.se.exterior.scms=ex-iiop,ex-hiop,myscm
```

さらに、各 SCM に対して次のプロパティを追加する必要があります。詳細については、付録 A を参照してください。

```
vbroker.se.exterior.scm.myscm.manager.type=Socket
vbroker.se.exterior.scm.myscm.manager.connectionMax=0
vbroker.se.exterior.scm.myscm.manager.connectionMaxIdle=0
vbroker.se.exterior.scm.myscm.listener.type=IIOP
vbroker.se.exterior.scm.myscm.listener.port=683
vbroker.se.exterior.scm.myscm.listener.proxyPort=0
vbroker.se.exterior.scm.myscm.listener.giopVersion=1.2
vbroker.se.exterior.scm.myscm.dispatcher.type=ThreadPool
vbroker.se.exterior.scm.myscm.dispatcher.threadMax=100
vbroker.se.exterior.scm.myscm.dispatcher.threadMin=0
vbroker.se.exterior.scm.myscm.dispatcher.threadMaxIdle=300
```

GateKeeper のストレス／負荷メトリック

GateKeeper は Java ベースの ORB サービスインプリメンテーションなので、さまざまな Java ツールを使ってパフォーマンス特性を取得できます。

VisiBroker コンソールは、任意の ORB サービス（GateKeeper を含む）に関するリアルタイムのパフォーマンス特性を提供します。割り当てメモリ、スレッド数、接続、フラグメンテーションなどに関する情報を表示できます。

サーブレットとしての GateKeeper のデプロイメント

ここでは、GateKeeper をサーブレットとして Tomcat 5.0 Web サーバーにデプロイメントする例について説明します。これ以外のバージョンの Tomcat を使用している場合は、多少の変更が必要です。

この例では、用意されている Client.properties とともに bank_agent サンプルを使用します。Client.properties の主な目的は、Web サーバーに埋め込まれた GateKeeper サーブレットを介してのみサーバーに接続するようにクライアントに指示することです。bank_agent サンプルは、次のディレクトリにあります。

```
<install_dir>/examples/vbroker/basic/bank_agent
```

このシナリオでサンプルを実行するために、ほかに次のファイルが必要です。

- **web.xml** - サーブレットとしてデプロイメントされる GateKeeper のデプロイメントデスクリプタ。
- **Client.properties** - Web サーバー内にサーブレットとして埋め込まれた GateKeeper を介して bank_agent クライアントが Bank サーバーに接続するように設定するためのプロパティ。

最後に、web.xml と Client.properties のダンプ画面があります（「web.xml」と「Client.properties」を参照）。これを指定されたディレクトリの指定されたファイルにコピー、貼り付け、および保存できます。

サンプルのビルド

1. http://jakarta.apache.org/tomcat/index.html から無料の Tomcat Web サーバーをダウンロードし、次の手順にしたがってインストールします。Web ブラウザを起動して http://localhost:8080 を参照すると、正常にインストールされたかどうかを確認できます。
2. web.xml（「web.xml」を参照）を <Tomcat root install>/webapps/gatekeeper_servlet/WEB-INF/web.xml にコピーして貼り付け、保存します。必要に応じてサブディレクトリを作成します。
3. 次のファイルを開き、編集します。

   ```
   <Tomcat root install>/webapps/gatekeeper_servlet/WEB-INF/web.xml
   ```

   osagent Tomcat ポートを正しく参照するように、次の部分を変更します。

   ```
   <init-param>
       <param-name>vbroker.agent.port</param-name>
       <param-value>YOUR OSAGENT PORT</param-value>
   </init-param>
    ...
   <init-param>
       <param-name>
           vbroker.se.exterior.scm.ex-hiop.listener.port
       </param-name>
       <param-value>
           TOMCAT HTTP PORT. OUT OF TOMCAT BOX, THIS MUST BE 8080
       </param-value>
   </init-param>
   ```

4. 次の場所にある jar を

   ```
   <install_dir>/lib/
   ```

   次の場所にコピーします。

   ```
   <Tomcat install root dir>/shared/lib
   ```

Tomcat の shared/lib/ ディレクトリに jar を配置すると、コンテナ内に配置されたすべての Web アプリケーションからこれらの jar を利用できるようになります。これでは不都合がある場合は、Tomcat のマニュアルを参照して、ほかの lib ディレクトリに配置してください。

- lm.jar
- sanctuary.jar
- vbjorb.jar
- sanct4.jar
- vbjclientorb.jar
- vbsec.jar

5 Client.properties（「Client.properties」を参照）を次のディレクトリにコピーして貼り付け、保存します。

```
<install_dir>/examples/vbroker/basic/bank_agent
```

そのファイルを開き、次の設定を編集します。

```
vbroker.orb.gatekeeper.ior=http://<host>:<port>/gatekeeper_servlet/
gatekeeper.ior
```

この <host> は Tomcat が実行されているコンピュータの IP、<port> は Tomcat が監視している HTTP ポートです。これは、上の web.xml 内のポートと同じ番号です。デフォルトで Tomcat をインストールした場合は、8080 になります。

サンプルの実行

1 既存の VisiBroker に適切な環境を設定します。UNIX プラットフォームでは、${VBROKERDIR}/vbroker.sh を実行します。

2 必要に応じて、basic/bank_agent サンプルをビルドします。

3 osagent が実行されていることを確認します。

4 ${JAVA_HOME} と ${PATH} が同じ適切な JDK を参照していることを確認します。

5 次のコマンドを実行して、Tomcat を起動します。

Windows :

```
<Tomcat root install>/bin/startup.bat
```

UNIX :

```
<Tomcat root install>/bin/startup.sh
```

6 サンプルの basic/bank_agent ディレクトリに移動します。

```
<install_dir>/examples/vbroker/basic/bank_agent
```

7 次のコマンドを実行して、Bank サーバーを起動します。

```
vbj Server
```

8 次のコマンドを実行して、クライアントを起動します。

```
vbj -DORBpropStorage=Client.properties Client
```

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//
EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
```

```
  <display-name>GateKeeper Servlet</display-name>

  <description> サーブレットとしての GateKeeper のサンプル </description>

  <servlet>

    <servlet-name>GateKeeperServlet</servlet-name>

    <servlet-class>
      com.inprise.vbroker.gatekeeper.servlet.Servlet
    </servlet-class>

    <load-on-startup />

    <init-param>
      <param-name>
        vbroker.se.exterior.scm.ex-hiop.listener.path
      </param-name>
      <param-value>
        /gatekeeper_servlet/servlet
      </param-value>
    </init-param>

    <init-param>
      <param-name>vbroker.agent.port</param-name>
      <param-value>PUT YOUR OSAGENT PORT</param-value>
    </init-param>

<!-- Some setups may not allow UDP broadcast to locate osagent
     In that case, uncomment and set the following correctly
    <init-param>
      <param-name>vbroker.agent.address</param-name>
      <param-value>
        PUT IP OF THE MACHINE, ON WHICH OSAGENT IS RUNNING
      </param-value>
    </init-param>
-->

    <init-param>
      <param-name>vbroker.gatekeeper.referenceStore</param-name>
      <param-value>
        webapps/gatekeeper_servlet/gatekeeper.ior
      </param-value>
    </init-param>

    <init-param>
      <param-name>vbroker.se.exterior.scms</param-name>
      <param-value>ex-iiop,ex-hiop</param-value>
    </init-param>


<!-- If you want Visibroker log messages, uncomment this.
     Log messages will go to the specified file below, relative
     to Tomcat root install dir

    <init-param>
      <param-name>vbroker.orb.debug</param-name>
      <param-value>true</param-value>
    </init-param>

    <init-param>
      <param-name>vbroker.orb.logLevel</param-name>
      <param-value>7</param-value>
    </init-param>

    <init-param>
```

```
      <param-name>vbroker.orb.warn</param-name>
      <param-value>2</param-value>
    </init-param>

    <init-param>
      <param-name>vbroker.orb.logger.output</param-name>
      <param-value>webapps/gatekeeper_servlet/log.txt</param-value>
    </init-param>
-->

    <init-param>
      <param-name>
        vbroker.se.exterior.scm.ex-iiop.listener.type
      </param-name>
      <param-value>Disabled-IIOP</param-value>
    </init-param>

    <init-param>
      <param-name>
        vbroker.se.exterior.scm.ex-hiop.listener.port
      </param-name>
      <param-value>8080</param-value>
    </init-param>

    <init-param>
      <param-name>
        vbroker.se.exterior.scm.ex-iiop.listener.port
      </param-name>
      <param-value>0</param-value>
    </init-param>

  </servlet>

  <servlet-mapping>
    <servlet-name>GateKeeperServlet</servlet-name>
    <url-pattern>/servlet</url-pattern>
  </servlet-mapping>

</web-app>
```

Client.properties

```
# 次は 1 行です。
vbroker.orb.dynamicLibs=com.inprise.vbroker.firewall.Init,com.inprise.vbroker.H
IOP.Init

vbroker.orb.alwaysTunnel=true
vbroker.orb.alwaysProxy=true

# 次は 1 行です。
vbroker.orb.gatekeeper.ior=http://host:8080/gatekeeper_servlet/gatekeeper.ior

# デバッグメッセージを使用する場合は、以下の行のコメントを解除します。
# vbroker.orb.debug=true
# vbroker.orb.warn=2
# vbroker.orb.logLevel=7
```

索引

H

I

J

N

O

P

R

S

T

V

W

あ

い

え

お

か

き

く

こ

さ

す

せ

そ

ち

つ

て

と

な

ね

は

ひ

ふ

ほ

ま

め

よ

ら

り

る

ろ