18

通信とメッセージ

概要

アプリケーションの各部分の通信インフラストラクチャを設計する方法は、アプリケーション (特に分散アプリケーション) の設計に影響する主な要素の 1 つです。たとえば、コンポーネントは相互に通信し、ユーザーによる入力をビジネス レイヤーに送信してから、データ レイヤーを通じてデータ ストアを更新する必要があります。コンポーネントが同じ物理ティアに配置されると、多くの場合、コンポーネント間の直接的な通信をあてにできます。ただし、多くのシナリオで行われるように、コンポーネントとレイヤーを物理的に別のサーバーとクライアント コンピューターに配置する場合、このようなレイヤーのコンポーネント間で、効率的かつ正確に通信する方法を検討する必要があります。

一般的には、直接的な通信 (コンポーネント間でメソッドを呼び出すなど) かメッセージ ベースの通信のどちらかを使用します。メッセージ ベースの通信では、コンポーネントを分離できるなど、多くのメリットがあります。コンポーネントを分離すると、保守容易性が向上するだけでなく、柔軟性も提供されるので、後から展開の戦略を簡単に変更できるようになります。ただし、メッセージ ベースの通信では、パフォーマンス、信頼性、セキュリティなど、考慮しなければならない問題もあります。

この章では、適切な通信手法を選択し、選択した手法で最高の結果を得るための方法を理解して、発生する可能性があるセキュリティと信頼性に関する問題の予測に役立つ設計ガイドラインを示します。ただし、この章の大部分では、適切なメッセージ ベースの通信メカニズムの設計と、非同期通信および同期通信、データ形式、パフォーマンス、セキュリティ、相互運用性、および実装テクノロジの選択に関するガイドラインに重点を置いて説明します。

一般的な設計ガイドライン

アプリケーションの通信方針を設計する際には、レイヤー間およびティア間の通信によるパフォーマンスへの影響を考慮します。論理上または物理上の境界を越える通信によって処理のオーバーヘッドが増加するので、ラウンド トリップの回数を減らし、ネットワーク経由で送信されるデータの量を最小限に抑える効率的な通信を設計します。通信方針を決定する際には、次のガイドラインを考慮します。

- **境界を越える場合は通信方針を考慮する**: 各境界について理解し、これらの境界が通信のパフォーマンスにどのように影響するかを理解します。たとえば、コンピューターのプロセス、コンピューター、マネージ コードからアンマネージ コードの呼び出しはすべて、アプリケーションのコンポーネントや、外部サービスおよび外部アプリケーションと通信する際に越える可能性がある境界です。
- **プロセス境界を越える場合はメッセージ ベースの通信の使用を検討する**: Windows Communication Foundation (WCF) を TCP または名前付きパイプ プロトコルと使用して、パフォーマンスを最大限に高めます。
- **物理的な境界を越える場合はメッセージ ベースの通信の使用を検討する**: 物理的な境界を越えてリモート コンピューターと通信する場合は、WCF の使用を検討します。信頼できる 1 回限りのメッセージの配信には、Microsoft Message Queuing の使用を検討します。
- **リモート レイヤーにアクセスする場合はパフォーマンスと応答性を最大限に高める**: リモート レイヤーと通信する場合は、粒度の粗いメッセージ ベースの通信方法を使用することで通信要件を減らし、できる限り非同期通信を使用して UI のブロックやフリーズを回避します。
- **境界を越えるデータ形式のシリアル化を考慮する**: 他のシステムとの相互運用性を確保する必要がある場合は、XML シリアル化の使用を検討します。XML シリアル化では、オーバーヘッドが増加する点に留意する必要があります。パフォーマンスが重要な場合は、バイナリ シリアル化の使用を検討します (バイナリ シリアル化は XML シリアル化よりも高速で、シリアル化されたデータのサイズが XML よりも小さくなります)。
- **通信中にメッセージや機密データを保護する**: 暗号化、デジタル証明書、およびチャネル セキュリティ機能を使用することを検討します。
- **べき等性と交換性を強化するメカニズムを実装する**: アプリケーション コードで、複数回到着するメッセージ (べき等性) と、バラバラの順序で到着する複数のメッセージ (交換性) を検出して管理できるようにします。

メッセージ ベースの通信のガイドライン

メッセージ ベースの通信では、クライアントが XML ベースのメッセージをトランスポート チャネル経由で渡すことによって呼び出されるサービス インターフェイスを定義することにより、呼び出し元にサービスを公開できます。通常、メッセージ ベースの呼び出しは、リモート クライアントから実行されますが、メッセージ ベースのサービス インターフェイスでは、ローカルの呼び出し元も同じようにサポートできます。メッセージ ベースの通信は、次のシナリオに適しています。

- 中長期的な投資が必要なビジネス システムを実装する場合 (たとえば、長期にわたってパートナーに公開され、パートナーが使用するサービスを構築する場合)。
- 高可用性を提供する必要があるか、信頼されないネットワーク経由で操作する必要がある大規模なシステムを実装する場合。この場合は、メッセージのストア アンド フォワードのメカニズムによって信頼性を向上できます。
- 構築しているサービスで使用する他のサービスから当該サービスを分離したり、当該サービスを使用する他のサービスから当該サービスを分離する必要がある場合。インターフェイスの詳細をクライアントに通知するメッセージ ベースのサービス インターフェイスを使用すると、個々のクライアントで具体的な実装を必要とすることなく、すべてのクライアントがサービスを簡単に使用できます。
- 非同期モデルを使用する実際のビジネス プロセスを使用する場合。

メッセージ ベースの通信を使用する際には、次のガイドラインを考慮します。

- 接続は必ず使用できるとは限らず、メッセージを格納してから、接続が使用できるようになったときに送信しなければならない場合があるので注意が必要です。
- メッセージに応答がない場合の対処方法を考慮します。通信の状態を管理するには、応答メッセージがない場合に後で処理できるように、ビジネス ロジックで送信メッセージをログに記録できます。
- 受信確認を使用して、強制的にメッセージが正しい順序になるように考慮します。
- インターネット通信には HTTP、イントラネット通信には TCP など、標準的なプロトコルを使用します。ニーズを満たすエンドポイント、プロトコル、および形式の既定の組み合わせが存在しない限り、カスタムの通信チャネルは実装しないようにします。
- 通信でメッセージ応答のタイミングが重要な場合は、応答メッセージが到着までクライアントが待機する非同期のプログラミング モデルの使用を検討します。また、クライアントが応答を待機しながら処理を継続して実行できる場合は、非同期モデルの使用を検討します。

メッセージ ベースの通信方針を設計する際には、安定性、再利用性、パフォーマンス、および設計の全体的な完成度に影響するトピックについても考慮する必要があります。次のセクションでは、これらの問題について詳しく説明します。

- 非同期通信と同期通信
- 結合度と凝集性
- データ形式
- 相互運用性
- パフォーマンス
- 状態管理

非同期通信と同期通信

同期通信と非同期通信を使用した場合の主要なトレードオフについて考慮します。同期通信は、呼び出しを受け取る順序を保証する必要があるシナリオや、呼び出しが返されるまで待機しなければならないシナリオに最適です。一方、非同期通信は、応答性が重要なシナリオや、呼び出し先が使用できるかどうかを保証できないシナリオに最適です。同期通信と非同期通信のどちらを使用するかを決める際には、次のガイドラインを考慮します。

- パフォーマンスを最大限に高め、疎結合を実現し、システムのオーバーヘッドを最小限に抑えるには、非同期通信モデルの使用を検討します。一部のクライアントが同期呼び出ししか実行できない場合は、既存の非同期のサービス メソッドを、クライアントとの同期通信を実行するコンポーネントにラップすることを検討します。
- 処理の実行順序を保証する必要がある場合や、以前の処理の結果に依存する操作を使用する場合は、非同期モデルの使用を検討します。
- 非同期のインプロセス呼び出しでは、プラットフォーム機能 (Begin や End のようなメソッドとコールバックなど) を使用して非同期のメソッド呼び出しを実装します。非同期のアウト プロセスの呼び出し (物理的なティアと境界をまたぐ呼び出しなど) では、メッセージングや非同期のサービス要求を使用することを検討します。

非同期通信を選択したときに、ネットワーク接続や呼び出し先の可用性を保証できない場合は、ストア アンド フォワードのメッセージ配信メカニズムを使用して、メッセージが紛失しないようにすることを考慮します。ストア アンド フォワードの設計方針を選択する場合は、システムやネットワークへの接続が切断した場合にメッセージを後

で配信できるように、ローカル キャッシュを使用してメッセージを格納することを検討します。また、システムやネットワークへの接続が切断されたり、システムやネットワークで障害が発生した場合にメッセージを後で配信できるように、メッセージ キューを使用してメッセージをキューに登録することを検討します。メッセージ キューでは、トランザクション メッセージを配信することが可能で、信頼できる 1 回限りの配信をサポートします。エンタープライズ レベルで他のシステムやプラットフォームと相互運用する必要がある場合や、電子データ交換を実行する必要がある場合は、BizTalk Server を使用して堅牢な配信メカニズムを提供することを検討します。

結合度と凝集性

アプリケーションの分散パーツ間に相互依存性をもたらす通信方法では、アプリケーションが密結合された状態になります。疎結合されたアプリケーションでは、通信を開始するために必要な要件を最小限に抑える方法を使用します。結合度と凝集性を設計する際には、次のガイドラインを考慮します。

- 疎結合では、ASP.NET Web サービス (ASMX) や WCF などのメッセージ ベースのテクノロジ、または自己記述型のデータと HTTP、REST、SOAP などの広く普及しているプロトコルの使用を検討します。
- 結合を維持するには、インターフェイスに、目的と機能分野に密接に関連したメソッドのみが含まれるようにします。

データ形式

ティア間でデータを渡す際に使用する最も一般的なデータ形式は、スカラ値、XML、DataSet、およびカスタム オブジェクトです。データ型を選択する際の主な考慮事項を次の表に示します。

データ型	**考慮事項**
スカラ値	組み込みのシリアル化のサポートが必要です。 スキーマが変更される可能性に対処できます。スカラ値は、メソッド シグネチャの変更を必要とする密結合を作成するので、呼び出し元のコードに影響を及ぼします。
XML	疎結合を使用する必要があります。疎結合では、呼び出し元が把握している必要がある情報は、ビジネス エンティティを定義するデータと、ビジネス エンティティのメタデータを提供するスキーマについてのみです。 サード パーティ製のクライアントを含む、さまざまな種類の呼び出し元をサポートする必要があります。 組み込みのシリアル化のサポートが必要です。

DataSet	複雑なデータ構造をサポートする必要があります。 一連のデータと複雑なリレーションシップに対処する必要があります。 DataSet に含まれるデータへの変更をトラックする必要があります。 組み込みのシリアル化のサポートが必要です。
カスタム オブジェクト	複雑なデータ構造をサポートする必要があります。 オブジェクト型を認識するコンポーネントと通信します。 バイナリ シリアル化をサポートして、パフォーマンスを向上する必要があります。

通信チャネルのデータ形式を選択する際には、次のガイドラインを考慮します。

- アプリケーションで主にインスタンス データを操作する場合は、単純な値を使用してパフォーマンスを向上することを考慮します。単純な値の型を使用すると、初期の開発コストを削減できます。ただし、密結合が作成されるので、今後この型を変更しなければならない場合に、保守にかかるコストが増加する可能性があります。
- XML では、事前のスキーマ定義が他のデータ形式よりも多く必要になる場合がありますが、疎結合が作成されるので、将来の保守にかかるコストを削減して、相互運用性を向上できます (たとえば、インターフェイスを、XML 準拠の呼び出し元に公開しなければならない場合など)。
- DataSet は、複雑なデータ型 (特に、データベースから直接読み込まれる場合) に適切です。ただし、DataSet にはスキーマと状態に関する情報も含まれるため、ネットワーク経由でやり取りするデータの全体量が増加することと、DataSet の特殊な形式によって他のシステムとの相互運用性が制限される可能性があることを理解する必要があります。アプリケーションで主に一連のデータを操作し、並べ替え、検索、データ バインドなどの機能が必要な場合は、DataSet の使用を検討します。
- カスタム オブジェクトは、他のデータ形式では要件が満たされない場合や、カスタム オブジェクトを必要とするコンポーネントと通信する場合に適しています。カスタム オブジェクトは、DataSet に比べてオーバーヘッドが少なく、バイナリ シリアル化と XML シリアル化の両方をサポートします。通常、通信チャネル間のデータ送信に使用するカスタム オブジェクトは、ビジネス エンティティから抽出したデータを含むデータ転送オブジェクト (DTO) になります。
- 通信プロセスで型情報が失われないようにします。バイナリ シリアル化では、型の忠実度が保たれます。これは、クライアントとサーバー間でオブジェクトをやり取りする際に便利です。ただし、この手法では、より厳密なバージョン管理システムをインターフェイスに実装する必要があります。既定の XML シリアル化では、パブリック プロパティとパブリック フィールドのみがシリアル化され、型の忠実度は保たれません。

相互運用性

アプリケーションとコンポーネントの相互運用性に影響する主な要素は、適切な通信チャネルを使用できるかどうかということと、通信に関連するコンポーネントが認識できる形式とプロトコルです。相互運用性を最大限に高めるには、次のガイドラインを考慮します。

- さまざまなプラットフォームおよびデバイスと通信できるようにするには、HTTP などの標準的なプロトコルと、XML のようなデータ形式の使用を検討します。選択するプロトコルは、ターゲット クライアントと通信できるかどうかに影響を及ぼす可能性があることに留意します。たとえば、通信先のシステムは、いくつかのプロトコルをブロックするファイアウォールで保護されている場合があります。
- インターフェイスとコントラクトのバージョン管理に関する問題を考慮します。ビジネス要件の変更、情報技術の要件、または他の問題によって、サービスを変更しなければならない場合があります。このような変更によってインターフェイス、メッセージ コントラクト、またはデータ コントラクトの互換性が失われる場合は、クライアントが使用できる新しいバージョンを作成することを検討します。この新しいバージョンでは、既存のクライアントが新しいインターフェイスで公開される機能にアクセスしなくても、以前のバージョンを使用することができます。詳細については、「サービスのバージョン管理」(http://msdn.microsoft.com/ja-jp/library/ms731060.aspx) を参照してください。
- 選択するデータ形式は、相互運用性に影響を及ぼすことがあります。たとえば、ターゲット システムが特定の種類のプラットフォームを認識しない場合や、ターゲット システムの処理とシリアル化の方法が異なる場合があります。
- 選択するセキュリティの暗号化と復号化の技法は、相互運用性に影響を及ぼすことがあります。たとえば、メッセージの暗号化と複号化の技法は、すべてのシステムで使用できるとは限りません。

パフォーマンス

通信インターフェイスの設計と使用するデータ形式は、(特にプロセス境界やコンピューターの境界を越える場合に)アプリケーションのパフォーマンスに大きな影響を及ぼします。相互運用性など、他の考慮事項では具体的なインターフェイスとデータ形式について考慮する必要がありますが、アプリケーションのレイヤー間やティア間の通信に関連するパフォーマンスの向上に役立つ技法があります。パフォーマンスについては、次のガイドラインを考慮します。

- 不要なデータは、できる限りリモート メソッドに渡さないようにして、ネットワーク経由で送信するデータ量を最小限に抑えます。このようにすると、シリアル化によるオーバーヘッドとネットワークの待ち時間が減少します。ただし、プロセス間の通信とコンピューター間の通信で、粒度の細かい

(chatty な) インターフェイスを使用することは避けます (粒度の細かいインターフェイスでは、クライアントは、1 つの論理単位の作業を実行するために複数のメソッドを呼び出す必要があります)。Façade パターンを使用して、既存の粒度の細かいインターフェイスに粒度の粗いラッパーを提供することを検討します。

- 個々のデータ型を 1 つずつ渡すのではなく、DTO を使用してデータを 1 つの単位として渡すことを検討します。
- アプリケーションでシリアル化のパフォーマンスが重要な場合は、カスタム クラスとバイナリ シリアル化の使用を検討します。
- 相互運用性を確保するために XML を使用する必要がある場合、大量のデータには、要素ベースの構造ではなく、属性ベースの構造を使用することを検討します。

状態管理

アプリケーションの通信に関連する要素では、複数の要求にまたがって状態を管理しなくてはならない場合があります。状態管理の実装方法を決定する際には、次のガイドラインを考慮します。

- 呼び出し間の状態管理は、やむを得ない場合にのみ行います。状態管理はリソースを消費するので、アプリケーションのパフォーマンスに影響を及ぼしたり、配置オプションが制限されたりする場合があります。
- コンポーネントやサービス内でステートフルなプログラミング モデルを使用する場合は、データベースなどの永続的なデータ ストアを使用して状態情報を格納し、トークンを使用して情報にアクセスすることを検討します。
- ASMX サービスを設計する場合は、ApplicationContext クラスを使用して状態を保持することを検討します (ApplicationContext クラスを使用すると、アプリケーション スコープとセッション スコープで、既定の状態ストアへのアクセスが提供されます)。
- WCF サービスを設計する場合は、プラットフォームで提供される拡張可能なオブジェクトを使用して状態を管理することを検討します。拡張可能なオブジェクトを使用すると、サービス ホスト、サービス インスタンス コンテキスト、操作コンテキストなど、さまざまなスコープで状態を維持できます。この状態は、すべてメモリに保持され、永続的なものではありません。状態を永続的に保持する必要がある場合は、(.NET Framework 3.5 で導入された) 永続的なストレージを使用したり、独自のカスタム ソリューションを実装したりできます。

コントラクト ファーストの設計

従来、開発者は、要件に基づいてサービスを設計してから、コードと要件に適したインターフェイスを公開する、"コード ファースト" の手法を使用してサービスを構築していました。ただし、コントラクト ファーストの手法では、異なる種類のシステム間やさまざまなクライアント間で発生する可能性がある互換性の欠如を軽減できることから、この手法がより一般的になっています。

コントラクト ファーストの設計は、公開するデータ、メッセージ、およびインターフェイスに関するサービス コントラクトを設計してから、そのコントラクトに基づいてサービス インターフェイス コードを生成するプロセスです。そこから、必要な処理を実行する、サービス インターフェイスのコードを実装できます。これにより、プロセスの最初の段階で使用するメッセージの形式とデータ型に集中できるので、相互運用性と互換性を最大限に高められます。patterns & practices の Web Service Software Factory: Modeling Edition など、インターフェイスの設計に役立つモデリング ツールを使用することもできます (このツールの詳細については、http://msdn.microsoft.com/servicefactory/ (英語) を参照してください)。また、XML、XSD、およびスキーマを使用してインターフェイスを設計してから、WSDL.exe (/server スイッチを指定) などのツールを使用してインターフェイス定義を生成することも可能です。Microsoft BizTalk Server などのメッセージ バス テクノロジを使用すると、コントラクト ファーストの設計の原理を使用しやすくなります。

コントラクト ファーストの設計を適用する場合に覚えておく必要がある原理は、次のとおりです。

- XML スキーマとデータ型を使用すると、プラットフォーム固有のデータ型を考慮しない (できない) ことになります。このため、インターフェイスを定義するのはさらに難しくなりますが、相互運用性と互換性を最大限に高められます。複雑なデータ構造が必要な場合は、すべてのクライアントが使用できる単純で標準的な XML 形式のデータから、複雑なデータ構造を構成します。
- サービスと通信する可能性があるプラットフォーム、クライアント、およびシステムについて考慮します。これらによって発生する可能性があるデータ型やデータ形式に関する制限に備えます。
- サービス コントラクトの設計に役立つツールの使用を検討します。ツールを使用することで、プロセスを大幅に簡略化および高速化できます。
- コントラクトの設計プロセスでは、可能な限り関係者全員と共同作業を行います。他の関係者が抱えている要件や要求によって、コントラクトが使用しやすくなり、さらに広く受け入れられるようになって、再利用性が最大限に高まることがあります。

コントラクト ファーストの設計の詳細については、「Contract-First Service Development」(http://msdn.microsoft.com/en-us/magazine/cc163800.aspx、英語) を参照してください。

セキュリティに関する考慮事項

安全な通信方針は、機密データがネットワーク経由で渡される際に読み取られるのを防ぎ、機密データが改ざんされるのを防止し、必要に応じて呼び出し元の ID を保証します。通信の保護には、トランスポート セキュリティとメッセージ セキュリティという 2 つの重要な領域があります。保護を最大限に強化するために、トランスポート セキュリティとメッセージ セキュリティの技法を組み合わせることを検討します。

トランスポート セキュリティ

トランスポート セキュリティは、2 つのエンドポイント間のポイント ツー ポイントのセキュリティを提供するために使用し、トランスポート レイヤーでは、ユーザー資格情報と要求を受信者に渡します。チャネルを保護すると、攻撃者がチャネル上のすべてのメッセージにアクセスするのを防ぎます。トランスポート セキュリティの一般的な手法は、Secure Sockets Layer (SSL) 暗号化とインターネット プロトコル セキュリティ (IPSec) です。トランスポート セキュリティを使用するかどうかを決める際には、次の事項を考慮します。

- トランスポート セキュリティでは、優れた相互運用性を実現する一般的な業界標準を使用します。トランスポート セキュリティは、下位レイヤーで実現できるので (ネットワーク ハードウェアでも実現できることがあります)、通常、暗号化と署名でより高速に機能します。ただし、サポートされる資格情報と要求は、メッセージ セキュリティより少ないというデメリットがあります。
- サービスとコンシューマーの間の通信が中間デバイス経由でルーティングされない場合は、トランスポート セキュリティを使用できます。メッセージが中間デバイスを経由する場合は、メッセージ セキュリティを使用します。この場合にトランスポート セキュリティを使用すると、メッセージは、経由する中間デバイスで復号化されてから再度暗号化されるので、セキュリティ リスクとなります。
- トランスポート セキュリティは、イントラネットなどの社内ネットワークに配置されたクライアントとサービス間の通信を保護するのに適しています。

メッセージ セキュリティ

メッセージ セキュリティは、あらゆるトランスポート プロトコルと併用できます。チャネル経由で渡される個々のメッセージのコンテンツは、安全なネットワークの外側で渡される際には必ず、機密性の高いコンテンツについては、安全なネットワーク内で渡される場合であっても、保護する必要があります。メッセージ セキュリティの一般的な手法は、暗号化とデジタル署名です。メッセージ セキュリティを使用するかどうかを決める際には、次のガイドラインを考慮します。

- インターネット経由で公開されるサービスなど、安全なネットワークの外側に渡される機密メッセージには、メッセージ セキュリティを実装することを検討します。ただし、一般的に、メッセージ セキュリティはトランスポート セキュリティよりも通信のパフォーマンスに大きな影響を及ぼすので、注意が必要です。メッセージの暗号化と署名は、部分的または特定のメッセージにのみ使用して、全体的なパフォーマンスを向上することもできます。
- クライアントとサービスの間に中間デバイスが存在する場合は、エンド ツー エンドのセキュリティが保証されるので、機密メッセージにはメッセージ セキュリティを使用します。中間サーバーは、メッセージを受信したときに SSL 接続や IPSec 接続を終了し、メッセージを次のサーバーに渡すために、新しい SSL 接続や IPSec 接続を作成します。そのため、メッセージ セキュリティを使用していないメッセージは、中間サーバーでアクセスされる危険性があります。

テクノロジの選択肢

マイクロソフト プラットフォームでは、Windows Communication Foundation (WCF) と ASP.NET Web サービス (ASMX) のどちらかのメッセージ テクノロジを選択できます。これ以降のセクションでは、各テクノロジの機能を理解し、シナリオに最適なテクノロジを選択するのに役立つ情報を提供します。

WCF テクノロジの選択肢

WCF では、さまざまな状況でサービスを実装するための包括的なメカニズムが提供され、サービスの構成と内容を細かく制御できます。次のガイドラインは、WCF の使用方法について理解するのに役立ちます。

- 次の状況では、WCF を使用することを検討します。
    - SOAP をサポートする他のプラットフォームとの相互運用性を確保する必要がある Web サービスと通信する場合 (J2EE ベースのアプリケーション サーバーなど)
    - SOAP ベースではないメッセージを使用して Web サービスと通信する場合 (RSS (Really Simple Syndication) などの形式を使用するアプリケーションなど)
    - サーバーとクライアントの両方が WCF を使用する場合に、SOAP メッセージとデータ構造のバイナリ エンコードを使用して通信する場合
    - REST Singleton & Collection Services (REST シングルトンと収集サービス)、ATOM Feed and Publishing Protocol Services (ATOM フィードと出版プロトコル サービス)、および HTTP Plain XML Services (HTTP プレーン XML サービス) を構築している場合

- SOAP 要求で WS-MetadataExchange を使用して、サービスに関する情報 (サービスの Web サービス記述言語 (WSDL) 定義とポリシーなど) を取得することを検討します。
- WS-Security を使用して、認証、データ整合性、データ プライバシー、およびその他のセキュリティ機能を実装することを検討します。
- WS-Reliable メッセージングを使用して、複数の Web サービスの中間デバイスを経由する必要がある場合でも、信頼できるエンド ツー エンドの通信を実装することを検討します。
- WS-Coordination を使用して、Web サービスのメッセージ交換コンテキストで、2 フェーズ コミット トランザクションを調整することを検討します。

WCF では、いくつかの通信プロトコルがサポートされます。

- インターネットからアクセスするサービスでは、HTTP プロトコルの使用を検討します。
- プライベート ネットワーク内でアクセスするサービスでは、TCP プロトコルの使用を検討します。
- 同じコンピューターからアクセスするサービスでは、名前付きパイプ プロトコルの使用を検討します。このプロトコルでは、共有バッファやデータを渡すためのストリームがサポートされます。

ASMX テクノロジの選択肢

ASMX では、インターネット インフォメーション サービス (IIS) Web サーバー経由で公開される ASP.NET ベースの Web サービスを構築するための、簡略化されたソリューションが提供されます。ASMX には次の特性があります。

- インターネット経由でアクセスできますが、使用できるプロトコルは HTTP のみです。既定でポート 80 が使用されますが、これは簡単に再構成できます。
- 分散トランザクション コーディネーター (DTC) のトランザクション フローがサポートされません。カスタム実装を使用して、長時間トランザクションをプログラムする必要があります。
- IIS 認証、承認に使用する Windows グループとして格納される役割、IIS と ASP.NET の偽装、および SSL トランスポート セキュリティがサポートされます。
- IIS に実装されているエンドポイント テクノロジをサポートします。
- プラットフォーム間の相互運用性が提供されます。

関連情報

Web リソースに簡単にアクセスするには、http://www.microsoft.com/architectureguide (英語) でオンライン版の参考文献を参照してください。

- データ転送とシリアル化
  (http://msdn.microsoft.com/ja-jp/library/ms730035.aspx)
- エンドポイント: アドレス、バインディング、およびコントラクト
  (http://msdn.microsoft.com/ja-jp/library/ms733107.aspx)
- SOA のメッセージングパターン (パート 1)
  (http://msdn.microsoft.com/ja-jp/library/cc947720.aspx)
- Principles of Service Design: Service Versioning
  (http://msdn.microsoft.com/en-us/library/ms954726.aspx、英語)
- Web Service Messaging with Web Services Enhancements 2.0
  (http://msdn.microsoft.com/en-us/library/ms996948.aspx、英語)
- Web サービス プロトコルの相互運用性ガイド
  (http://msdn.microsoft.com/ja-jp/library/ms734776.aspx)
- Windows Communication Foundation セキュリティ
  (http://msdn.microsoft.com/ja-jp/library/ms732362.aspx)
- ASP.NET を使用した XML Web サービス
  (http://msdn.microsoft.com/ja-jp/library/ba0z6a33.aspx)