Microsoft
.NET

～ＣＬＲの研究とＷｅｂサービスの試作～

平成１５年2月１０日

大分県情報サービス産業協会
技術研究会 ドットネット部会

目次

1　はじめに

2000 年 6 月、マイクロソフトが「.NET（ドットネット）戦略」を発表してから、2 年半が経過した。発表当時は未知の部分が多かった Microsoft.NET も、昨年のフレームワークリリースや、開発ツールのリリースで、我々開発者にとっては、一段と身近な存在となりつつある。

.NET とはコンピューティングとインターネットを融合することで、次世代のインターネット環境を構築するというコンセプトであり、従来のインターネットの問題を根本から解決し、ユーザーと開発者の両方に大きな恩恵をもたらすものである。

これからのインターネットの世界が、この.NET を軸に大きな転換を遂げることは間違いなく（現に、キーテクノロジーである SOAP・UDDI は Microsoft プラットフォームばかりではなく、JAVA プラットフォームでも採用され稼動できる状況にある）、すべての IT 関係者にとって、.NET を正しく理解することは必須条件と言ってもよい。

本部会では、.NET の本質とそれを支えるテクノロジー、そして XML Web サービスを利用したアプリケーションを実際に構築する事で、その理解を深めていく事にした。

2　.NET について

2.1　.NET の定義

マイクロソフトは、『.NET は Microsoft の XML Web サービスのためのプラットフォーム』と明言してる。

XML Web サービスにより、アプリケーションはオペレーティングシステムや言語の種類を問わず、インターネット上で通信を行い、人間の手を借りることなくソフトウェア同士が会話する、分散オブジェクト環境を容易に構築する事が可能となる。

> 現在でも Web のしくみを使った情報サービスは多数あり、日々増加の一途をたどっている。しかし、これらの情報サービスは、ソフトウェアが機械的に利用可能な「XML Web サービス」ではなく、人間が Web ブラウザを通じて表示し、操作する「Web アプリケーション」である。

この XML Web サービスを実現する為に必要なものすべてを「.NET プラットフォーム」と呼び、.NET プラットフォームの構成要素には、次のようなものがある。

- **ユーザーエクスペリエンス**
- **クライアント**
- **サービス**

- **サーバー**
- **ツール**

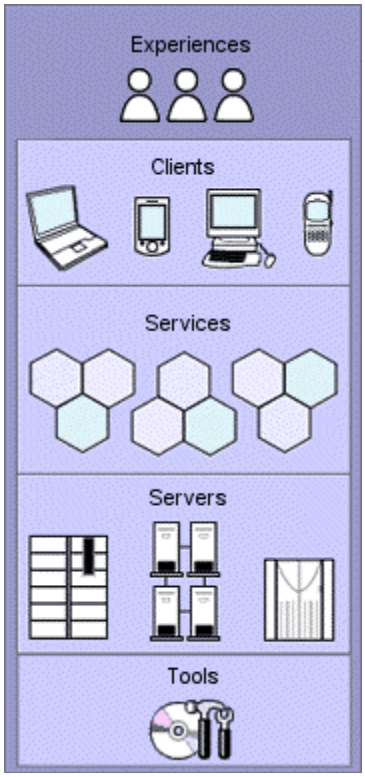
2.1.1 ユーザーエクスペリエンス

ユーザー体験。ユーザーインターフェイスよりもさらに広義の意味があり、意訳すれば、「使い勝手」となるであろう。

XML Web サービスを利用しているアプリケーションをいつでも、どこでも、どのような形ででも、同一の使い勝手で享受できると言うことである。

2.1.2 クライアント

クライアントとは、PC、ノート PC、ワークステーション、携帯電話、PDA、タブレット PC、ゲーム機器、および、その他、XML Web サービスにアクセスできる、すべてのデバイスを指す。



.NET プラットフォームのイメージ

2.1.3 サービス

XML Web サービスや、XML Web サービスを利用したアプリケーションを容易に開発する為に、マイクロソフトが用意したビルディングブロックサービス（完成済みの XML Web サービス）である。

2.1.4 サーバー

XML Web サービスの導入、運用、管理、およびオーケストレーションの為の、サーバーインフラストラクチャである。

2.1.5 ツール

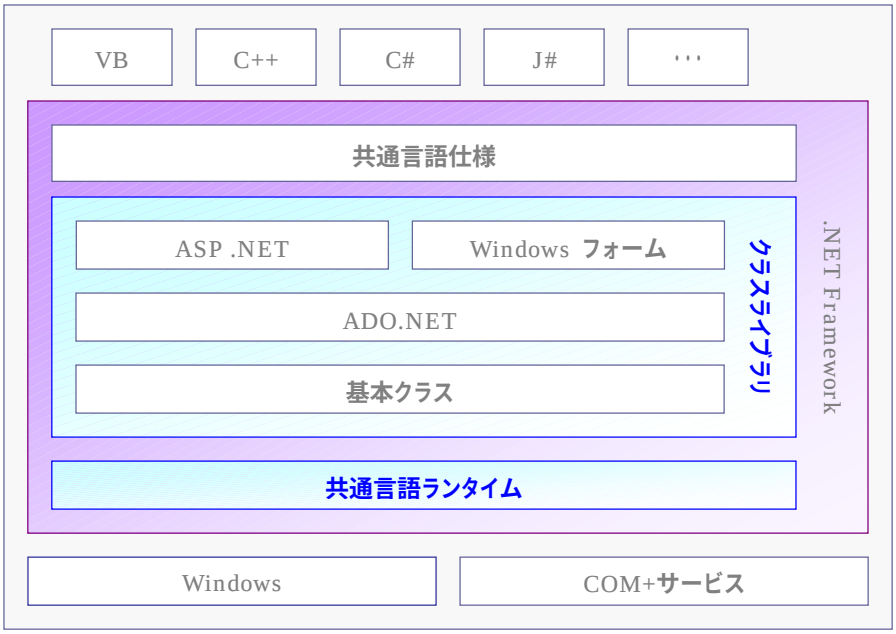
XML Web サービスを開発、実行する為の開発ツールを意味し、現時点では、.NET Framework と Visual Studio .NET がこれに相当する。

.NET Framework は.NET のキーテクノロジーであり、複数言語アプリケーション実行環境である。

2.2 .NET Framework

.NET Framework は、インターネットの高度な分散環境において、アプリケーション開発を簡単に行うことができる新しいコンピューティング プラットフォームであり、次の目的を実現するためにデザインされている。

- オブジェクト コードがローカルに保存され実行されるか、インターネットに分散されローカルに実行されるか、リモートで実行されるかにかかわらず、一貫したオブジェクト指向プログラミング環境を提供すること。
- ソフトウェア配置やバージョン管理の競合を最小化するコード実行環境を提供すること。
- 作成者が不明なコードや、信頼性の高くないサード パーティ製のコードも含めて、コードの安全な実行を保証できるコード実行環境を提供すること。
- スクリプトやインタープリタが実行される環境でのパフォーマンスの問題を回避するコード実行環境を提供すること。
- Windows ベースや Web ベースのアプリケーションなど、多種のアプリケーションの開発において、ユーザーが一貫した方法で作業できること。
- .NET Framework に基づいたコードをほかの任意のコードと統合できるように、すべての通信を業界標準に準拠して構築すること。



.NET Framework アーキテクチャ

.NET Framework には、共通言語ランタイムと.NET Framework クラスライブラリという 2 つの主要なコンポーネントがある。

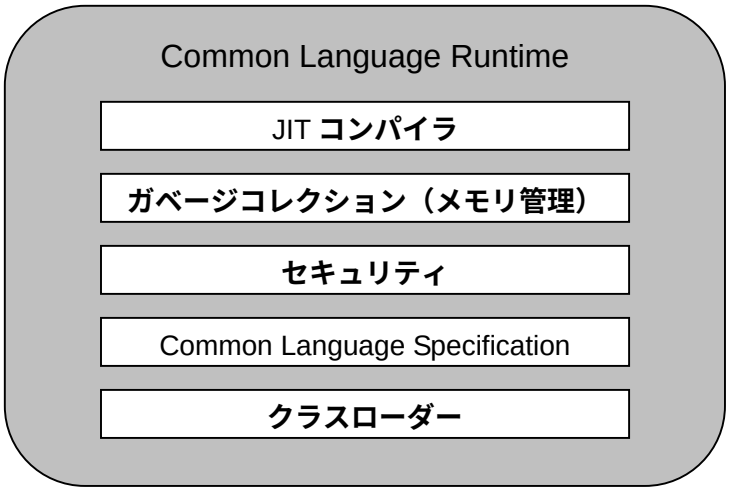
共通言語ランタイムは、.NET Framework の基礎となるコンポーネントであ

る。このランタイムは、実行時のコード管理を行うエージェントのようなものであり、メモリ管理、スレッド管理、リモート処理などの重要なサービスを提供したり、セキュリティや保全性を確保するために厳密なタイプ セーフやその他のコード整合性を強制的に適用したりする。

クラスライブラリは、従来のコマンドラインやグラフィカルユーザーインターフェイス(GUI)を使用したアプリケーションから、ASP.NET がもたらした最新の機構に基づく Web フォームや XML Web サービスなどのアプリケーションに至るまでのアプリケーション開発に使用できる、再利用可能な型の包括的なオブジェクト指向コレクションである。

3　共通言語ランタイム（ＣＬＲ）について

共通言語ランタイム（Common Language Runtime：CLR）とは、.NET Framework アプリケーションの基盤となり、プログラミング言語に依存しない中間言語『MSIL（MicroSoft Intermediate Language）』を実行するための環境および仕様のこと。この環境を実現するために、以下に挙げるテクノロジーが提供されている。



共通言語ランタイムの構成

・JIT コンパイラ

各プログラム言語に対応したコンパイラによって中間コードに変換されたプログラムを、実行時に CPU ネイティブコードに変換するコンパイラ。

・ガベージコレクション（メモリ管理）

プログラム実行中に作成されたオブジェクトインスタンスが、どこからも参照されなくなると、必要に応じて自動的に削除する機能。

・セキュリティ

リソースアクセスなどが有効かどうかを実行エンジンレベルでチェック。

・Common Language Specification（CLS：共通言語仕様）

異なるプログラミング言語で作成されたプログラム間の相互運用性をサポートするために定められた規約

・クラスローダー

プログラムロード時に、タイプセーフであるかなど、有効なプログラムかどうかをチェックする機能。

この章では、CLR の中で特に重要である、JIT コンパイラ、Common Language Specification、ガベージコレクションについて後述する。

3.1 ＪＩＴコンパイラ

3.1.1 ＪＩＴコンパイラとは

VB、C#など、各プログラミング言語のコンパイラが生成した中間言語 MSIL をプラットフォームのマシンコード（ネイティブコード）に変換する方法。

既に Java に取り入れられている技術であり、Java の実行方式は基本的にはコンパイルにより中間コードであるクラスファイルを生成し、Java 仮想マシン（JVM）が解釈しながらプログラムを実行する。

一方.NET では、各プログラム言語に対応したコンパイラにより中間コードが生成された後、JIT コンパイラにより CPU に対応したネイティブコードにコンパイルされる。

Java での実行方式のように、全てのコードを一気にコンパイルするのではなく、呼び出すメソッドごとにコンパイルして、CPU 資源の浪費を防ぐという機能を提供している。

3.1.2 ＪＩＴコンパイラの種類

各言語ソースから MSIL にコンパイルされ、その MSIL をさらに JIT コンパイラがネイティブコードにコンパイルする際に、ネイティブコードの生成効率を上げるために、３つのコンパイルモデルがある。

・エコノミー（「エコノモード」、「“Econo”－JIT」）

“最適化されていない”ネイティブコードを生成する。コードはキャッシュされるが、場合によっては破棄されて生成し直される。最適化の作業には時間的なリスクが伴う。

つまり、より速いコードを生成するためには、コンパイルの段階で最も効率のいいコードを作り出す必要があるため、その作業に時間が必要となる。

最適化の作業に時間が掛けられない場合、取り敢えず動くコードが早く欲しいといった場合に使用する。Web アプリケーション等、頻繁に修正が行われるようなときに有効である。

・スタンダード（「“Standard”－JIT」）

“最適化された”ネイティブコードを生成する。このとき、MSIL は、セキュリティのアクセスパーミッションやタイプ情報をもとに、コードの整合性検証を行う。

なお、CLR では「“Standard”－JIT」が規定のネイティブコード

コンパイラとして使われる。「“Standard”‐JIT」は「“Econo”‐JIT」に比べて、実行時の効率を重視するような、リッチなクライアントアプリケーションや Web サービスを提供するためのロジックの実行に有効である。

・PreJIT（「“Pre”‐JIT」）

ネイティブコードへの変換を行うタイミングが、エコノミー・スタンダードのように初めて呼び出されたときではなく、サービスやアプリケーションをインストールしたとき、もしくは呼び出し要求時にコードを生成する。

この方式のメリットは、あらかじめインストール時にコンパイル作業が行われているため、スタートアップ時の作業はなく、時間を削減できることにある。

3.1.3 ＭＳＩＬへのコンパイル

MSIL は CPU に依存しない一連の命令で、効率的にネイティブコードに変換できる。MSIL には、オブジェクトに対する読み込み、格納、初期化、および呼び出し用の命令のほかに、算術演算と論理演算、制御フロー DMA（Direct Memory Access）、例外処理、およびその他の操作のためのの命令も含まれている。

コンパイラは、MSIL を生成するときにメタデータも生成する。メタデータには、コード内の型について、それぞれの型の定義型のメンバのシグネチャ、コードが参照するメンバ、共通言語ランタイムが実行時に使用するその他のデータなどが記述されている。

MSIL とメタデータは、実行可能ファイルのファイル形式として使用されてきた従来の Microsoft PE と COFF（Common Object File Format）に基づき、それらを拡張したポータブル実行可能（PE）ファイルに格納される。MSIL、ネイティブコード、およびメタデータを保存できるこのファイル形式を使用すると、オペレーティングシステムが共通言語ランタイムのイメージを認識できるようになる、MSIL と共にメタデータがこのファイルに格納されるため、コードは自己記述型になる。つまり、タイプライブラリやインターフェース定義言語（IDL：Interface Definition Language）は必要なくなる。共通言語ランタイムは、実行時にこのファイルから必要に応じてメタデータを検出および抽出する。

3.1.4 ＭＳＩＬからネイティブコードへのコンパイル

MSIL は実行する前に.NET Framework の JIT コンパイラによってネイティブコードに変換する必要がある。ネイティブコードは CPU 固有のコードで、JIT コンパイラと同じコンピュータアーキテクチャ上で実行される。共通言語ランタイムは、サポートしている CPU アーキテクチャごとに JIT コンパイラを提供しているため、開発者は 1 セットの MSIL を記述して JIT コンパイルし、異なるアーキテクチャの複数のコンピュータ上で実行できる。しかし、マネージコードがプラットフォーム固有のネイティブ API またはプラットフォーム固有のクラスライブラリを呼び出す場合、そのコードは特定のオペレーティングシステムでしか実行できない。

JIT コンパイラは、実行時に呼び出されることがないコードがあることを考慮している。つまり、ポータブル実行可能（PE）ファイル内にあるすべての MSIL をネイティブコードに変換するために時間とメモリを費やすのではなく、実行時に必要になった MSIL に変換し、その結果生成されたネイティブコードを保存して、以降の呼び出しで利用できるようにしておく。型が読み込まれると、ローダーはスタブを作成し、その型の各メソッドに結び付ける。メソッドが初めて呼び出されると、スタブは JIT コンパイラに制御を渡し、JIT コンパイラがそのメソッドの MSIL をネイティブコードに変換し、そのネイティブコードを直接実行するようにスタブを変更する。それ以降は、この JIT コンパイル済みのメソッドを呼び出すと生成済みのネイティブコードが直接実行され、コンパイルとコードの実行に必要な時間を節約できる。

共通言語ランタイムには、インストール時コード生成と呼ばれる別のコンパイルモードも用意されている。インストール時コード生成モードは標準の JIT コンパイラと同じ方法で MSIL をネイティブコードに変換するが、より大きなコード単位を一度に変換し、生成したネイティブコードを後からアセンブリが読み込まれて実行されるときに使用できるように保存している。

インストール時コード生成を使用すると、既にインストールされているほかのアセンブリに関する情報を考慮しながら、インストール中のアセンブリ全体がネイティブコードに変換される。インストール時コード生成で生成されたファイルは、標準の JIT オプションでネイティブコードに変換された場合よりもすばやく読み込まれて起動される。

3.2 CLS

CLS とは Common Language Specification（共通言語仕様）と表記されるもので、異なるプログラミング言語で作成されたプログラム間の相互運用性をサポートするために定められた規約である。以下、この CLS を説明していく。

3.2.1 CLR を実現させる規約

CLS を説明するにあたり、CLR の言語仕様、CTS について触れる必要があります。まず、CLR の言語仕様について説明していく。以下の三点は CLR の主な言語仕様であり、またそれに伴い定義すべき規約について列挙している。

（1）CLR にネイティブな言語は IL である。

.NET では、その上で動作するプログラムは基本的に IL コードの形で、CLR が解釈、実行する。そのため、CLR と IL 間で整数や、浮動少数点数などについてデータ型を共通に規定する必要がある。

（2）CLR は複数言語に対応した実行環境である。

CLR では、複数の言語を用いてプログラミングすることが可能である。しかし CLR にネイティブな言語は（1）の通り、IL のみであり、その上の層に位置する各プログラミング言語では、IL でサポートされるデータ型、インストラクションなどの要素と、自らの言語が持つデータ型、文法などとの間でのすり合わせが必要となる。

これらの要素が共通化していれば、異なるプログラミング言語で作成したプログラム間でのデータの受け渡しや、メソッドの呼び出しが可能となる。

（3）CLR はオブジェクトプログラミングをサポートする。

.NET ではオブジェクト指向により、プログラムの再利用性を高めている。そのために CLR で、再利用可能なデータ構造の定義、実行時にバインディングされるメソッドの呼び出し、例外処理などの機能を提供する必要がある。これらの機能をその上で動作するプログラム言語が使用するには、CLR との間にやはりすり合わせの必要が発生してくる。

上記の点から、CLR の中核となる機能を実現するためにはこのような規約が必要となることが分かる。この細かい規約の役割を担っているのが次に述べる CTS（Common Type System）である。

3.2.2 CTS(Common Type System)

前節で紹介したように、CLR はプログラム言語に関して規約を必要とする。CTS はこの規約の役割を担っており、プログラム言語間で共通の基本データ型、データ型定義、命名規約などを定めている。つまり、CTS は CLR をサポートするプログラミング言語がどういうタイプシステムを採用するべきか記述するものである。

では.NET 上で動作するすべてのプログラミング言語はこの CTS に完全に準拠しなければならないのか、となるとそうではない。もちろん、先にのべた三点の要件を満たすためには完全な準拠が必要であるが、その中で例えば IL のみをターゲットとした言語であるならば、それは必要ではない。

しかし、CLR が異なる言語で作成されたプログラム間（CTS 完全準拠のものとそうではないもの）での相互運用をサポートするためには、さらなる規約が必要となる。それが CLS である。

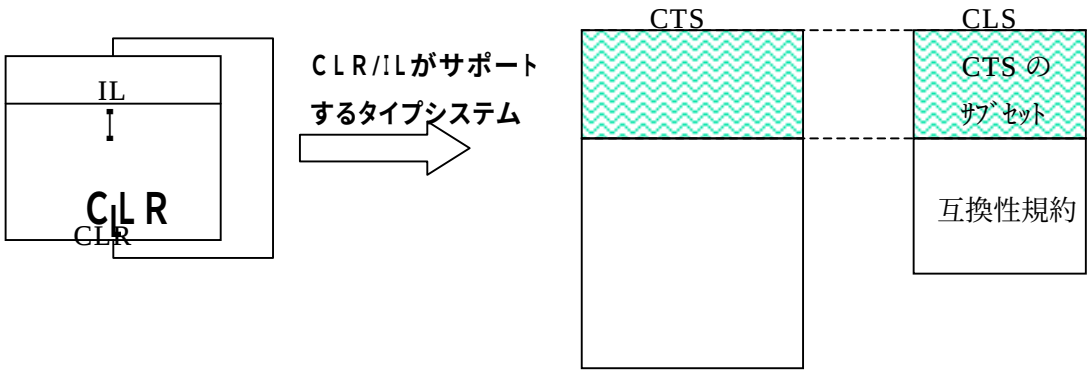
3.2.3 CLS(Common Language Specification)

CLS は前節で述べたように異なる言語間での相互運用性をサポートする上での規約を定めている。

各プログラムはメタデータという形で、そのプログラムが外部に対して公開するインターフェイスを自己記述する。またデータ型やメソッド呼出形式などはCTSに準拠したサブセットを定め、これを遵守させることで、異なるプログラミング言語間での互換性をもたせている。

CLS が規定するのは、あくまでプログラムが外部に公開するインターフェイス部のみであって、その実際の中身は CTS のサブセットとプログラム間の互換性についての規約である。

つまり、CLS により.NET Framework 上で動作する各プログラミング言語同士の相互運用の実現が可能となっている。



3.3 ガベージコレクション

3.3.1 現状

プログラムを高速に動作させるために必要な条件の一つにいかにメモリの使用量を減らすかということがある。UNIX や Windows NT などの一般的な仮想記憶システムではプログラムの使用するメモリの容量の増大は以下の 2 つの点で性能に影響する。

プログラムの必要部分が実メモリに乗っている確率が低くなり、ページフォルトの頻度が高くなる。ページフォルトの発生はプログラムの実行性能を大きく低下させる。

プログラムのワーキングセットの増大はシステム上で動作している他プロセスの使用できるワーキングセット量を減らし、システム全体のパフォーマンスが低下する。

Windows の場合、どの程度本格的なメモリ管理を OS が行っているのか判らないが、メモリの使用量が性能にインパクトを与えるのはまず間違いないだろう。さらに Java のようにガベージコレクションを行うプログラムではガベージコレクタの振舞いが性能劣化の要因として加わってくる。ガベージコレクションはプログラムが使用したメモリの中で、未使用となったものを探し出して回収する処理である。ガベージコレクタが頻繁及び長時間動作するほどプログラムの性能は低下する。

もちろんメモリ使用量を減らすために余分なロジックが必要になるのでは得失は微妙であるが、メモリ使用量の削減は常に意識しておく必要がある最適化項目といえるだろう。

3.3.1.1 開発者サイドでは

C++の様に年季の入った言語では、プログラマーが作成したオブジェクトを自分で削除しなければならない為、これによって 2 つの問題が生じる。

(1) プログラマーがオブジェクトを生成し、使い終わった時に削除するのを忘れる事。
こうしたリークは、最終的にプロセスのメモリ空間を全て使い果たし、クラッシュを引き起こす。

(2) オブジェクトを削除した後に、そのメモリ領域に誤ってアクセスすると言う事。

削除されたオブジェクトメモリを構成していたトランジスタに、たまたま最もらしい値が含まれていると、プログラムはその誤ったデータを元に動作し続ける。これらのバグはなかなか再現出来ず、追跡するのが厄介である。現状上記に当てはまるものは VC++であり、VB ではこの種の問題は発生しない。

VB6.0 ではオブジェクトの参照をカウントし、オブジェクトのカウントが 0 になったら、オブジェクトを自動的に削除し、メモリを回収する。

3.3.2 ガベージコレクションとは

3.3.1 の中で若干述べたが、ガベージコレクションとは　GC(Garbage Collection)＝「ごみ集め」の意味である。コンピュータでごみ集めと言えば「使わなくなったメモリを自動的に解放すること」を指す。C 言語で言えば、malloc したメモリのうち使わなくなったものを free してまわることが相当する。

ガベージコレクションを一言で表すと、メモリに残っている不要なデータやプログラムで取り出せないデータが占有している無駄な記憶領域（ガベージ）を洗い出し、その記憶領域を再び使えるようにする、となるだろう。

書籍などに「使えるメモリがこまぎれになっているのを整理するのが GC」とあるがそれは二次的な目的である。実際、メモリの回収はしても詰め直し(compaction)はしない GC も多くある。例えば Ruby の GC などがそれにあたる。

ガベージコレクションは、オブジェクトのライフタイムと、オブジェクトの占めるヒープメモリの管理にランタイムコンポーネントが責任をもつシステムである。この概念は、.NET や Java が最初ではない。ガベージコレクションを利用する開発言語や実行環境は他に Lisp、Prolog、Smalltalk などたくさんある。

今回の Microsoft.NET Framework でも、プログラマーがリソース管理にわずらわされずにビジネスロジックに専念出来るようにガベージコレクションを実装してきたと言う事が言える。

3.3.3 .NET Framework のメモリ管理方式概略

3.3.3.1 メモリの割り当て

アプリケーションが new 演算子を使ってオブジェクトを作成すると､ガベージコレクタは管理ヒープにて連続したメモリの割り当てを行っていく。よって、スタックからのメモリ割り当てと殆ど同じ速度となる。又、アプリケーションからそれらのオブジェクトに高速にアクセス出来る。

3.3.3.2 メモリの解放

ガベージコレクション起動のタイミングは、アプリケーションが新しいオブジェクトを作成しようとした時にジェネレーション 0 が一杯であった時実行される。(ジェネレーションについいは後述。)

ガベージコレクションは､アプリケーションが使用しなくなったオブジェクトのメモリを解放する。

そして、残ったオブジェクトをメモリの下位アドレスにシフトし、ヒープの中のギャップを全て取り除く。(メモリコンパクションと言われる。)

3.3.3.3 ジェネレーションとパフォーマンス

ガベージコレクタのパフォーマンスを最適化する為に、管理ヒープは 3 つのジェネレーション(0～2)に分類される。

この手法は Microsoft がこれまでのガベージコレクション手法の経験から、コンピュータソフトウェア業界で有効だと認識される以下の原則に基づいている。

・管理ヒープの一部のメモリコンパクションは、全体のメモリコンパクションより高速。
・オブジェクトが新しい程その存続期間は短く、古いほど長い。
・新しいオブジェクトは相互に関連を持つ傾向があり、アプリケーションによって同時に頻繁にアクセスされる。

上記より、ガベージコレクタがコレクションを実行した際は管理ヒープ全体をチェックするのではなく、特定のジェネレーション(まず 0)だけをチェックし開放する。

方式は、

(1)　ジェネレーション 0=Full 状態でのオブジェクト作成要

求時、ガベージコレクション実行。

(2) ジェネレーション 0 に残ったオブジェクトをジェネレーション 1 へ移す。

(3) ジェネレーション 0=Full 状態でのオブジェクト作成要求時、ガベージコレクション実行。

(4) ジェネレーション 0 に残ったオブジェクトをジェネレーション 1 へ、ジェネレーション 1 のオブジェクトをジェネレーション 2 へ移す。

尚、ジェネレーションをどれだけコレクションするかは、オブジェクト割り当て時点で判断している。

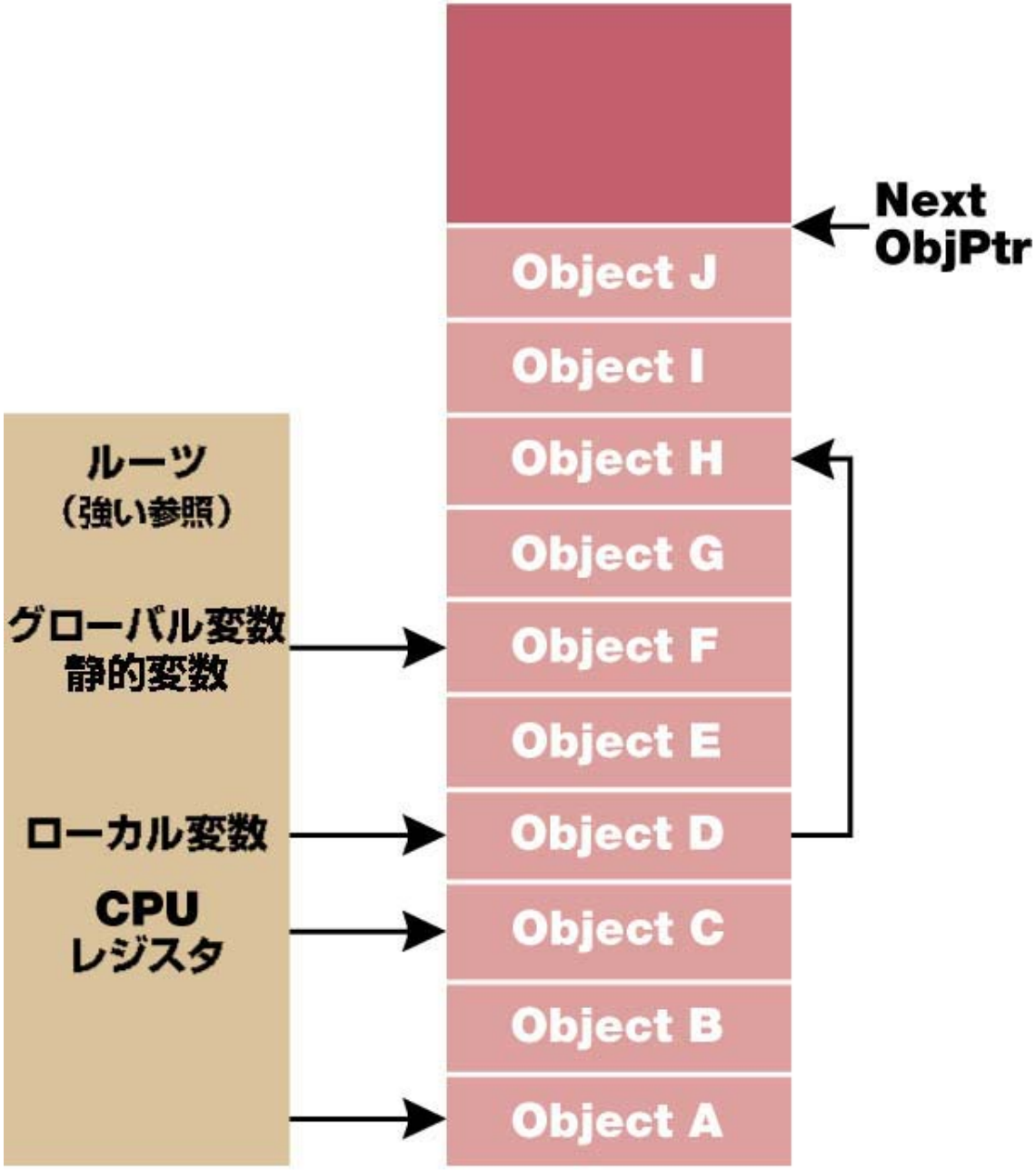
3.3.4 その他調査事項

（1） ガベージコレクションの統計値

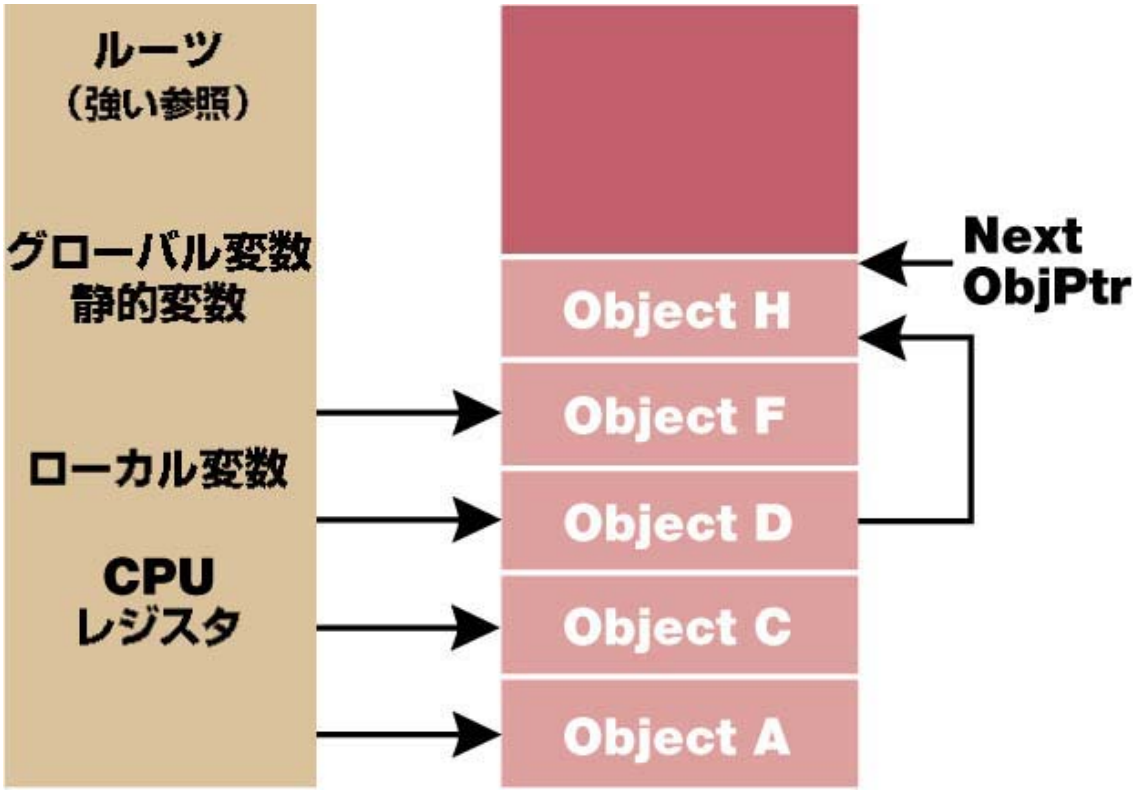
.NET ランタイムから“ COM+Memory ”というパフォーマンスオブジェクトとして公開されている。

（2） ガベージコレクションの必要性

Microsoft は、GC を使わなければ CLR の多くの利点がなくなると主張している。ただし、強制的に.NET Framework に GC を使わなくするようにはできない。C#開発者は特定のキーワードを使用して GC を避けることができる。C++開発者はクラスに _gc キーワードをつけて GC を使うことを選択しなければならない。しかし、GC を避けると、.NET クラスとの連携に重要な問題があり、特に開発言語間の連携に影響するため推奨されていない。



ヒープ内に確保されたオブジェクト



ガベージコレクション終了後の管理ヒープ

4　Web サービス試作

4.1 Microsoft.NET と Web サービス

Microsoft.NET の特徴のひとつに、Web サービスを容易に構築・利用できる事が挙げられている。Web サービスは昨今話題になっている技術であり、今後普及する可能性が高い。

そこで、どのくらい容易に Web サービスを構築・利用できるのかを検証するため、試作プログラムを作成した。

以下、Web サービスの概要、利点と欠点、及び試作プログラムについての説明を行う。

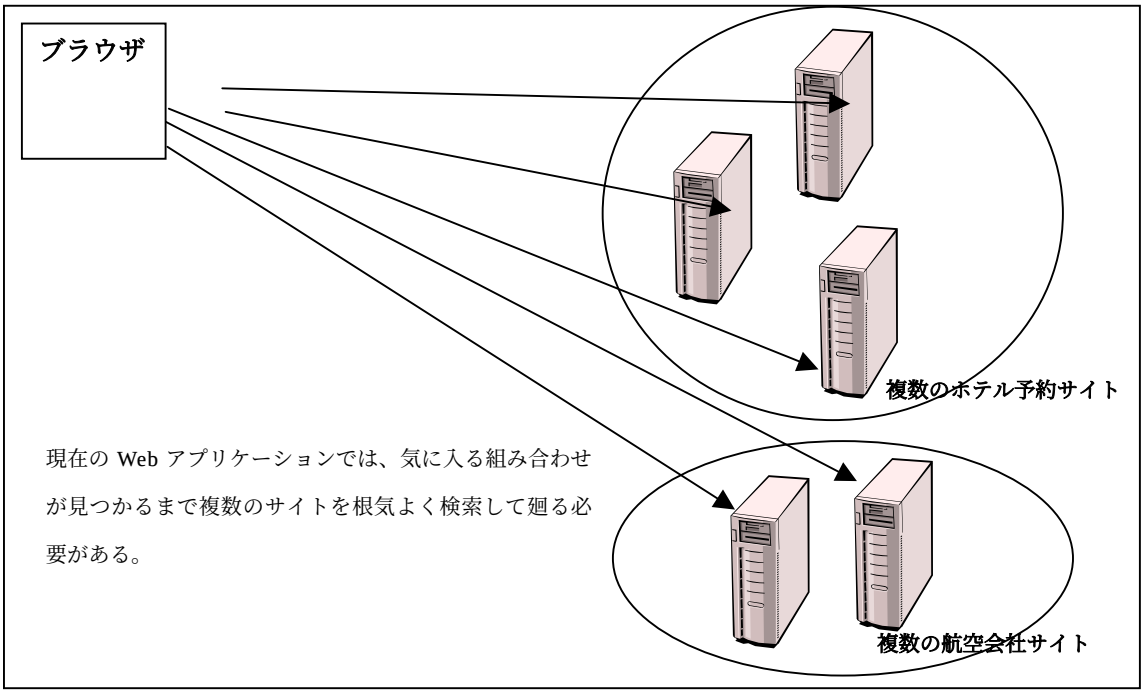
4.2 Web サービスの概要

Web サービスは、インターネットやイントラネットを通じて、HTTP プロトコルを経由して、プログラム的にアクセスできる汎用的なソフトウェアサービスである。

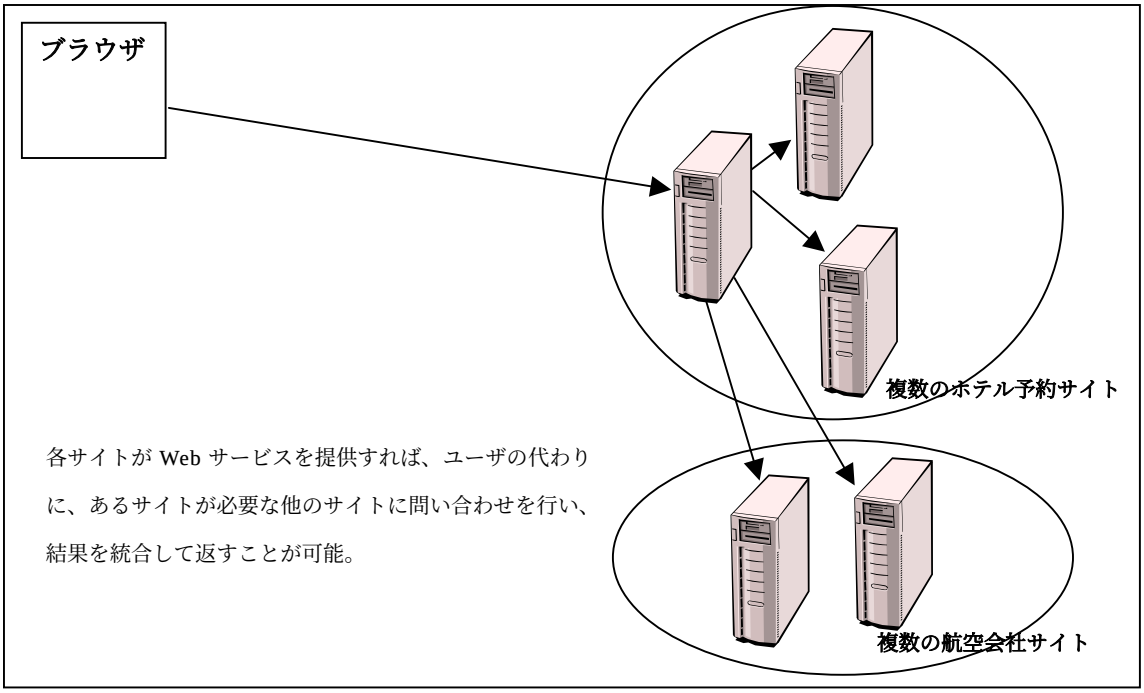
通常の Web アプリケーションは、ブラウザに表示された情報を人間が見ることを前提としている。このため、Web アプリケーションが提供している機能を、他のアプリケーションから複合的に利用するのが難しい。

一例として、Web サイトを利用して旅行の手配を行う場合考えてみる。
Web サイトでホテルと交通手段の手配を行う場合、いろいろなサイトを巡って、個々にホテルや航空機の予約を行わなければならない。
この場合、個々に空き状況や価格を確認し予約することになるが、例えば最も価格が安くなる組み合わせを探したいとすれば、ユーザーが自分でメモする等を行わなければならない。

一方、Web サービスはプログラムから呼び出され、Web サービスが返した情報はプログラムが処理することを前提としている。
このため、先の例をとると、ホテルのサイトと航空会社のサイトが Web サービスを公開し、これを統合して検索するサービスがあれば、ユーザーは総合的な条件（例えば最も価格が安い組み合わせなど）を指示することができる。



現在の Web アプリケーションの場合



各サイトが Web サービスを提供している場合

先の例はサーバ間通信であったが、Web サービスは XML ベースでやりとりを行うため、利用する側はクライアントや他の Web サービス（サーバ）を問わな

い。また、HTTP＋XML ベースのため、Web サービスを提供する側、またサービスを受ける側のハードウェアや OS を問わない。

今後、この汎用性を活かした活用方法が期待されている技術である。

4.3 Web サービスの利点、及び欠点

Web サービスは、現在の SOAP（Simple Object Access Protocol）と呼ばれる XML ベースのプロトコルを使用した通信が主流である。

この、XML ベースであるプロトコルを使用している事に起因する利点、欠点を以下に述べる。

4.3.1 Web サービスの利点

HTTP プロトコルで XML テキストデータをやりとりすることで通信するため、HTTP での接続が可能、かつ、SOAP プロトコルを解釈できれば、あらゆるデバイス間での通信が可能であることが、利点としてあげられる。プログラム間の通信手段としては CORBA や IIOP、COM+などを挙げることもできるが、CORBA 以外はプラットフォーム依存性が高く、SOAP に比べ汎用性が低い。CORBA も HTTP 上のプロトコルではないため、SOAP の方がよりインターネット上の汎用性が高い。（HTTP サーバのみで、通信可能）

これは、標準的なビジネスアプリケーションをインターネットを通じて簡単に統合できることを意味している。

インターネットのみならず、イントラネット内での利用も十分考えられる。これは、異種プラットフォーム間の連携の場合に特に有効である。

4.3.2 Web サービスの欠点

CORBA 等の、バイナリレベルの通信と比較すると、データ転送量が大きくなる。（常に XML テキストデータを送受信するため）また、プログラム的な観点では、オブジェクトそのものを渡すことができないため、処理は全てサーバ側（サービスの提供側）で行うことになる。（つまり分散オブジェクト技術ではない）

従って、実行速度が厳しく求められるような処理が多数存在するシステムの構築には不向きである。

4.4 Web サービス試作の概要

今回の試作では、以下の点の確認を目的とした。

① Microsoft.NET での Web サービス（サーバー側）に必要な作業
② Microsoft.NET での Web サービス利用（クライアント側）に必要な作業
③ Web サービスを複数の環境で利用するのに必要な作業

③を確認するため、Windows プログラムと Web アプリケーションの２種類を作成し、それぞれ①で作成した Web サービスを利用するように、試作する。

なお、コーディングには基本的に VisualStudio.NET を使用した。これは、Microsoft が.NET プログラムを作成する上で、生産性の向上を目的として提供している開発ツールであるためである。

試作したプログラムの仕様は、以下の通りである。

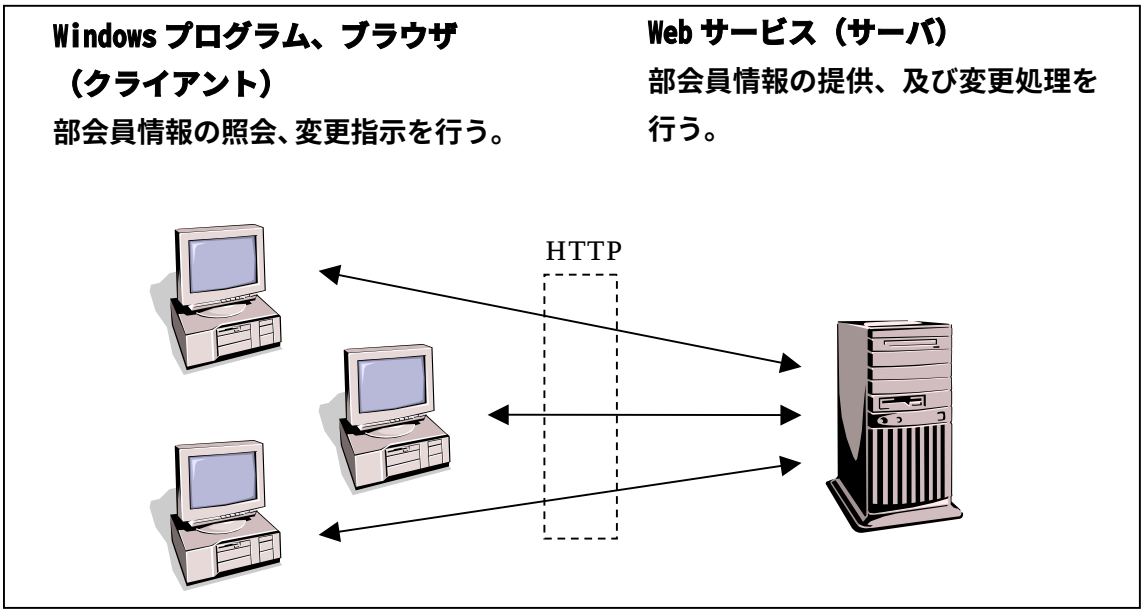
【機能概要】

OISA 部会員の名簿の管理を行う。

【機能詳細】

(1) 部会員情報の照会、登録、修正、削除ができる。
(2) 操作は、部会単位で表示された一覧表から行う。
(3) 本機能は、Windows プログラム、及びブラウザから利用でき、情報を管理するテーブルは共通である。

試作したシステムの概要イメージは、以下の通りである。



試作システムの概要イメージ

4.5 Web サービス試作の詳細

試作したシステムは、３つのサーバーブロックと、１つのクライアントブロックで構成される。各サーバーブロックは論理的に独立しているので、単一のサーバーマシンに配置しても、別々のマシンに配置してもよい。

【サーバーブロック群】

・データベースサーバー

データの永続化を担当。

・Web サービスサーバー

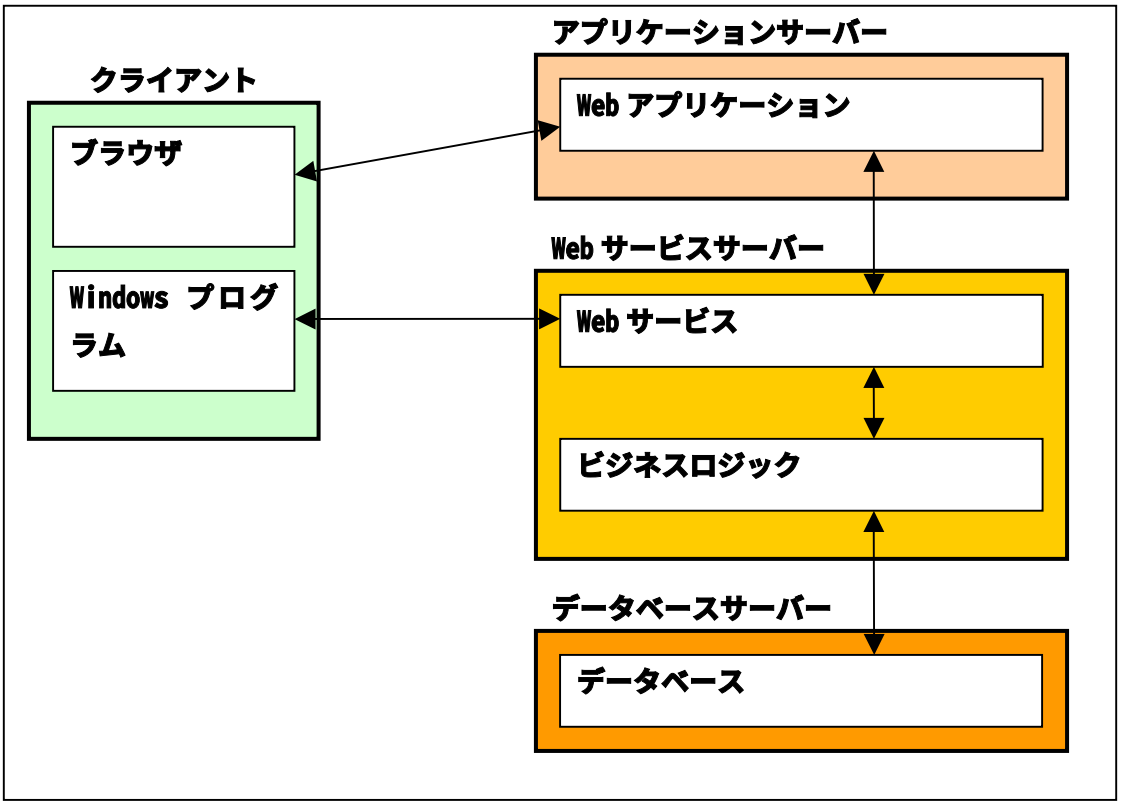
ビジネスロジックを内包し、外部からの部会員情報照会・変更要求に応じて処理サービスを提供する。

・アプリケーションサーバー

ブラウザからの要求（ブラウザ上のボタン押下等）に応答し、表示用の HTML 編集を行う。部会員情報に関わる実際の処理（取得や変更）は、Web サービスサーバーに依頼する。

【クライアントブロック】

ブラウザの場合はアプリケーションサーバーにアクセスし、部会員情報の照会・変更を行う。Windwos アプリケーションの場合は Web サービスサーバーにアクセスし、ブラウザと同様の処理を行う。



試作システムの論理ブロック構成

試作では、上記ブロックに合わせ、以下のプログラムを作成した。

① ビジネスロジック

クラス名「DBcommon」。

部会員情報の取得、更新、登録、削除等の基本操作をメソッドとして持つ。

② Web サービス

クラス名「DotNetWebService」。

DBcommon クラスを使って、部会員情報の取得、更新、登録、削除の機能を Web サービスとして公開する。

③ Web アプリケーション

クラス名「members」。

ブラウザ用の画面編集、及びブラウザからの要求に応答する。部会員情報に関わる処理は DotNetWebService クラスが提供する Web サービスを使用。

④ Windows プログラム

クラス名「frmWinClient」。

Windows 画面。部会員情報に関わる処理は DotNetWebService クラスが提供する Web サービスを使用。

なお、開発言語には Microsoft C#を使用した。

4.6 Web サービス試作の結果

試作の結果、以下の点が確認できた。

（1） Microsoft.NET での Web サービス（サーバー側）に必要な作業

Microsoft.NET プログラム開発ツールである VisualStudio.NET を使用した場合、Web サービス作成は非常に簡単である。

手順としては、

① 新規作成のテンプレートで「ASP.NET Web サービス」を選択。

② 通常のクラスを作成し、パブリックメソッドにサービスとして提供する処理を記述。

③ パブリックメソッドに WebMethod 属性をつける。

（C#では、メソッドの先頭に[WebMethod]と書けばよい）

インターネットで公開する場合は、セキュリティの設定などが必要となるが、基本的には上記の手順で作成できる。

（2） Microsoft.NET での Web サービス利用（クライアント側）に必要な作業

利用側も、非常に簡単であった。手順としては、

① VisualStudio.NET 上で、「Web 参照の追加」メニューから、利用したい Web サービスの URL を指定。これにより、Web サービス呼び出し用のプロキシクラスが自動生成。

② プロキシクラスのインスタンスを生成。プロキシクラスには、Web サービスとして公開されているメソッドが定義されているので、これを使用。

プロキシクラスが自動で作成されるため、利用側プログラムは Web サービスを、通常のクラスが持つメソッドの呼び出しと全く同じ手順で使用できる。

なお、このプロキシクラスは.NET FrameworkSDK（無償で使用できる開発環境）に付属しているツール「Wsdl.exe」を使っても作成できる。

（3）Web サービスを複数の環境で利用するのに必要な作業

Windows アプリケーションから Web サービスを呼ぶ場合と、Web アプリケーションから Web サービスを呼ぶ場合、どちらも呼び出し部分のコー

ドは全く同じになる。よって、環境の違いに起因する特別な作業は発生しなかった。（Microsoft.NET を使って開発している限りに置いては）
結果を以下にまとめる。

- Web サービスを、非常に簡単に作成することができた。
- .NET の場合、Windows プログラムと Web プログラムから同一の方法で呼び出すことができた。また、作成も簡単。
- Microsoft.NET は、Web サービスを作成するための有効な手段と言える。

5　まとめ

Microsoft.NET は Microsoft 社のネットワーク戦略から開発環境まで、幅広い意味を持つため、わかりにくいところがある。

しかしながら、今回の調査のように狭義の技術部分に焦点を当てて考えると、.NET の持つ先駆的な仕組みが見えてくる。

一般の開発現場において最も興味深いのは、.NET が今後どうなって行くかであろう。WindowsDNA のように、普及しないまま消えていくのか、それとも Java を凌ぐ勢いで普及していくのか。

.NET の強みは、Web 開発だけでなく Windows プログラムの開発基盤も取り込んでいる点と思われる。Windows GUI プログラムの RAD ツールとしては Visual Basic がデファクトスタンダードとなって久しいが、Visual Basic は.NET に取り込まれた。つまり、今後 Visual Basic で新規開発を行う場合、それは必ず.NET アプリケーションになる。

（今後数年間は、Visual Basic 6 のサポートも継続されると思われるが、保証はない。）

Visual Basic.NET を使って開発すると、今回試作した XMLWeb サービスの機能も使える。これまでスタンドアロン、またはクライアント・サーバの形態しか取れなかった環境で、ちょっとしたコードを加えるだけで Web 対応になる。こうした、既存スキル（あるいは開発環境）から Web システムへ展開していくといったアプローチが可能である点も興味深い。

.NET は発表直後、Java との比較論が非常に多く見られた。（現在もこの論調が多いが）Java は.NET に比べ、これまでの開発・運用実績が多い。これが当面の最大のアドバンテージになると思われる。また、この流れが急に変わるとは思えない。Web 開発にあたって、.Java でなく.NET を選択するとすれば、現時点では開発の容易さの１点と思われる。つまり、Java に比べ比較的容易に Web システムを構築できる点である。無論、セキュリティやパフォーマンスチューニングが必須であることは Java と変わりない。ここでいう「容易」は、主にコーディングやデバッグの容易さを指す。小規模のシステムなら、Web の画面イメージに DB 接続部品を貼り付け、少し設定してやればそれで動く。（こういった作りについては、保守の面からの議論は当然あるが、「できる」ところが.NET の特徴とも言える）

Visual Studio.NET のリリースから１年を迎え、今後 Java と.NET がどのように発展していくのか。今後の動向にも注目したい。

6　参考資料

■ .NET の定義
http://www.microsoft.com/japan/net/whatis.asp

■ MSDN Online Japan
http://www.microsoft.com/japan/msdn/default.asp

■ 特集：.NET Framework 入門
http://www.atmarkit.co.jp/fdotnet/special/dotnetframework_overview/dotnframewk_over01.html

■ Microsoft.NET Framework の自動メモリ管理 Part Ⅰ
http://www.microsoft.com/japan/msdn/net/mag00/GCI.asp

■ Microsoft.NET Framework の自動メモリ管理 Part Ⅱ
http://www.microsoft.com/japan/msdn/net/mag00/GCI2.asp

■ .NET のメモリ管理
http://www.microsoft.com/japan/msdn/library/ja/jpdnbnetguide/htm/IntroDotNET_ch02-07.asp

■ まるごと図解 最新 ドットネットがわかる
著者：片山一夫
発行：技術評論社

7　メンバー紹介

網中　誠	（部会長）	株式会社 オーイーシー
渡邉　善近	（副部会長）	大分シーイーシー株式会社
飯島　祐史		新日鉄ソリューションズ株式会社
伊野　哲哉		株式会社 富士通大分ソフトウェアラボラトリ
金崎　義徳		エスティケイテクノロジー株式会社
税所　義貴		エスティケイテクノロジー株式会社
佐藤信太郎		大銀コンピュータサービス株式会社
飯田　裕治	（技術委員）	株式会社 オーガス
阿南　光洋	（技術委員）	三井造船システム技研株式会社