# Java と.NET

## ― Web サービスにおける Java と.NET の比較 ―

平成 16 年 2 月 10 日

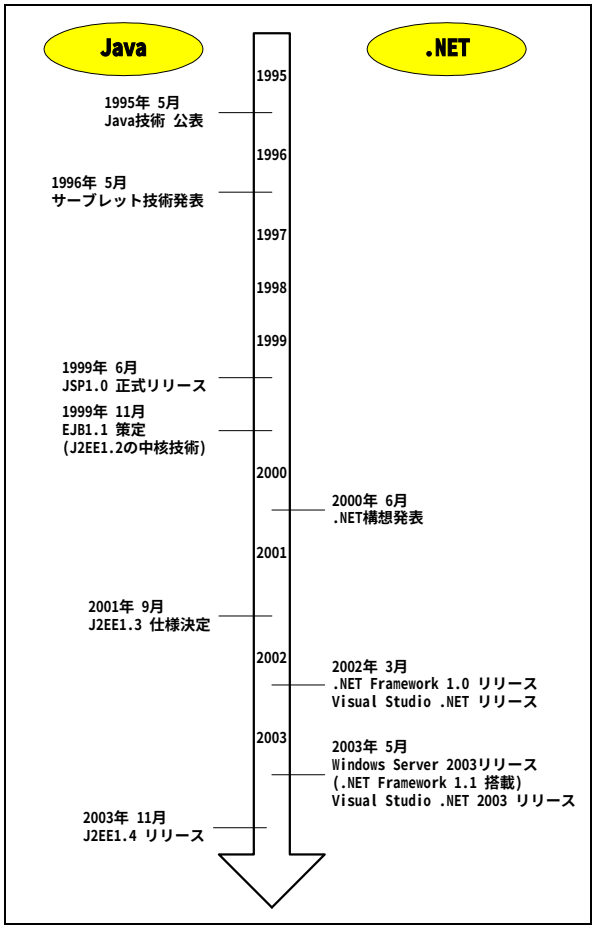
大分県情報サービス産業協会
技術研究会 Java 部会

# 目 次

## 1．はじめに

Java は 1995 年 5 月 23 日に開催された「Sun World Expo」のデベロッパーカンファレンスで公表されました。Java はプロセッサや OS の種類に関係なく、どこでも動作可能なソフトウェアという特徴をもっています。このためインターネットのマルチプラットフォーム環境に適した言語として急速に普及しました。とくにマシンや OS の違いに振り回されることもなく、ある程度の処理能力が期待できるサーバー上で動作させる技術は、企業 EC サイトのアプリケーションサーバーのロジック部分、基幹システム連携のロジック部分の開発などにおいて、デファクトスタンダードの地位を築いています。近年では携帯電話や家電製品等に対して、Kilo Virtual Machine(KVM：128K 程度のメモリで Java を動作させることが可能な Java 仮想マシン)を実装させることで、さまざまな機器に Java テクノロジーが組み込まれ、これらの機器とネットワークサーバーを接続し、ありとあらゆるデータ処理が行なわれています。

一方、2000 年 6 月マイクロソフトより次世代インターネット戦略としてドットネット(.NET)が発表されました。.NET 対応のソフトウェアやハードウェアデバイスは、ネットワーク(インターネット)を介して通信し、それぞれが分散する部品の 1 つとして連携しながらさまざまな処理を行ないます。この際の通信手段としては Web サービス(マイクロソフトは XML Web サービスと呼んでいる)と呼ばれるオープンプロトコルを使用します。Web サービスは、HTTP(Web ページのアクセスで使用)や SMTP(メール送信で使用)といった既存のトランポートプロトコルを使用し、そのうえで XML 仕様のデータをやり取りすることで、アプリケーション連携を可能にするものです。この Web サービスプロトコルを利用したアプリケーション連携の基盤となるものが、マイクロソフトが Windows OS 上に実装した.NET プラットフォームです。




**Java と.NET の歴史**

本部会では、Web サービスとして同様の性質をもつ Java と.NET について、それぞれの Web システムアーキテクチャ、および H14 年度のドットネット部会で作成された Web サービス試作と同様の試作を Java にて作成し、比較検証を行ないました。

# 2．Java について

## 2.1 概要

Java は、異機種間ネットワークによる分散環境のもとで、アプリケーション開発作業を支援する目的で設計されています。システムリソースの消費を最小限に抑えながら、ハードウェアやソフトウェアといったプラットフォーム種別を問わず実行でき、かつ動的な拡張が可能なアプリケーションを安全な形で提供することを主眼として開発されました。Java の誕生から現在に至るまで継続されるポリシーを以下に示します。

- **Simple, Object Oriented, and Familiar**

  **―― 「シンプル」「オブジェクト指向」「親和性」**

  Java は言語への親しみやすさを維持するために、C++との共通点(仕様、文法)を残すとともに、煩雑な手続き(構造体、ポインタ etc)を削除することによりシンプルな言語として開発されました。また、複雑化するネットワークベースの環境に対応するためにオブジェクト指向を取り入れました。

- **Robust and Secure**

  **―― 「安定したセキュリティ」**

  Java ではコンパイル時チェックとダイナミックチェック(ランタイム)という 2 段階のチェックを行なうことにより、エラーの起きる状況をできる限り排除し、信頼性と安全性に優れ安定したソフトウェア開発を可能としています。

- **Architecture Neutral and Portable**

  **―― 「機種非依存と移植性」**

  Java は「バイナリ・コード・フォーマット」により固有のアーキテクチャに依存しない性質をもっています。またシステムが Java インタプリタとランタイム・システムを実装していれば、コンパイル済みの Java プログラムをどのシステムに対しても簡単に移植することができます。

- **High Performance**

  **―― 「優れた性能」**

  Java はインタプリタがランタイム環境をチェックせずに、そのまま高速動作する設計を採用したこと、自動ガーベッジ・コレクタが優先順位の低いバックグラウンド・スレッドとして実行され、常時必要なメモリが確保されることから最大限の性能を得ることができます。

- **Interpreted, Threaded, and Dynamic**

  **―― 「インタプリタ形式、スレッド対応、ダイナミックチェック」**

  Java はインタプリタ型であるため、コンパイル～リンクのサイクルを実行せずにプロトタイプを作成できます。環境は動的拡張が可能であるため、処理途中のクラスでも必要に応じてロードできます。また、マルチスレッド対応であるためリアルタイム・レスポンスを得ることも可能です。

## 2.2 Java の構文

JavaはC/C++の特性を継承しつつ、これらの言語から不要な機能を効果的に取り除いた、比較的コンパクトな形で完成しました。これによりプログラマが信頼性のあるアプリケーションを開発する場合において、作業の負荷が大幅に軽減されるようになります。またC/C++等のプログラミング言語を使い慣れたプログラマの場合、わずか数週間でJavaを習得することができます。

### Javaの特徴

1. 多少の例外を除き、C/C++の基本データ型(数値型、論理型、配列型)、一般的な演算子はすべてJavaでも使用可能です。またJavaでは＋演算子を使用して文字列の連結を行なうことができます。
2. 配列はC/C++とは異なりオブジェクトとして提供されるため、種類を問わず任意の配列を宣言し割り当てることができます。配列へのアクセスはすべてチェック対象となり、添え字が配列の有効範囲を超えていると例外が出力されます。
3. 多重ループ、スイッチ構文からbreak/continueを使用する場合、ラベルを使用してループ、スイッチ構文に名前を付け、このラベルに対応したブロックに対してbreak/continueを適用できます(マルチレベル・ブレーク)。
4. Javaのメモリ割り当てモデルと自動ガーベッジ・コレクションの機能により、明示的なメモリ管理に依存する従来の方法(ポインタ、ポインタ演算、malloc、freeなど)が必要ありません。また、Javaのランタイム・システムはバックグラウンドを利用して、別のスレッドがCPUサイクルを争奪することがないタイミングで、ガーベッジ・コレクションを実行します。
5. Javaでは、言語(構文)レベルに加え、ランタイム・システムとThreadオブジェクトからのサポートによってマルチスレッド処理がサポートされます。
6. Javaでは、C/C++における冗長な機能の削除が行なわれています。以下に削除された機能を示します。
   - typedef/define/プリプロセッサ
   - 構造体(structure)/共用体(unions)
   - 関数
   - 多重継承
   - goto文
   - 演算子のオーバロード
   - 自動強制型変換(Automatic Coercions)
   - ポインタ

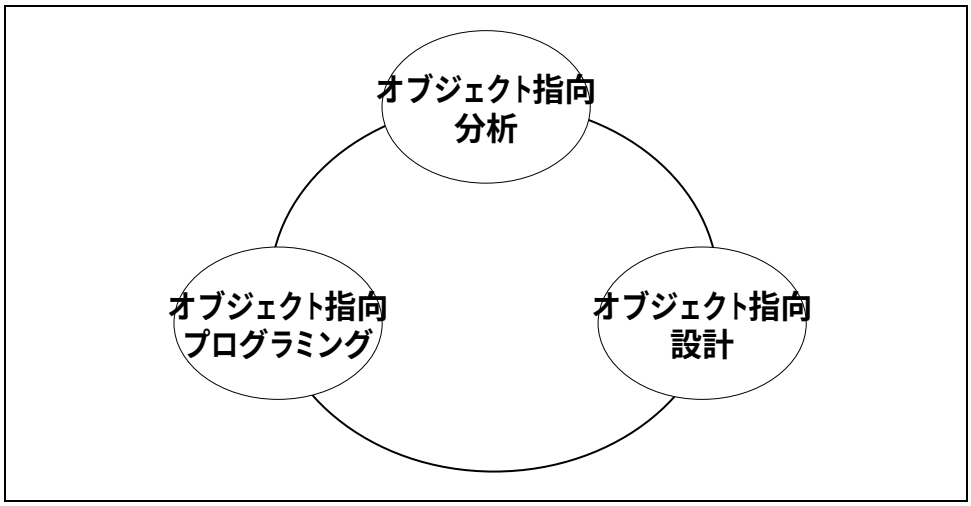
## 2.3 オブジェクト指向

### 2.3.1 オブジェクト指向とは

オブジェクト指向とは、システムを構成するすべての「もの」が「オブジェクト」として定義され作成されるという考え方、およびそれを実践するための方法論などを指す言葉です。システム構築する際には、必ずモデリングという作業が必要になります。モデリングでは、扱う対象の一部または全部を何らかの形でデータ化したうえで、さらにそれに必要な「機能」を与えるという作業が要求されます。オブジェクト指向の考え方は、こうしたモデリングを経て定義されるシステムを構成するすべての「もの」が、データと手続き(機能)を持った「オブジェクト」として定義できるというものです。単にデータと手続きを一体として記述できるのみならず、それらのオブジェクトへの外部からのアクセスを明示的に制限できる機能を持つことも大きな特徴です。これを情報隠蔽と呼びます。こうしたオブジェクトの性質は、クラスという単位で定義されますが、さらにオブジェクト指向ではこのクラスに継承という関係を定義することにより、重複するデータや手続きをまとめあげて強固な構造を定義する、記述を簡略化して再利用性を促す等を可能にしています。

**オブジェクト指向のシステム開発**

オブジェクト指向はその性質上、複雑なシステムを構築するときにこそ本来のメリットを生かすことができるものです。すなわち、オブジェクト指向をプログラミングの手段として用いるだけでは、その本来のメリットを十分に引き出せているとはいえません。オブジェクト指向プログラミング、オブジェクト指向分析、オブジェクト指向設計を組み合わせてこそ、はじめて本来のメリットが出せるものです。このような背景により、近年ではオブジェクト指向分析、オブジェクト指向設計、デザインパターンが注目されています。




**オブジェクト指向のシステム開発**

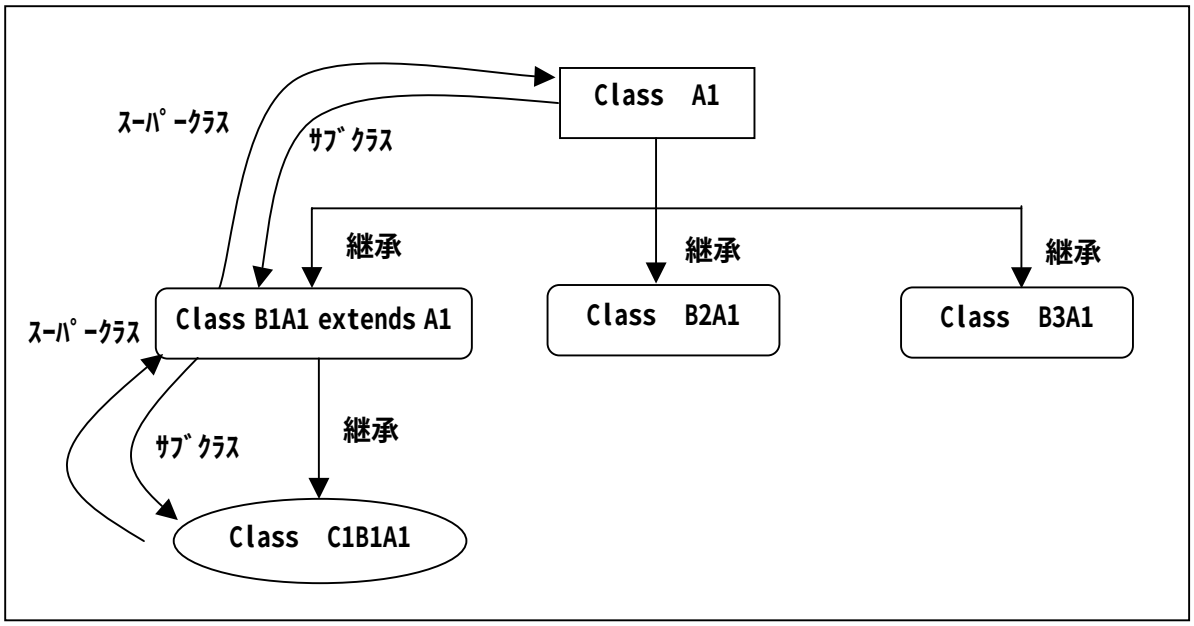
**Java 言語におけるオブジェクト指向の手法**

- カプセル化（Encapsulation）
  情報隠蔽とモジュール化を実現します。
- 多態性（Polymorphism）
  異なるオブジェクトに同じメッセージを転送することにより、メッセージを受け取るオブジェクトの特性に応じたふるまいを設定できます。
- 継承（Inheritance）
  既存のクラスに基づいて新しいクラスやふるまいを定義することにより、コードを再利用、および編成します。
- 動的バインディング（Dynamic binding）
  オブジェクトは、ネットワークを介して、事実上どの位置からでも採用できます。コードの作成時に、具体的なオブジェクトのタイプが不明であっても、オブジェクトにメッセージを転送するだけで済みます。動的バインディングによって、プログラムの実行中に最大限の柔軟性が確保されます。

### 2.3.2 クラス継承

**クラス継承**

クラス継承(クラス拡張)とは既存のクラスを流用し機能を追加する仕組みです。継承元になるクラスをスーパークラス、継承して作成される新クラスをサブクラスと呼びます。クラス継承の形式は extend キーワードを使用します。クラス継承を行なうことにより、スーパークラスのメソッドはサブクラスにてすべて定義されるため、メソッドを作成しなおすことなく利用できます。サブクラス化はすでに開発済みの既存コードだけでなく、汎用的に実用化されているコード等を利用し、変更が必要な部分だけ置き換えることが可能となるため､既存コードの再利用性を高め設計/開発/テスト作業を節約することができます。

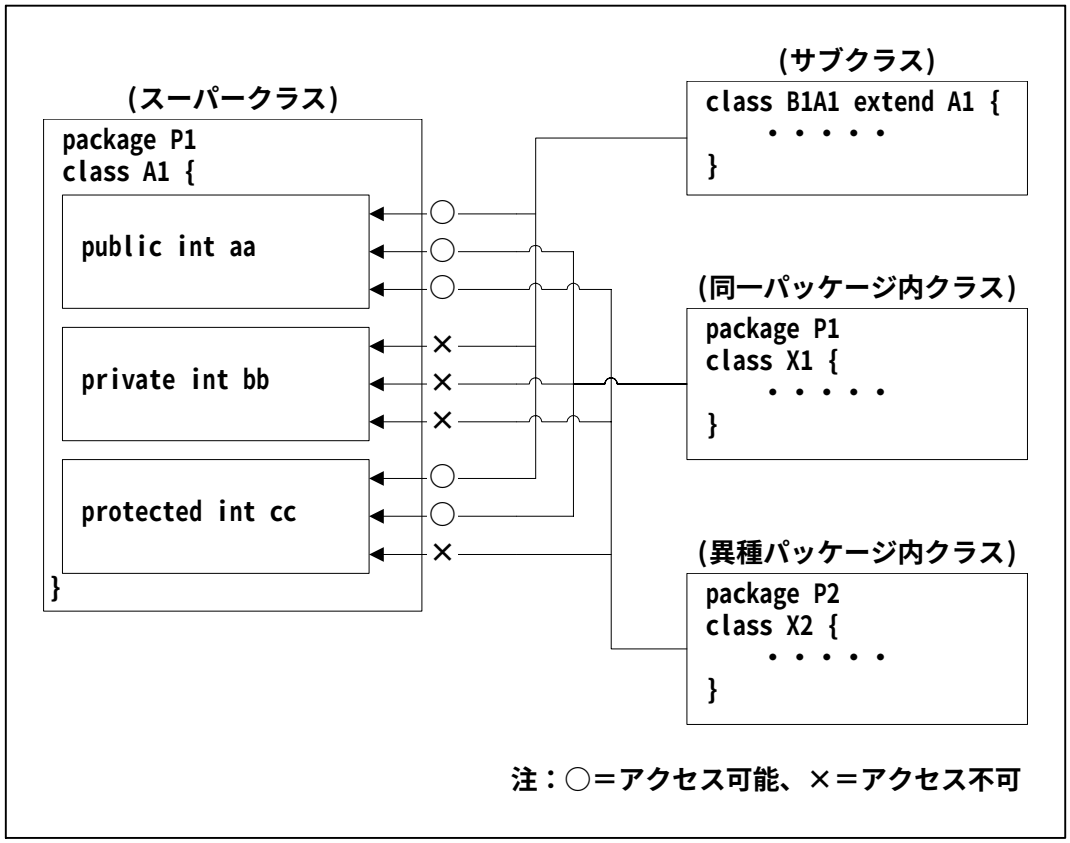



クラス継承

**アクセス制限**

サブクラスからスーパークラスのメソッドにアクセスする場合、スーパークラスのメソッドのアクセス修飾子(public/private/protected)により、アクセスが制限されます。

- public 型　・・・ 他クラスからアクセスできる
  記述が簡単であるが、安全性が低い
- private 型　・・・ 他クラス(継承関係にあるクラスを含む)からアクセスできない
  public 型とは対照的に、安全性は高いが、記述に配慮が必要
- protected 型・・・ 継承関係にあるクラスはのみアクセスできる
  サブクラス内　　：public メンバーと同様
  別パッケージのクラス：private メンバーと同様

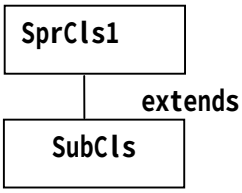



**アクセス修飾子によるアクセス制限**

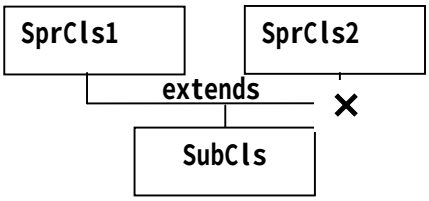
## インタフェース

Java では多重継承の機能が削除(C/C++言語の機能のうち Java では採用されていないという意)されています。したがって多重継承で必要な機能はインタフェースによって提供されます。インタフェースは抽象メソッドだけを集めた宣言を行なう機能で、定数とメソッドの書式のみ記述します。インタフェースの特徴を以下に示します。

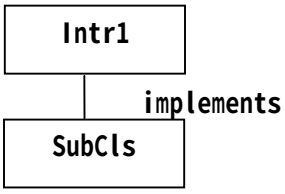
- 単独でクラスに実装できる
- 複数のインタフェースを実装することができる
- クラスの継承とインタフェースの実装を同時に指定することができる
- 直属のスーパークラスはただひとつしか持つことができない



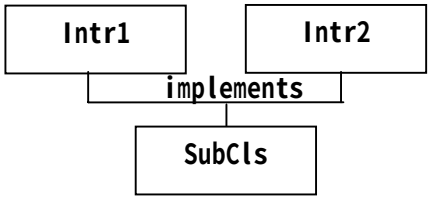

１つのクラスを継承する



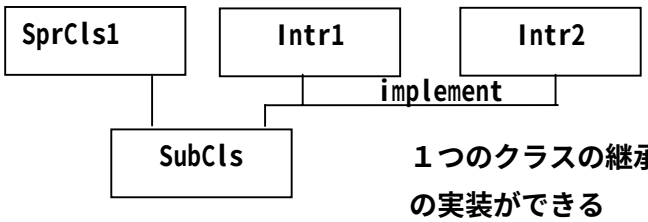

２つのクラスを継承することはできない




１つのインタフェースを実装する




２つのインタフェースを実装することができる




１つのクラスの継承と複数のインタフェースの実装ができる

### 2.3.3 パッケージ化

**パッケージ**

Java プログラムは、クラスを継承(extends)したり、インタフェースを実装(implements)したりすることで組み立てた、クラスファイルの集合です。一つのまとまりとしてアプリケーションを考えたとき、そこで利用したクラスなどを、他のアプリケーションから利用されたくないことがあります。また、クラス一つに一つのクラスファイルが作成されるので、巨大なアプリケーションでは、大量のファイルが作成され、他のアプリケーションと混ざったときに、管理が大変になります。このようなときに、クラスやインタフェースのまとまりをパッケージ(package)にして、必要があればアクセス修飾子でアクセス制限しておきます。また、既存のパッケージをインポート(import)して利用することも出来ます。パッケージの役割は、アプリケーションのクラスファイルを階層化して、クラスファイル名の衝突を避けることです。

**JavaArchive**

Java では一般に、パッケージは JAR(JavaARchive)という圧縮ファイルで用意します。JAR は ZIP フォーマットの圧縮形式であり、実行環境に含まれるコアパッケージ(便利な機能を持ったクラスを標準クラスライブラリとして実装し、それらをカテゴリごとにパッケージ化したもの)も JAR ファイルとして配布されており、多くのベンダーから、クラスライブラリが JAR ファイルとして配布されています。

**[JAR の特徴]**

- 多数のファイル(クラスファイル以外も含め)を 1 つのファイルにまとめる
- プラットフォームから独立したファイル形式
- ZIP フォーマットの圧縮形式
- パッケージを jar コマンドで圧縮することで作成・解凍することができる
- Web では JAR ファイルにするとダウンロードの速度が改善される
- アプレットなどを JAR ファイル中の個別のエントリに電子的な署名を付加することにより配布元を保証することができる

## 2.4 Java 仮想マシン(Java Virtual Machine)

一般的にプログラミング言語はインタプリタ型言語とコンパイラ型言語とに大別することができます。Java や C 言語などで書かれたソースコードはそのままでは実行することができず、コンピュータが処理できる機械語に翻訳する必要があります。インタプリタ型言語とコンパイラ型言語とはこの機械語への翻訳処理が異なります。Java はインタプリタ型言語とコンパイラ型言語の中間に位置しています。

| | コンパイラ型<br>(C/C++など) | インタプリタ型<br>(Perl など) | Java |
|---|---|---|---|
| 人間が記述するもの | ソースコード | ソースコード | ソースコード |
| 機械が解釈するもの | 実行コード | ソースコード | 中間コード |
| 実行までのプロセス | コンパイラがソースファイルを実行コードに変換し、CPU が実行コードを実行する。 | インタプリタがソースファイルを解釈し、あらかじめ用意された実行コードの呼び出しを判断して CPU に実行させる。 | コンパイラがソースファイルをバイトコードと呼ばれる中間コードに変換し、Java 仮想マシン(JVM)上で実行される。 |

**コンパイラ型、インタプリタ型と Java の違い**

### バイトコード

Java では直接機械語への翻訳は行なわず Java 仮想マシンに対する中間コードへと一括翻訳を行ないます。バイトコードとは全ての命令が 1 バイトであることから名付けられています。Java では、ソースコード(.java)をクラスファイル(.class)と呼ばれる中間ファイルへコンパイルします。インタプリタ型言語でありながらコンパイルを行なうことで、実行ファイルのサイズを小さくし、処理の高速化を実現しています。また、Java の特徴であるマルチプラットフォームについても中間コードを生成することで可能となっています。

### Java 仮想マシン(Java Virtual Machine)

Java 仮想マシン(以下、JVM)とはクラスファイルに翻訳された Java のソースコードをインタプリタ形式で実行します。逐次翻訳しながら実行するためコンパイル型言語に比べて実行速度は遅くなりますが、クラスファイルは JVM にとって機械語のような存在なので単純なインタプリタ型言語よりも大幅に実行速度が改善されます。また、JVM 自体も処理の高速化のために様々な技術を用いています。

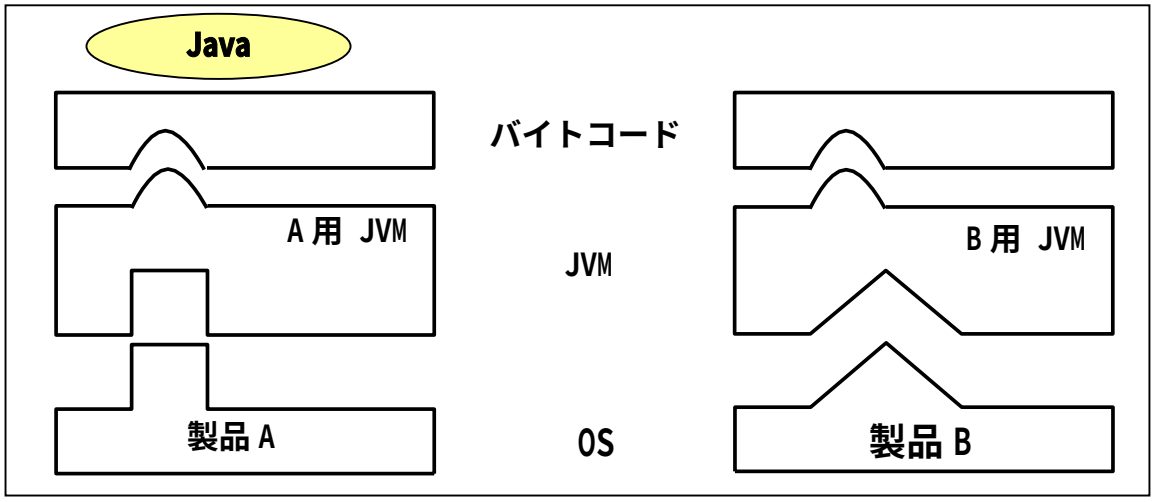
- JIT(Just-In-Time)コンパイラ …あらかじめコンパイルや処理の最適化を行っておくことで実行速度を向上させる方式
- Java Hot Spot Runtime …バイトコードを機械語に翻訳する際に複雑な処理の部分のみ最適化させてコンパイルするインタプリタ。

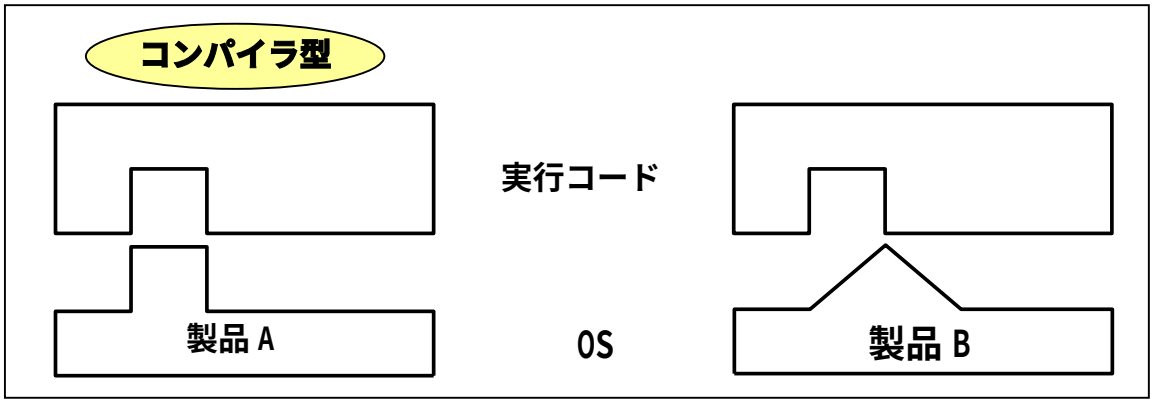
**マルチプラットフォーム性**

Java は中間コードや JVM を用いることでインタプリタ型言語の欠点である処理速度の低下を補っていますが、同時にこれは非常に大きな特徴を備えることになります。それはマルチプラットフォーム環境の実現です。通常、ソースコードの翻訳は OS に依存する形で処理されます。特にコンパイラ型言語ではその傾向が強くなります。これはたとえば、A という OS でコンパイルされた実行可能ファイルは B という OS 上では特殊なソフトウェアを使用しない限り実行することが困難になります。またプログラミング言語自身も個々の OS に最適化するため独自の記述方式を採用することもあります。従って、ソフトウェアの移植性の面では不利になってきます。

これに対して Java は JVM に対してコンパイルを行ないます。JVM への翻訳作業はどの OS 上でも違いがないので、OS 上の JVM に対しても実行作業を行なうことができます。このため開発者は OS の違いを意識することなく開発を行なうことができ、同時に移植性が高くなることで開発効率も向上します。JVM 自体は OS に依存しますが JVM は Java の開発元であるサンマイクロシステムズ社から無償で入手できるため開発者の負担は大きく軽減します。

以上のように、Java はバイトコード、JVM といった手法を使用することで、マルチプラットフォームで高速処理を実行できる開発環境を実現しています。



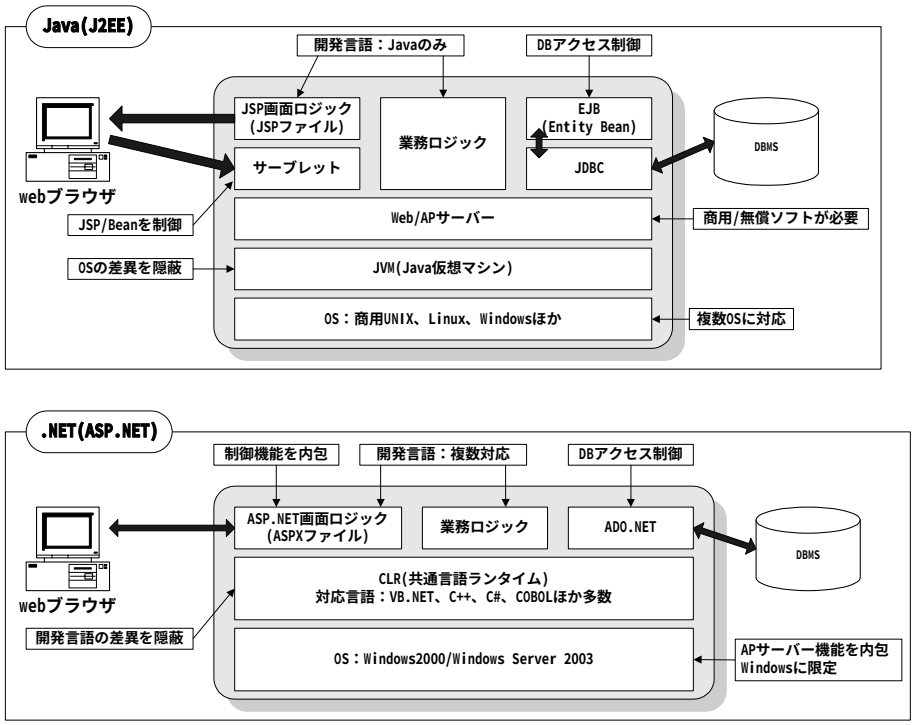



マルチプラットフォームの概念図

## 3．Web システムアーキテクチャ

前章で述べたように Java はマルチプラットフォーム環境での動作に適したプログラミング言語として急速に普及しました。Java のサーバーサイド技術であるサーブレット(1996.5)、JSP(1996.6)がリリースされてから約 7 年が経過し、Java で構築した大規模基幹システムは数多く存在しており､実績の豊富さから大規模システムなら Java というイメージが定着しています。一方、Microsoft から発表されたドットネット(2000.6)も.NET Framework(2002.3)のリリースに伴い、.NET Framework を採用した業務システムとして本格稼動が相次いで始まっています｡Java と.NET の Web システムにおけるアーキテクチャや機能は非常によく似ており、主な違いは以下の 4 つがあげられます。

1. Java の開発言語は Java に限られるが、.NET は CLR(Common Language Runtime)が言語の違いを吸収し、VB.NET や C#、COBOL などさまざまな言語が使用できます。
2. .NET では画面表示を ASPX ファイルにて行ないます。これは Java の JSP ファイルに相当するもので機能的には同じですが、Java の場合、Web ブラウザから呼び出されるのがサーブレットであるのに対して、.NET では Web ブラウザにある ASPX ファイルが自分自身を呼び出す(ポストバック)方式を取っています。
3. .NET Framework には AP サーバー機能が内包されており、別途導入する必要がありませんが、その反面、他社製品を選択する余地もありません。
4. Java は JVM(Java 仮想マシン)により､OS の違いを吸収しさまざまな OS や CPU 上で動作可能です。一方、.NET は OS が Windows 2000 または Windows Server 2003 に限定されます。




**Java と.NET による Web システムアーキテクチャの比較**

### 3.1 Java による Web システムアーキテクチャ

#### 3.1.1 サーブレット/JSP

サーブレット(Servlet)は Java プログラムによって動的にコンテンツを作成する技術です。サーブレットプログラムはさまざまな Web クライアントからの要求に応じて処理を行ない、主に HTML を作成し結果として返却します。サーブレットの場合、入力である「要求(request)」と出力である「応答(response)」は HTTP 通信の要求/応答になります。

サーブレットの特徴として

- 多くの要求を処理できるようにマルチスレッドで稼動
- HTTP 通信管理をサーブレットコンテナが行なう(開発者の負担が少なくなる)
- 純粋な Java プログラムである

Web サーバーは Web クライアントの要求を判別し、静的コンテンツの場合は Web サーバー自身が応答を返し、動的コンテンツの場合はサーブレットに要求をパスします。サーブレットはパスされた要求を処理し、応答を Web サーバーに返します。要求が静的/動的コンテンツなのかは、要求の URL で決定されます。

一方、JSP(JavaServer Pages)はサーブレットと同様に動的コンテンツを作成する技術です。サーブレットを作成するためには Java 言語を知っている必要があり、Web ページ開発者にとってサーブレットを Java で作成するのは余計な手間が掛かってしまいます。これを改善するために作られたのが JSP で、HTML に似た JSP 用タグを利用し、HTML を記述するように動的コンテンツを作成することができます。ひとつのページの中で静的コンテンツの部分は HTML で記述し、動的コンテンツの部分は JSP で記述します。JSP は単なるテキストファイルに HTML/JSP タグを記述して作成し、アプリケーションサーバーに配置するだけで、あとはアプリケーションサーバーが自動で動作を行ないます。つまり静的コンテンツの HTML 作成と同様の手順で動的コンテンツの JSP が作成可能となります。

JSP を作成するときはタグ言語で記述しますが、アプリケーションサーバー上で動作するときはサーブレットとして動作します。しかしアプリケーションサーバー内の JSP コンパイラーが JSP→サーブレット変換を自動で行なうため、JSP を利用することで Java 言語を知らなくてもサーブレットを作成することができるといえます。

| | サーブレット | JSP |
|---|---|---|
| 記述言語 | Java 言語<br>(Java プログラム内に HTML を記述) | HTML＋JSP タグ＋Java 言語<br>(スクリプトレット) |
| 開発ツール | J2SDK または Java 統合開発ツール | JSP 対応ホームページ作成ツール |
| 修正作業 | Java プログラム修正後、再コンパイル | JSP ファイル修正のみ |
| 作成技術者 | Java 言語アプリケーション開発者 | Web コンテンツ開発者 |

**サーブレットと JSP の比較**

### 3.1.2 JDBC ドライバによるデータアクセス

JDBC とは、Java DataBase Connectivity の略で、Java からデータベースをアクセスするための API で、データベースに依存しないインタフェースでアクセスできます。JDBC の主な機能を以下に示します。

- データベースとの接続
- SQL の実行
- SQL の実行結果へのアクセス

JDBC ドライバは、実装方法により、以下の 4 種類があります。

タイプ 1: JDBC-ODBC ブリッジ

内部で、ODBC ドライバを使って、高速にアクセス

プラットフォームに依存した、ODBC ドライバが必要

タイプ 2: ネイティブ API

内部で、ネイティブ API のメソッドを使って、高速にアクセス

プラットフォームに依存した、専用ドライバが必要

タイプ 3: JDBC ネット

すべて Java で、汎用ネットワークプロトコルを使ってアクセス

プラットフォーム独立だが、ネットワークプロトコルのコネクタが必要

タイプ 4: Java ネイティブプロトコル

すべて Java で、ネイティブプロトコルを使ってアクセス

プラットフォーム独立なため、サーブレットやアプレットなどで利用できます。

JDBC ドライバは、データベースによって対応状況が異なります。

・PostgreSQL の JDBC ドライバ

PostgreSQL では、タイプ 4 の JDBC ドライバがリリースされています。

PostgreSQL のタイプ 4 の JDBC ドライバは､ダイレクトドライバと呼ばれています。

・Oracle の JDBC ドライバ

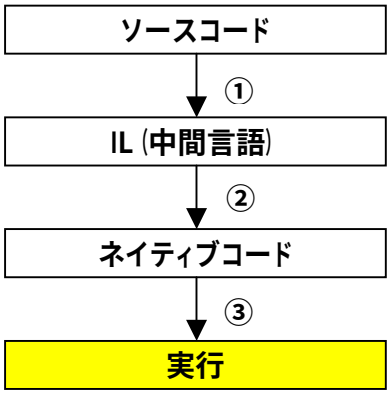
Oracle では、タイプ 2 とタイプ 4 の JDBC ドライバがリリースされています。

Oracle のタイプ 2 の JDBC ドライバは、OCI（Oracle Call Interface）ドライバ、タイプ 4 の JDBC ドライバは、thin ドライバと呼ばれています。

### 3.2 ドットネットによる Web システムアーキテクチャ

**CLR による開発言語の差異の隠蔽について**

ドットネット(以下、.NET)対応の言語で記述されたコードは共通言語ランタイム（CLR）によって実行されます。VB のランタイムと違い、CLR は.NET 対応の全ての言語に「共通」です。CLR でアプリケーションが実行されるには各言語で記述されたソースコードはネイティブコードに変換される必要があります。




**アプリケーションの実行まで**

① ソースコードを各言語のコンパイラによって中間言語（IL：Intermediate Language）に変換。
② IL は実行時に JIT コンパイラ（Just In Time コンパイラ）によってネイティブコードに変換。
③ ネイティブコードに変換されたアプリケーションは CLR で実行。

ASP.NET や Windows フォームなど、.NET アプリケーションを動作させるには CLR がインストールされている必要があります。.NET で C/S システムを開発して各クライアントにアプリケーションを配布する際は、.NET Framework 再頒布パッケージをインストールしてもらうなどして実行環境を整えてもらう必要があります。Web アプリケーションの場合はサーバーに実行環境を構築しておけばよいので、この点からも Web アプリケーションの方が導入しやすいことになります。

**Web ブラウザインタフェース「ASP．NET」について**

ASP.NET は Web アプリケーションサーバーとしての機能と、そのアプリケーションを開発・運用するための機能を併せ持っています。

**＜メリット＞**

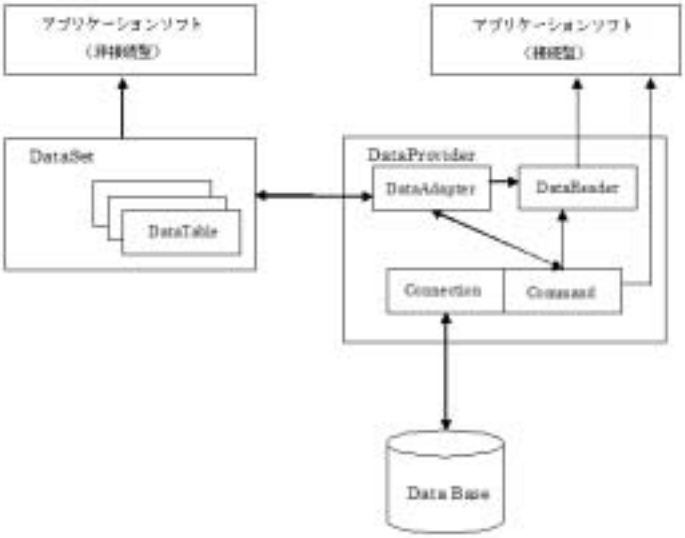
1. コンパイルが必要な言語でも、Web アプリケーションや Web サービスを実行できます。
2. ユーザーインタフェース部分とプログラム部分を別ファイルに記述できます。ユーザーインタフェース部分は「.aspx」ファイルに記述し、動的処理を行なうプログラム部分を C#で「.cs」ファイルとして記述することにより、HTML デザイナーとプログラム担当者がそれぞれの作業に専念できるようになり、開発や保守がしやすくなります。
3. 複数の言語を扱えるため、同じ内容のプログラムを別のプログラムに一から書き直す手間が減ります。プログラムを二次利用することでコストパフォーマンスの向上にもつながります。

**＜デメリット＞**

1. ASP.NET を十分に活用するには、.Net Framework についての知識も必要となります。

## データベースアクセス「ADO.NET」について

ADO.NET は ADO を更に発展させ、. NET アプリケーションの様々なデータストアとデータのやり取りを行なうことができるようになっています。ADO.NET ライブラリには、データソースに接続するためのクラスやクエリを DB に送信するクラス､結果を処理するクラスが含まれています。また、ADO との大きな違いは、ADO が接続型に対して、.NET は非接続データキャッシュを利用して、データをオフラインで処理することができるようになった点です。下記に ADO.NET 全体の仕組みを示します。




**ADO.NET の仕組み**

アプリケーションソフトが ADO.NET を利用してデータベースにアクセスする場合、Data Provider が必要な処理を行ないます。

【DataSet】非接続型レコードセット。このオブジェクトは非接続型なので、データを取り込むと、データベースとの接続を切断します。DataSet に何らかの変更があった場合、最後にもう１度接続して、データベースに変更点を書き込みます。

【DataProvider】 アプリケーションとデータとの間を取り持つ役割を果たします。Connection，Command，DataReader，DataAdapter の４つから構成されています。

【Connection】ADO の Connection と同様に、データベースとの接続を行ないます。

【Command】 SQL ステートメントを実行する仕組みで、結果セットを返す場合と、返さないものがあります。

【DataReader】接続型レコードセット。DataSet を介さずに、ユーザプログラムに直接データを送る仕組みです。

【DataAdapter】 DataSet とデータソース間のデータの取得および保存のためのブリッジとなります。DataAdapter により複数のレコードセットをコレクションする事が可能になりました。

# 4．Web サービス試作

## 4.1 目的

今回の Web サービス試作では、以下の確認を目的として作成を行ないました。

1. Java での Web サービス配信に必要な作業
2. 平成 14 年度.NET 部会が試作した Web サービスとの比較
3. システムのクラス化による作業の分担

Web サービス作成にあたり、システムを機能別にクラスとして分割し、個々に作成する形式をとり、最終的に各クラスを集合させ 1 つのシステムを構成する形で、Java の特長を確認しました。

## 4.2 環境

コーディングには、Java2 SDK Standard Edition を使用しました。これは、Sun Microsystems 社が無償で提供しているツールです。HTTP サーバーには、Java との連携に優れている Apache + Tomcat を使用しました。これらも無償で提供されています。データベースは、.NET 部会が使用したデータベースを利用します。

| | |
|---|---|
| OS | Windows 2000 |
| 開発ツール | Java2 SDK, Standard Edition Version 1.4.1 |
| Web サーバー | Apache 2.0.45 |
| サーブレット・コンテナ | Tomcat 4.1.12 |
| データベース | SQL Server (MSDE) |

**Web サービス構築環境**

## 4.3 仕様

【機能概要】

OISA 部会員の名簿を管理します

【機能詳細】

(1) 部会員情報の照会、登録、修正、削除ができます

(2) 操作は、部会単位で表示された一覧表から行います

※平成 14 年度.NET 部会が作成した仕様に基づきます

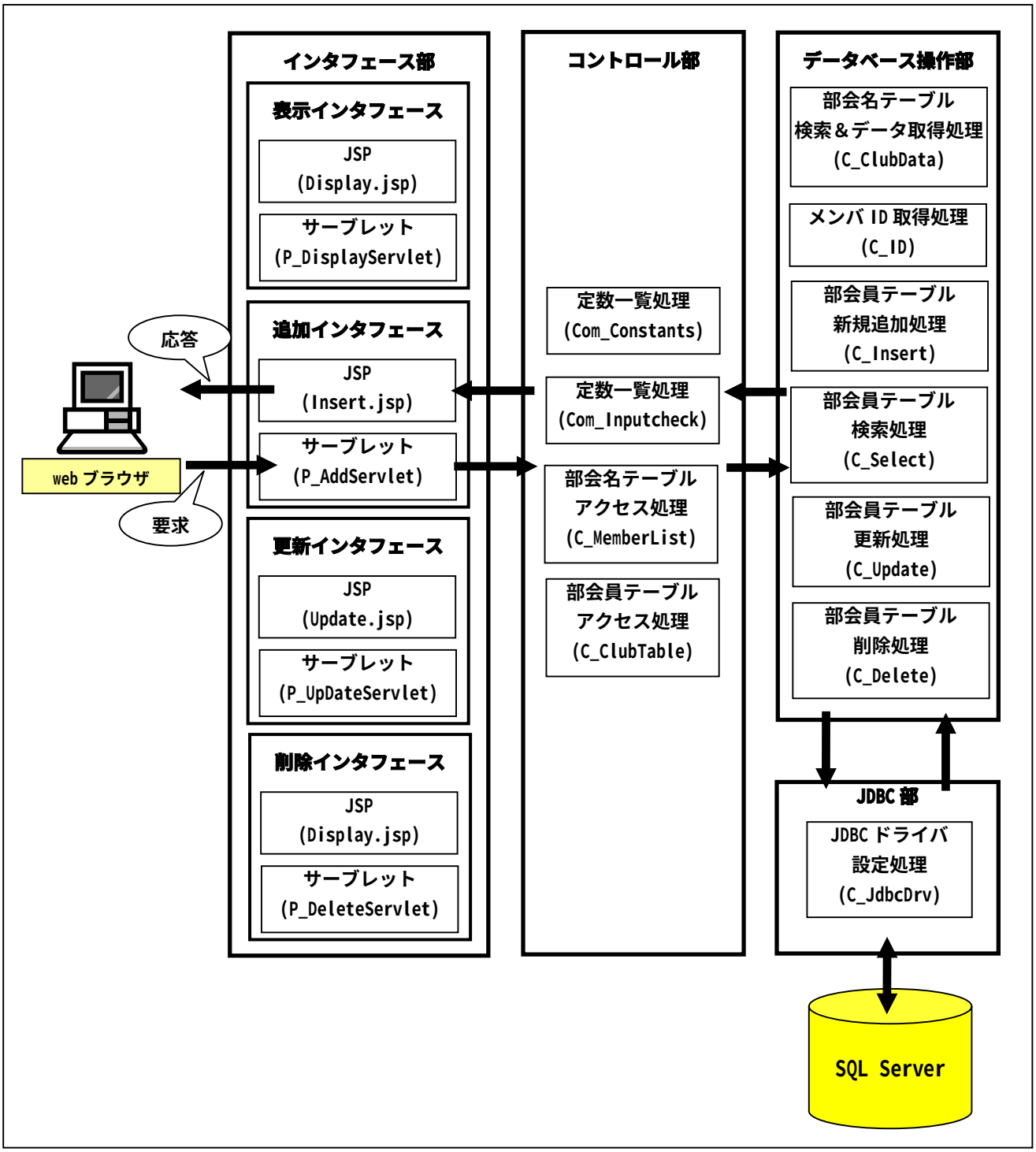
### 4.4 機能詳細

#### 4.4.1 画面イメージ

試作システムは表示画面(初期画面)/新規追加画面、更新画面の 3 つの画面にて構成されています。以下に今回の Web 試作システム画面イメージを示します。

### 4.4.2 システム構成

試作システムはインタフェース/コントロール/データベース操作部の３つのパートに分かれ、さらに各パートで機能別にクラスとして分割した形で構成されます。以下に今回の Web 試作システム概略図を示します(括弧はクラス名を表示)。




システム概略図

### 4.5 結果

試作の結果、以下を確認することができました。

#### 1. Web サービス提供に必要な作業

サーバー上で Java の Web サービスを行なう場合、開発ツールである Java2 SDK 以外に Web サーバーおよびサーブレット・コンテナの配置(セットアップ)が必要となります。試作では、Web サーバーに「Apache2」、サーブレット・コンテナには「Tomcat4.1」を使用しました(どちらも無償)。Web サーバーやサーブレット・コンテナは複数のベンダーから提供されており、またそれぞれに異なるバージョンのものが存在するので、OS の種類も含めるとそれらの組み合わせは膨大な数になります。不慣れな人にとっては、戸惑う部分が多く、分かりづらいものがあると言えます。

#### 2. .NET 試作品との比較

Java と.NET の試作をそれぞれ使用してみましたが、操作方法および処理能力において顕著な違いは見られませんでした。これは今回作成したシステムの規模が小さいことが影響していると考えられますが、今回の比較ではどちらのプラットフォームを使用しても同等の結果が得られることが分かりました。

#### 3. システムのクラス化による作業分担

Java はシステムを機能ごとにクラス化し、クラス毎に作業を分担できることから作業効率に優れていると言えます。クラス間の連携を取るためにインタフェース(クラス名)と機能(クラスのメソッド)の仕様が決定すれば、誰がどこで作成しても動作させることができることを確認しました。ただし、仕様が明確に決定されていない場合は、逆に作業が難航するという短所もあります。また、表示部分を JSP、内部処理をサーブレットで作成することで、開発/保守が行ないやすいと言えます。

## 5．まとめ

これまで説明してきたように Java はオブジェクト指向言語であり、実行時バイトコードにコンパイルして実行します。これらの特長を持つことで開発時には分散開発を容易にしコードの再利用性を高めています。また、マルチプラットフォームを実現しているため、開発環境に依存されず資源の再利用性や移植性を高めることができます。インタプリタ方式の欠点は JVM に様々な高速化技術を組み込むことで解消し、Java 本来の優位性を最大限に活用するように工夫されています。

Web サービスの開発環境としての.NET との比較においては、基本的に大きな違いはありません。しかし Java は大規模な Web システムの開発実績が多数残されており、言語としての信頼性、障害発生時の回避方法などにおいて一日の長があるといえます。その一方で、GUI を実現するには、別途 GUI アプリケーション作成のスキルが必要となります。サーブレット/JSP を書けるプログラマは増加していますが、クライアント Java を使いこなせるプログラマは非常に少なく、従来の Visual Studio のスキルが生かせる Visual Studio .NET を使った開発の方が優れているといえます。

Java と.NET を比較してきましたが、両者はどちらかで完全に置き換えられるようなプラットフォームではありません。Java も.NET もアプリケーションやサービスを実装するためのプラットフォームですが、その周辺に存在するツールやミドルウェアなどを含めて考えると、同様のソリューションを提供しているとは言えません。一般的に Java は運用実績(信頼性)が高い、.NET は開発が容易であるという利点があります。Java に関して言えば、この先 JSF(Java Server Faces)のリリースが控えており、対応する開発ツールが Sun Microsystems 等から提供され、Visual Studio .NET と同様のスタイルでのプログラミング作成が可能になると思われます。しかしこれらのツールが Visual Studio .NET と同様の完成度を得るためには、.NET が Java と同様の信頼を得るのと同じように時間が必要となります。したがって、現状 Web システムを構築する場合は、ユーザーのニーズに応じて自由に選択されるべきであり、必要に応じて共存利用されるものであると考えます。たとえば

> **基幹系業務に求められる信頼性と既存システムとの連携を考慮し、サーバー側は Java(J2EE)で、ユーザーインタフェースは使いやすさを考慮し、.NET で Windows アプリケーションを作成しクライアント側に配置する。**

といった Java と.NET の連携も考えられます。Java と.NET をつなぐインタフェースとしては、両者が業界標準プロトコルの SOAP(Simple Object Access Protocol)に対応しているため、プラットフォームの違いを意識することなくデータ連携を行なうことが可能となります(現状は、標準仕様にしたがって SOAP エンジンの実装が行なわれているとしても、仕様の実装がそれぞれに違うため「ゆらぎ」とも言うべき違いが生じています)。

Web システムの構築(提供/利用)において、プラットフォームが Java なのか、.NET なのかを選ぶことはあまり重要ではなく、適材適所で混在させ、それらをどのようにして連携させていくのかが重要であると言えます。Java と.NET が別々の方向に発展するのか、それとも共存する方向に発展するのかは、今の段階では明確になっていませんが、Java と.NET の連携が当たり前となる時代が訪れることを期待して、今後の動向に注目していきたいと考えます。

# 参考資料

## Java 関連資料


- Sun Microsystems
  http://java.sun.com
- White Paper The Java Language Environment: Contents
  http://java.sun.com/docs/white/langenv/
- Java 2 SDK, Standard Edition ドキュメント、
  http://java.sun.com/j2se/1.4/ja/docs/ja/index.html
- Just Arks 未来を語る - Java・XHTML・Linux
  http://pcweb.mycom.co.jp/special/2001/justarks/
- JDBC について
  http://ash.jp/db/jdbc.htm
- 「まるごと図解 サーバーサイド Java がわかる」
  著者：藤田 一郎
  発行：技術評論社
- 「新 Java 言語入門」
  著者：林 晴比古
  発行：SOFT BANK


## .NET 関連資料


- 初めての Microsoft .NET
  http://www.atmarkit.co.jp/fdotnet/basics/msdotnet/msdotnet_01.html
- C/S モデルから ASP.NET への移行
  http://asp.dataweb.ne.jp/contents/aspnet/
- 平成 14 年度 ドットネット部会研究論文
  「ＣＬＲの研究とＷｅｂサービスの試作」
  http://www.oisa.jp/14ronbun/dotnet/dotnetH14.html
- 「ドットネットによる Web システムアーキテクチャ」
  平成 15 年度 ドットネット部会調査資料

# メンバー紹介および活動報告

## メンバー紹介

| | | |
|---|---|---|
| 橋本 貴之 | (部会長) | エスティケイテクノロジー(株) |
| 姫野 和憲 | (副部会長) | (株)富士通大分ソフトウェアラボラトリ |
| 手島 尚之 | | 九州東芝エンジニアリング(株) |
| 森 宗美 | | エスティケイテクノロジー(株) |
| 秦 純一 | | (株)エイビス |
| 黒田 智博 | | 新日鉄ソリューションズ(株) |
| 武田 慎一朗 | | 新日鉄ソリューションズ(株) |
| 阿野山 丈尚 | | 中津コンピュータカレッジ |
| 阿南 光洋 | (技術委員) | 三井造船システム技研(株) |
| 薮田 真司 | (技術委員) | (株)アール・シー・シー |

## 活動経緯

第 1 回：2003/07/15
- 全体説明会
- 部会長、副部会長選出、活動内容、スケジュールについて議論

第 2 回：2003/08/06
- 活動内容、スケジュールの決定

第 3 回：2003/08/25
- Java の特徴について調査報告(1)

第 4 回：2003/09/09
- Java の特徴について調査報告(2)

第 5 回：2003/10/08
- Java による Web システムアーキテクチャについて調査報告

第 6 回：2003/11/12
- Web サービス試作(名簿管理アプリケーション)決定、設計検討

第 7 回：2003/12/10
- Web サービス試作(名簿管理アプリケーション)の設計検討
- H14 年度ドットネット部会作成の Web サービス試作のデモ動作
- 論文の構成、担当の決定

第 8 回：2004/01/14
- 論文ラフ案の議論、修正

第 9 回：2004/01/21
- 論文ラフ案の修正、発表用資料の議論

第 10 回：2004/02/04
- 論文、発表用資料の完成

# 名簿管理システム 設計資料

## 1. データベースレイアウト

【部会員テーブル(MemberList)】

| 項目名 | フィールド名 | 型 | 備考 |
|---|---|---|---|
| ID | ID | int 型 | 主キー |
| 部会名 ID | ClubID | int 型 | |
| 氏名 | Name | char(20)型 | |
| 会社名 | Company | char(60)型 | |
| 所属部署名 | SectionName | char(60)型 | |
| 電話番号 | Tel | char(12)型 | |
| FAX 番号 | Fax | char(12)型 | |
| E-mail アドレス | Email | char(50)型 | |
| 年齢 | Age | int 型 | |
| 区分（役職） | Part | char(20)型 | |

【部会名テーブル(ClubTable)】

| 項目名 | フィールド名 | 型 | 備考 |
|---|---|---|---|
| 部会名 ID | ClubID | int 型 | キー |
| 部会名 | ClubName | char(30)型 | 1：Java 部会<br>2：ドットネット部会<br>3：セキュリティ部会 |

## 2. クラス一覧

| パート | パッケージ名 | クラス名 | 説明 |
|---|---|---|---|
| インタフェース部 | P_Display | P_DisplayServlet | 名簿表示処理 |
| | P_ADD | P_AddServlet | 名簿追加処理 |
| | P_UpDate | P_UpDateServlet | 名簿更新処理 |
| | P_Delete | P_DeleteServlet | 名簿削除処理 |
| コントロール部 | P_Constants | Com_Constants | 定数一覧 |
| | P_Util | Com_Inputcheck | 入力値チェック処理 |
| | P_Parameter | C_MemberList | 部会名テーブルメンバ アクセス処理 |
| | | C_ClubTable | 部会員テーブルメンバ アクセス処理 |
| データベース操作部 | P_ExecSQL | C_ClubData | 部会名テーブル 検索＆データ取得処理 |
| | | C_ID | ID 自動取得処理 |
| | | C_Select | 部会員テーブル 検索処理 |
| | | C_Insert | 部会員テーブル 新規追加処理 |
| | | C_Update | 部会員テーブル 更新処理 |
| | | C_Delete | 部会員テーブル 削除処理 |
| JDBC 部 | P_JdbcDrv | C_JdbcDrv | データベースアクセスドライバ処理 |

# 3. メソッド一覧

## 3.1 インタフェース部

### 3.1.1 P_DisplayServlet

| | | | |
|---|---|---|---|
| ﾒｿｯﾄﾞ | void init(ServletConfig config) | | |
| 機能 | 初期化処理 | | |
| 引数 | <I> ServletConfig (config) | 戻り値 | なし |

| | | | |
|---|---|---|---|
| ﾒｿｯﾄﾞ | void doGet(HttpServletRequest req, HttpServletResponse res) | | |
| 機能 | doPost を Call する。 | | |
| 引数 | <I> HttpServletRequest (Request)<br><I> HttpServletResponse (Response) | 戻り値 | なし |

| | | | |
|---|---|---|---|
| ﾒｿｯﾄﾞ | void doPost (HttpServletRequest req, HttpServletResponse res) | | |
| 機能 | 表示画面からのイベントを取得し、表示処理を行なう。 | | |
| 引数 | <I> HttpServletRequest (Request)<br><I> HttpServletResponse (Response) | 戻り値 | なし |

| | | | |
|---|---|---|---|
| ﾒｿｯﾄﾞ | boolean meiboDisplay(Hashtable hResult) | | |
| 機能 | 取得したデータの整合性をチェックし、JSP に応答を返す | | |
| 引数 | <I> Hashtable (ハッシュ) | 戻り値 | boolean true:成功<br>false:失敗 |

### 3.1.2 P_AddServlet

| | | | |
|---|---|---|---|
| ﾒｿｯﾄﾞ | void init(ServletConfig config) | | |
| 機能 | 初期化処理 | | |
| 引数 | <I> ServletConfig (config) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void doGet(HttpServletRequest req, HttpServletResponse res) | | |
|---|---|---|---|
| 機能 | doPost を Call する。 | | |
| 引数 | <I> HttpServletRequest (Request)<br><I> HttpServletResponse (Response) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void doPost (HttpServletRequest req, HttpServletResponse res) | | |
|---|---|---|---|
| 機能 | 新規追加画面からのイベントを取得し、新規追加処理を行なう。 | | |
| 引数 | <I> HttpServletRequest (Request)<br><I> HttpServletResponse (Response) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | boolean meiboInsert(Hashtable hResult) | | |
|---|---|---|---|
| 機能 | 取得したデータの整合性チェックを行ない、データベースへ登録する。 | | |
| 引数 | <I> Hashtable (ハッシュ) | 戻り値 | boolean true:成功<br>false:失敗 |

| ﾒｿｯﾄﾞ | Hashtable makeHashData(HttpServletRequest pReq) | | |
|---|---|---|---|
| 機能 | 画面からの各項目データをハッシュテーブルへ格納する。 | | |
| 引数 | <I> HttpServletRequest (リクエスト) | 戻り値 | Hashtable |

| | | | |
|---|---|---|---|
| ﾒｿｯﾄﾞ | void evalPage(String pPageName,<br>HttpServletRequest pRequest,<br>HttpServletResponse pResponse,<br>boolean include_flag ) | | |
| 機能 | 指定した URL へ include/forward する。 | | |
| 引数 | <I>String<br>(呼び出すページの名前)<br><I>HttpServletRequest<br>(リクエスト)<br><I>HttpServletResponse<br>(レスポンス)<br><I>boolean ( true:include,<br>false:foward) | 戻り値 | なし |

### 3.1.3 P_UpDateServlet

| | | | |
|---|---|---|---|
| ﾒｿｯﾄﾞ | void init(ServletConfig config) | | |
| 機能 | 初期化処理 | | |
| 引数 | <I> ServletConfig (config) | 戻り値 | なし |

| | | | |
|---|---|---|---|
| ﾒｿｯﾄﾞ | void doGet(HttpServletRequest req, HttpServletResponse res) | | |
| 機能 | doPost を Call する。 | | |
| 引数 | <I> HttpServletRequest<br>(Request)<br><I> HttpServletResponse<br>(Response) | 戻り値 | なし |

| | | | |
|---|---|---|---|
| ﾒｿｯﾄﾞ | void doPost (HttpServletRequest req, HttpServletResponse res) | | |
| 機能 | 変更画面からのイベントを取得し、変更処理を行なう。 | | |
| 引数 | <I> HttpServletRequest<br>(Request)<br><I> HttpServletResponse<br>(Response) | 戻り値 | なし |

| メソッド | boolean meiboUpdate(Hashtable hResult) | | |
|---|---|---|---|
| 機能 | 取得したデータの整合性チェックを行ない、データベースへ登録する。 | | |
| 引数 | <I> Hashtable (ハッシュ) | 戻り値 | boolean true:成功<br>false:失敗 |

| メソッド | Hashtable makeHashData(HttpServletRequest pReq) | | |
|---|---|---|---|
| 機能 | 画面からの各項目データをハッシュテーブルへ格納する。 | | |
| 引数 | <I> HttpServletRequest<br>(リクエスト) | 戻り値 | Hashtable |

| メソッド | void evalPage(String pPageName,<br>HttpServletRequest pRequest,<br>HttpServletResponse pResponse,<br>boolean include_flag ) | | |
|---|---|---|---|
| 機能 | 指定した URL へ include/forward する。 | | |
| 引数 | <I>String<br>(呼び出すページの名前)<br><I>HttpServletRequest<br>(リクエスト)<br><I>HttpServletResponse<br>(レスポンス)<br><I>boolean ( true:include,<br>false:foward) | 戻り値 | なし |

### 3.1.4 P_DeleteServlet

| メソッド | void init(ServletConfig config) | | |
|---|---|---|---|
| 機能 | イニシャル処理 | | |
| 引数 | <I> ServletConfig (config) | 戻り値 | なし |

| メソッド | void doGet(HttpServletRequest req, HttpServletResponse res) | | |
|---|---|---|---|
| 機能 | doProcess を Call する。 | | |
| 引数 | <I> HttpServletRequest<br>(Request)<br><I> HttpServletResponse<br>(Response) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void doPost (HttpServletRequest req, HttpServletResponse res) | | |
|---|---|---|---|
| 機能 | doProcess を Call する。 | | |
| 引数 | <I> HttpServletRequest (Request)<br><I> HttpServletResponse (Response) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void doProcess(HttpServletRequest req, HttpServletResponse res) | | |
|---|---|---|---|
| 機能 | リクエストを取得し削除処理を実行。画面表示クラスを call する。 | | |
| 引数 | <I> HttpServletRequest (Request)<br><I> HttpServletResponse (Response) | 戻り値 | |

| ﾒｿｯﾄﾞ | void destroy() | | |
|---|---|---|---|
| 機能 | サーブレットの後処理。 | | |
| 引数 | なし | 戻り値 | なし |

## 3.2 コントロール部

### 3.2.1 Com_Constants

定数一覧のためメソッドはない

### 3.2.2 Com_Inputcheck

| ﾒｿｯﾄﾞ | int getColumnLength(String strMoji) | | |
|---|---|---|---|
| 機能 | 文字列の文字数を返す。 | | |
| 引数 | <I> String (文字列) | 戻り値 | int 文字数 |

| ﾒｿｯﾄﾞ | boolean blnChkEmpty(String strMoji) | | |
|---|---|---|---|
| 機能 | 文字列長がゼロならば false を返す。 | | |
| 引数 | <I> String (文字列) | 戻り値 | boolean true:文字列長>0<br>false:文字列長=0 |

| ﾒｿｯﾄﾞ | String sanitize_Txt( String strTmp ) | | |
|---|---|---|---|
| 機能 | HTML 中のタグではない通常のテキスト部分にデータを埋め込む場合のサニタイジング。 | | |
| 引数 | <I> String (チェック対象文字列) | 戻り値 | String チェック後の文字列 |

### 3.2.3 C_MemberList

| ﾒｿｯﾄﾞ | C_MemberList() | | |
|---|---|---|---|
| 機能 | コンストラクタ | | |
| 引数 | なし | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setID( int val ) | | |
|---|---|---|---|
| 機能 | ID の設定 | | |
| 引数 | <I>int (ID) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setClubID( int val ) | | |
|---|---|---|---|
| 機能 | 部会名 ID の設定 | | |
| 引数 | <I> int ( 部会名 ID ) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setName( String val ) | | |
|---|---|---|---|
| 機能 | 参加者氏名の設定 | | |
| 引数 | <I>String (参加者氏名) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setCompany( String val ) | | |
|---|---|---|---|
| 機能 | 会社名の設定 | | |
| 引数 | <I>String (会社名) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setSection( String val ) | | |
|---|---|---|---|
| 機能 | 所属部署の設定 | | |
| 引数 | <I>String (所属部署名) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setTel( String val ) | | |
|---|---|---|---|
| 機能 | 電話番号の設定 | | |
| 引数 | <I>String (電話番号) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setFax( String val ) | | |
|---|---|---|---|
| 機能 | FAX 番号の設定 | | |
| 引数 | <I>String (FAX 番号) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setEmail( String val ) | | |
|---|---|---|---|
| 機能 | E-mail アドレスの設定 | | |
| 引数 | <I>String (E-mail アドレス) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setAge( int val ) | | |
|---|---|---|---|
| 機能 | 年齢の取得 | | |
| 引数 | <I)>int (年齢) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setPart( String val ) | | |
|---|---|---|---|
| 機能 | 区分の設定 | | |
| 引数 | <I>String (区分) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | int getID() | | |
|---|---|---|---|
| 機能 | ID の取得 | | |
| 引数 | なし | 戻り値 | int ID |

| ﾒｿｯﾄﾞ | int getClubID() | | |
|---|---|---|---|
| 機能 | 部会名 ID の取得 | | |
| 引数 | なし | 戻り値 | int 部会名 ID |

| ﾒｿｯﾄﾞ | String getName() | | |
|---|---|---|---|
| 機能 | 参加者氏名の取得 | | |
| 引数 | なし | 戻り値 | String 参加者氏名 |

| ﾒｿｯﾄﾞ | String getCompany() | | |
|---|---|---|---|
| 機能 | 会社名の取得 | | |
| 引数 | なし | 戻り値 | String 会社名 |

| ﾒｿｯﾄﾞ | String getSection() | | |
|---|---|---|---|
| 機能 | 所属部署の取得 | | |
| 引数 | なし | 戻り値 | String 所属部署名 |

| ﾒｿｯﾄﾞ | String getTel() | | |
|---|---|---|---|
| 機能 | 電話番号の取得 | | |
| 引数 | なし | 戻り値 | String 電話番号 |

| ﾒｿｯﾄﾞ | String getFax() | | |
|---|---|---|---|
| 機能 | FAX 番号の取得 | | |
| 引数 | なし | 戻り値 | String FAX 番号 |

| ﾒｿｯﾄﾞ | String getEmail() | | |
|---|---|---|---|
| 機能 | E-mail アドレスの取得 | | |
| 引数 | なし | 戻り値 | String E-mail アドレス |

| ﾒｿｯﾄﾞ | int getAge() | | |
|---|---|---|---|
| 機能 | 年齢の取得 | | |
| 引数 | なし | 戻り値 | int 年齢 |

| ﾒｿｯﾄﾞ | String getPart() | | |
|---|---|---|---|
| 機能 | 区分の取得 | | |
| 引数 | なし | 戻り値 | String 区分 |

### 3.2.4 C_ClubTable

| ﾒｿｯﾄﾞ | C_ClubTable() | | |
|---|---|---|---|
| 機能 | コンストラクタ | | |
| 引数 | なし | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setClubID( int val ) | | |
|---|---|---|---|
| 機能 | 部会名 ID の設定 | | |
| 引数 | <I>int (部会名 ID) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void setClubName( String val ) | | |
|---|---|---|---|
| 機能 | 部会名の設定 | | |
| 引数 | <I>String (部会名) | 戻り値 | なし |

| ﾒｿｯﾄﾞ | int getClubID( ) | | |
|---|---|---|---|
| 機能 | 部会名 ID の取得 | | |
| 引数 | なし | 戻り値 | int 部会名 ID |

| ﾒｿｯﾄﾞ | String getClubName( ) | | |
|---|---|---|---|
| 機能 | 部会名の取得 | | |
| 引数 | なし | 戻り値 | String 部会名 |

## 3.3 データベース操作部

### 3.3.1 C_ClubData

| ﾒｿｯﾄﾞ | String getClubName( int clubid ) | | |
|---|---|---|---|
| 機能 | 部会名 ID から部会名を取得する | | |
| 引数 | <I>int (部会名 ID) | 戻り値 | String 部会名 |

| ﾒｿｯﾄﾞ | Boolen getClubName( C_ClubTable[] ct ) | | |
|---|---|---|---|
| 機能 | 登録部会名と ID を取得する | | |
| 引数 | <O>C_ClubTable[]<br>(クラブテーブル) | 戻り値 | Boolen True：成功<br>False：失敗 |

| ﾒｿｯﾄﾞ | int getDataCnt( ) | | |
|---|---|---|---|
| 機能 | ID 数 を取得する | | |
| 引数 | なし | 戻り値 | int ID 数 |

### 3.3.2 C_ID

| ﾒｿｯﾄﾞ | C_ID() | | |
|---|---|---|---|
| 機能 | コンストラクタ | | |
| 引数 | なし | 戻り値 | なし |

| ﾒｿｯﾄﾞ | int getID() | | |
|---|---|---|---|
| 機能 | ID の自動取得。空いている ID 番号を取得します。 | | |
| 引数 | なし | 戻り値 | int ID |

### 3.3.3 C_Select

| ﾒｿｯﾄﾞ | C_Select | | |
|---|---|---|---|
| 機能 | コンストラクタ | | |
| 引数 | なし | 戻り値 | なし |

| ﾒｿｯﾄﾞ | boolean execSQL( int clubid, C_MemberList[] parm ) | | |
|---|---|---|---|
| 機能 | 指定した部会の登録メンバー全てを取得します | | |
| 引数 | <I>int (部会名 ID)<br><O> C_MemberList[]<br>(取得した登録ﾒﾝﾊﾞｰのﾃﾞｰﾀ) | 戻り値 | Boolen True：成功<br>False：失敗 |

| ﾒｿｯﾄﾞ | int getDataCnt( int clubid ) | | |
|---|---|---|---|
| 機能 | 指定した部会の登録メンバーの人数を取得します | | |
| 引数 | <I>int (部会名 ID) | 戻り値 | int 登録メンバーの人数 |

### 3.3.4 C_Insert

| ﾒｿｯﾄﾞ | C_Insert() | | |
|---|---|---|---|
| 機能 | コンストラクタ | | |
| 引数 | なし | 戻り値 | なし |

| ﾒｿｯﾄﾞ | boolean execSQL( C_MemberList parm ) | | |
|---|---|---|---|
| 機能 | メンバーデータの新規登録処理 | | |
| 引数 | <I>C_MemberList<br>(新規登録するデータ) | 戻り値 | Boolen True：成功<br>False：失敗 |

### 3.3.5 C_Update

| ﾒｿｯﾄﾞ | C_Update | | |
|---|---|---|---|
| 機能 | コンストラクタ | | |
| 引数 | なし | 戻り値 | なし |

| ﾒｿｯﾄﾞ | boolean execSQL( C_MemberList parm ) | | |
|---|---|---|---|
| 機能 | 登録メンバーのデータを更新します | | |
| 引数 | <I> C_MemberList<br>(更新する登録ﾒﾝﾊﾞｰのﾃﾞｰﾀ) | 戻り値 | Boolen True：成功<br>False：失敗 |

### 3.3.6 C_Delete

| ﾒｿｯﾄﾞ | C_Delete() | | |
|---|---|---|---|
| 機能 | コンストラクタ | | |
| 引数 | なし | 戻り値 | なし |

| ﾒｿｯﾄﾞ | boolean execSQL( int id ) | | |
|---|---|---|---|
| 機能 | 登録メンバーデータを削除する | | |
| 引数 | <I>int (部会名 ID) | 戻り値 | Boolen True：成功<br>False：失敗 |

## 3.4 JDBC 部

### 3.4.1 C_JdbcDrv

| ﾒｿｯﾄﾞ | boolean OpenDB() | | |
|---|---|---|---|
| 機能 | データベース接続処理 | | |
| 引数 | なし | 戻り値 | Boolen True：成功<br>False：失敗 |

| ﾒｿｯﾄﾞ | void CloseDB() | | |
|---|---|---|---|
| 機能 | データベース切断処理 | | |
| 引数 | なし | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void CloseST() | | |
|---|---|---|---|
| 機能 | Statement オブジェクト終了処理 | | |
| 引数 | なし | 戻り値 | なし |

| ﾒｿｯﾄﾞ | void CloseRS() | | |
|---|---|---|---|
| 機能 | ResultSet オブジェクト終了処理 | | |
| 引数 | なし | 戻り値 | なし |

| ﾒｿｯﾄﾞ | boolean ExecQuery( String ExecSQL ) | | |
|---|---|---|---|
| 機能 | ResultSet を返す SQL 文(Select)実行処理 | | |
| 引数 | <I> String (SQL 文) | 戻り値 | Boolen True：成功<br>False：失敗 |

| ﾒｿｯﾄﾞ | boolean ExecUpdate( String ExecSQL ) | | |
|---|---|---|---|
| 機能 | SQL 文(Select 以外)実行処理 | | |
| 引数 | <I> String (SQL 文) | 戻り値 | Boolen True：成功<br>False：失敗 |

| ﾒｿｯﾄﾞ | ResultSet getDbResult() | | |
|---|---|---|---|
| 機能 | ResultSet オブジェクト外部提供処理 | | |
| 引数 | なし | 戻り値 | ResultSet<br>(ExecQuery()の返値結果) |