新潟工科大学

情報電子工学科

1999 年度卒業研究報告書

テーマ：分散オブジェクト環境 CORBA の性能評価

研究者	学籍番号	氏　名
	9612031	小島　勇治
	9612053	高井　彰士

指導教員　　青山 幹雄

提出日　平成 12 年 2 月 4 日 (金)

分散オブジェクト環境CORBA の性能評価

小島勇治　　高井彰士

Evaluation of CORBA Distributed Object Environment

Yuji Kojima　Akihito Takai

1章　はじめに

1.1 研究の背景

コンピュータシステムの開発は，ダウンサイジングの影響でクライアント／サーバシステムが浸透してシステムの拡張性を向上させたり開発コストを抑制できるようになったが，いくつかの問題でてきた．ネットワーク上であらゆる情報資源をどのように結ぶか，どこに必要な情報が存在するのか，１つのマシンでは限界があるとき，どうすればその負荷を他の複数のマシンに分散させてアプリケーション全体の性能を高めることができるのか．このような課題に対する分散オブジェクトコンピューティングの代表的なアプローチが分散オブジェクト環境 CORBA（Common Object Request Broker Architecture），JavaRMI（Remote Method Invocation），HORB（Hirano Object Request）であり，情報社会で大きな位置を占めるようになった．これからは，オブジェクト指向でクライアント／サーバ型アプリケーションを効率的に開発するために分散オブジェクト環境が必要になる．CORBA のような分散オブジェクト技術は，オブジェクト指向の考え方を分散コンピューティングに適用することで必要な情報の共有を行える環境を提供する．

1.2 研究の目的

分散オブジェクトコンピューティングでは，多くの種類の OS（オペレーティングシステム）や様々なプログラム言語が使用されているために環境に対する適応性が必要になる．そのなかのデータ転送を取り扱う上で基本的な型が，それぞれの OS によって示すデータの内容が違っているために，データを正しく転送できない．この問題に対処するために CORBA では，IDL（Interface Definition Language）を用いて分散オブジェクト環境上で多様なプログラム言語の型を取り扱えるように規定している．しかし，この利点を持つ CORBA を使用した研究や実用はまだ少ない．そこで，本研究では CORBA を用いてデータを転送するときに，性能の観点からどのデータ型を利用するべきか比較し検討することを目的とする．

2 章　分散オブジェクト環境

2.1　分散システム

分散システムとは，複数のマシンである処理することによって，１つのマシンで処理するときよりもかかる負担を少なくすることで処理を効率良く行おうとするもので，ネットワークを意識せずに１台のコンピュータのような操作を行えるように実現ことが必要になる．

システムを分散化する理由として，ある問題をいくつかの小さな問題に分解することで安価なコンピュータを使用すれば，大型コンピュータに頼らなくても計算処理を並列に行えること，大きなデータを移動させるのは困難であり，ユーザはデータサーバから必要な情報を提供してもらうことでデータのある場所で制御や管理や共有を行えること，複数のコンピュータに同様のデータを持たせているので１つのコンピュータで障害が起きても他のコンピュータで処理を続けることが可能なことが挙げられる．

分散アプリケーションを開発するために多くの標準化が進み, 最近では, CORBA や RMI によって構造化されたデータをネットワークを介して転送するプロトコルを提供できる．さらに，Java 言語と組み合わせることで，低レベルのネットワーク通信から，分散されたオブジェクトまで様々なレベルの分散コンピューティングの開発が可能である．

分散システムの特徴を利用して分散アプリケーションを開発するとき以下のことが大切である．

1) データと機能を分割して分散させるときに，最適な配置を行う．
2) 柔軟で拡張可能な通信プロトコルによって，様々な形の通信が実装でき，送信される情報の型やフォーマットの変更要求に適合させる．
3) マルチスレッド処理は，CPU 時間，ローカルな記憶装置，ネットワーク帯域の使用を最適化するための効果的な方法である．そのために，マルチスレッドを作成して制御する．
4) 安全性を確保するには，アクセス元の出所を認証できるようにして，資源に対してアクセスレベルを定義して，データ転送するときに暗号化する．

2.2　オブジェクト指向

近年，オブジェクト指向でのソフトウェア開発方法が特に注目されている．その理由は，ソフトウェアの機能が高度化／複雑化してプログラムの難易度が高くなっているためである．いままでは，機能詳細化（functional decomposition）という機能に基づいて分解する方法がとられていた．しかし，この機能詳細化では，現在の複雑化／高度化した問題を扱うのには適切でない．そこで，複雑な問題をオブジェクトに分解していくオブジェクト指向という考え方を用いる．オブジェクト指向とは全対象をオブジェクトに分解し，オブ

ジェクト間の関係を明確にすることで複雑な問題をシステマティックに取り扱う方法論である．また，今後の分散オブジェクト環境上でのソフトウェア開発において，特に重要になってくるのは Java 言語との関連性である．本格的なオブジェクト指向言語であり強力な言語仕様である Java 言語を用いることで，効率的なソフトウェア開発が可能となる．

オブジェクトとクラス

オブジェクト指向は「もの（オブジェクト）」に命令して処理を進めていく考え方である．

一般的なオブジェクトの定義は「オブジェクトとは，実世界に存在する“ もの ”である」である．オブジェクトは，データ（状態・情報・データ構造）と処理するためのいくつかの操作を持ち，基本的にはオブジェクトが持っているデータを他のオブジェクトからは参照できない．このデータを隠蔽することをカプセル化といい，システムの保守性や拡張性を向上するための重要な概念である．

クラスは，同じ性質を持つオブジェクトを集め，抽象化したもので，属性と操作がある．属性はオブジェクトの持つ具体的なデータ値を抽象化したもので実際の値は持たない．操作は，各オブジェクトが持っている操作を複数のオブジェクトから共通利用する．

抽象化とスーパー抽象化

抽象化とは，「ある実体の根本的な性質に着目して偶然の出来事は無視する」ことである．オブジェクトを抽象化しクラスを定義するときは，主に各オブジェクトが持っている具体的な値が除かれ，システムにとって根本的でない属性も除かれる．これに対し，スーパー抽象化とは，複数のクラスにおいて同じ性質を持つクラス同士を抽象化することを指す．即ち，共通の性質をスーパークラスに定義して，固有の性質をサブクラスとして定義する．

カプセル化

カプセル化とは，データとデータを処理するときの操作を１つにまとめて，決められたインタフェース以外の方法ではデータ操作を許可しないようにすることである．データを全て非公開にするとデータを処理する専用の操作をインタフェースとして公開すればデータ構造に変更があってもインタフェースの変更がなかったならば，変更における影響の範囲はその操作のうちだけにとどめることが可能になる．これは，システムの保守性や拡張性を保証する上で大変重要である．クラスのデータ構造を非公開にすると利用するプログラムは直接データを参照できないため，インタフェースを使ってデータにアクセスすることになる．カプセル化を行うには，インタフェースはシグニチャ（操作名，戻り値，パラメタ）を公開し，メソッド中のコーディングやデータの処理方法は隠蔽する．これは，データ構造や操作方法を知らなくてもシグニチャでデータを参照でき，データ構造や操作方法の変更もシグニチャの変更がなかったならば，プログラムのコードの変更は必要なくて済む．

インタフェース

インタフェースは，一般に使われる側のオブジェクトが使う側にアクセスするための手

段である．

パッケージ

パッケージとは，システムを細分化するときの単位であり，クラスをグループ化するときの単位である．巨大なシステム全体を理解しやすい範囲に分解し，そのサブシステムの中でクラスを定義すると他のパッケージと協調関係を結ぶときに，具体的なクラスを意識せず今見ている範囲だけ考えれば良い．似た役割を持つクラスをグループ化してパッケージとすると分業の単位となり，保守面でもクラス管理を行うときに有利となる．

オブジェクト指向システムでは，システム内のオブジェクト同士が互いに協調し合いながら 1 つのユースケースを実現する．オブジェクトを抽象化したクラス同士は必ず何らかの関係がなくてはデータの送受信などができない．クラス間には，以下の関係（関連，継承，集約，依存）が存在する．

・関連

オブジェクト同士がいつも関係があるのではなくある理由から一時的に結びついた関係で，リンクするオブジェクト同士には協調関係ができる．

・継承

スーパークラス（親クラス）とサブクラス（子クラス）の関係を継承関係という．スーパークラスとは，いくつかのクラス間の共通な性質をまとめたものである．スーパークラスの全ての性質（属性，操作，関係）をサブクラスは受け継ぐことができる．また，サブクラスでは，スーパークラスの性質を再定義や，自らの性質を追加できる．抽象クラスは，インスタンスが生成できないクラスのことである．

ポリモルフィズム

スーパークラスと同じシグニチャの操作をサブクラスにも再定義することである．同じ操作であっても異なるクラスから生成されたインスタンスでは，異なる動作をすることである．継承関係での再定義はどのクラスの操作を起動するかはクラスではなく，要求されたインスタンスが決める．インスタンスは，生成したクラスを知っているので自ら探し操作を起動する．操作の起動を行うクラスではスーパークラスのインタフェースを参照し操作を起動するだけである．

・ 集約

部分クラスは，全体クラスの 1 部である．

操作の委譲とは，全体クラスのオブジェクトに要求した処理を部分クラスのオブジェクトが処理することである．そして，集約関係の構造を隠蔽するためや共通な処理を部品化するために使要する．操作の委譲は，明示的なプログラミングが必要である．しかし，継承関係はコンパイル時に決定するが，集約関係は実行時に変更できるというメリットがある．操作の伝播とは，全体クラスへの処理要求があったときに全体クラスで処理をして，更に処理を部分クラスに伝え，部分クラスでも処理を行うことである．コンポジション集約とは，全体クラスを生成すると部分クラスが生成されて，全体クラスをなくすと部分クラスもなくなることである．

・依存
一方のクラスの内容を変えたときに片方のクラスに影響を与える関係である．

2.3 分散オブジェクト環境

分散オブジェクト環境とは WWW(World Wide Web)やクライアント／サーバシステムなどの分散ソフトウェアを分散していることを意識せずにオブジェクト指向で開発するためのミドルウェアである．一般的な分散オブジェクト環境としては，CORBA(Common Object Request Broker Architecture)，JavaRMI(Remote Method Invocation)，HORB(Hirano Object Request Broker)などがある．

分散オブジェクト環境の大きな特徴として，透過性が挙げられる．透過性とは，分散オブジェクト間連携のときに重要になってくる．以下の透過性の内容を挙げる．

1) 位置：オブジェクトの置かれた異なるマシン，プロセス
2) 言語：オブジェクトの実装言語
3) プラットフォーム：オブジェクトの置かれたマシンの OS と通信プロトコル

2.3.1 分散オブジェクト環境のアーキテクチャ

CORBA，JavaRMI，HORB に代表される分散オブジェクト環境はいずれもクライアント上のオブジェクトがサーバ上のオブジェクトにサービスを要求するクライアント／サーバ・アーキテクチャをとる．

透過性を実現するためにブローカ(Broker)とプロキシ(Proxy)の２つのアーキテクチャ／デザインパターンを組み合わせている．クライアントのブローカが位置や名前の解消を行い，目的に合ったオブジェクトのあるサーバのブローカへ要求を送る．クライアントプロキシをスタブ，サーバ側プロキシをスケルトンと呼ぶ．プロキシはクライアントとサーバでブローカとオブジェクト間のインタフェースの整合を取る．

スタブがインタフェースデータを下位の通信プロトコルのデータへ変換することをマーシャリング(marshaling)，スケルトンが送られてきた通信プロトコルのデータを復元することをアンマーシャリング(unmarshaling)と呼ぶ．図-2.1 に詳細を示す．

2.3.2 オブジェクトマネージャ

オブジェクトマネージャは, 分散オブジェクトの中核部分でスケルトンとオブジェクトリファレンスを管理する．図-2.2 においてブローカに含まれて定義されることが多い．

クライアントが新しいオブジェクトを要求するとき, オブジェクトマネージャは要求されたオブジェクトのクラスに対応するスケルトンを決めて，そのスケルトンをもとに新しいオブジェクトを生成する．そして，そのオブジェクトの参照（リファレンス）をクライアントに返す．

クライアントからサーバのメソッド呼び出しを行う場合には, オブジェクトマネージャがクライアントのリクエストをサーバの適切なオブジェクトに割り当て，結果をクライアントに返す．クライアントがリモートオブジェクトを必要としなくなると，オブジェクトマネージャにオブジェクト消滅のリクエストを出す．マネージャはサーバのオブジェクトを消滅させオブジェクトが使用していたメモリを解放する．

2.3.3

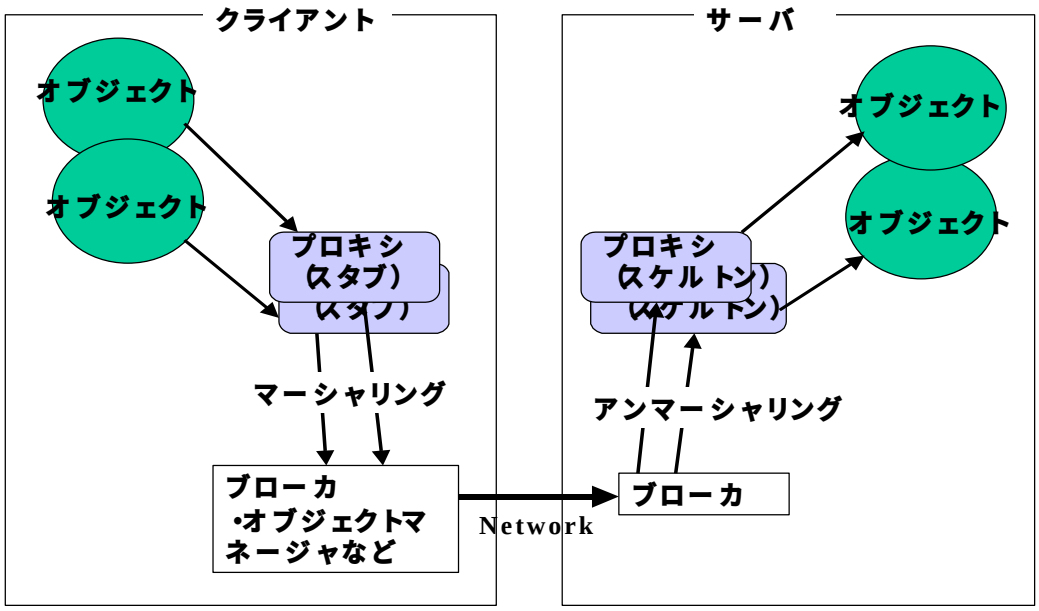


図 - 2.1 分散オブジェクト環境のアーキテクチャ

オブジェクト通信プロトコル

クライアントがリモートオブジェクトと通信を行うには, リクエスト処理のために一般的なプロトコルが必要である．少なくともオブジェクト参照，基本データ型を送受信できるプロトコルである必要がある．クライアントはローカルなスタブにアクセスするだけで, 通信部分の詳細から隠蔽されその部分を分散オブジェクト環境に任せる事が重要である．この事によって，クライアントアプリケーションが簡潔になり，リモートオブジェクトを分散していることを意識せずに使用する事ができる．

2.3.4 セキュリティ

ネットワーク上でオブジェクトを分散するのであるから，セキュリティは重要な問題である．ブローカへのリクエストを作成するクライアント，スタブはオブジェクトレジストリへのアクセスに対して認証され許可される必要があり，他の領域やオブジェクトへのアクセスは制限される．また，クライアントオブジェクトとリモートオブジェクト間の処理は，暗号化される必要がある．分散オブジェクト環境がこれらの操作をサポートすること

が必要である．

2.3.5 分散オブジェクト環境の意義

図-2.2 に示すように，分散ソフトウェアのアーキテクチャによって分散オブジェクト環境を用いたオブジェクト間連携にはいくつかのパターンがある．以下に主なシステムを挙げる．

1) 3 層クライアント／サーバ・システム
クライアント，アプリケーションサーバ，DB サーバ間のオブジェクト連携

2) Web ベースのシステム
Web クライアントとサーバ間，ならびに Web サーバとアプリケーションサーバやデータベースなどのサーバ間オブジェクト連携

このようにクライアントとアプリケーションサーバ，データベースサーバを分けることにより，データの更新はデータベースサーバを変更するだけで，アプリケーションサーバを変更しないですむ．サーバを増やすときも同様である．またビジネスロジックをアプリケーションサーバに記述するので，クライアントの軽量化（シン･クライアント）ができる．サーバが故障してもデータベースサーバとアプリケーションサーバが分離しているのでデータを失うことはなく，システムの信頼性も向上する．

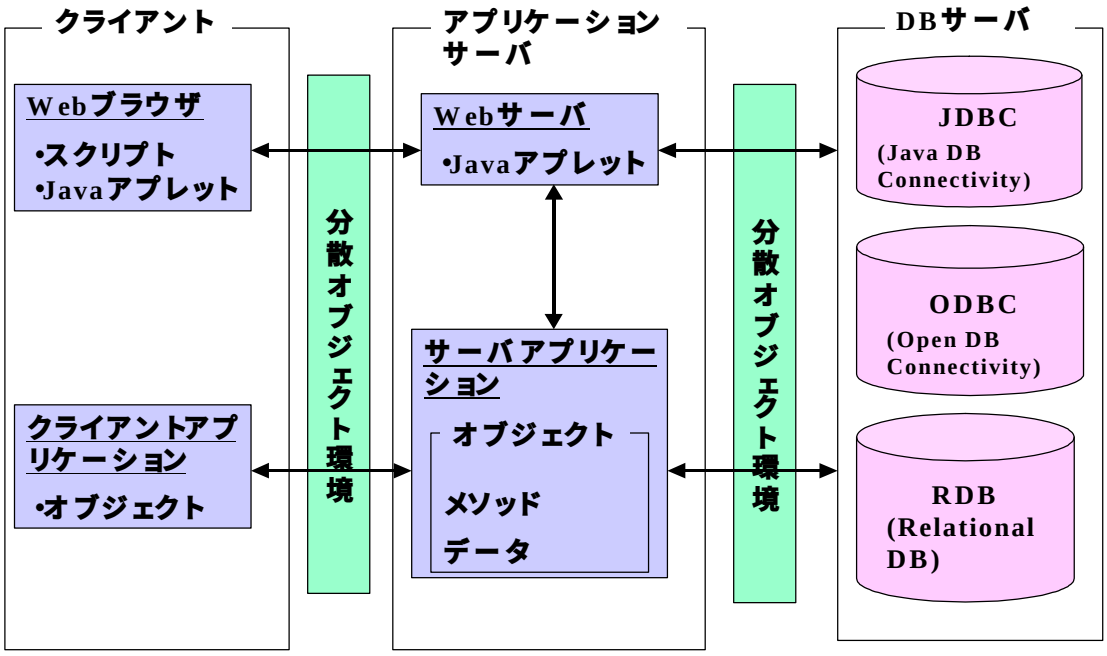


図 - 2.2 分散オブジェクト環境の役割

2.4 Java

2.4.1 Java の適用

Java 言語の利用は多くの分野に渡っている．その中で，Java プログラミングがアプレット開発に有効であるのは，Web ブラウザがＰＣやワークステイションだけでなく，ネットワークコンピュータ（NC）や「JavaStation」のようなデバイス上でも動作するからである．アプレットはローカルへの入出力が禁止されているが，セキュリティ面では利点になっている．また，プラットフォームに依存しないので，スタンドアロン・アプリケーションに有効であり，Java は多くの API を標準でサポートしている．そのため，アプリケーションの開発でライブラリやプログラミング・キッドを用意しなくてよく，API が標準化によってプログラムのポータビリティは確保されるため，移植性を高くするのが困難なサーバ・アプリケーションの開発に適している．他には，高度な Java プログラムを実行するための Java がある．最近，良く使われるようになった Java 言語の特徴を以下で述べる．

2.4.2 インタプリタ言語（Java）

Java 言語はインタプリタ言語である．インタプリタ言語は，プログラムを実行するときソフトウェアで作られた計算エンジン（インタプリタ）により実行する特徴を持つ．インタプリタという擬似的な CPU を用いて，移植性やポータビリティ，セキュリティ面で力を発揮する．

2.4.3 中間コード

Java 言語はソースコードを中間コードに変換してファイルにセーブしておく．そして，変換されたファイルから中間コードを読み込みメモリ上において，その内容に基づきプログラムを実行する．中間コードを作成するときに，文法チェックを行うためタイプミスエラーは実行前に検出できる．中間コードはソースコードよりもサイズが小さくメモリ消費を押さえ，更に最適化することで実行速度を速くできる．

2.4.4 プログラム動作の仕組み

Java 言語は，スタック型インタプリタ言語であり，スタック上にデータを置き演算をする．再帰的プログラムを実行するときには，レジスタの内容を退避をする必要が無い．

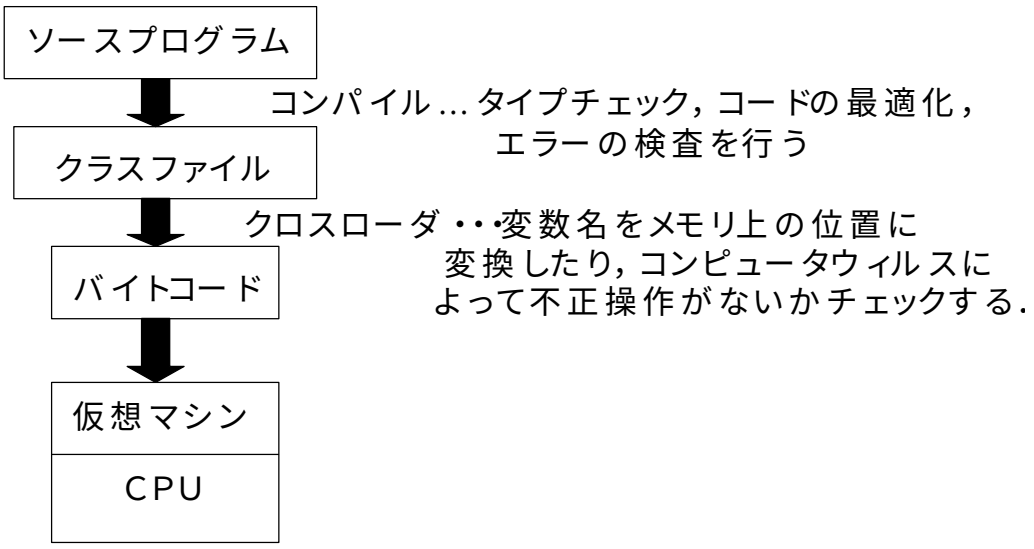


図 - 2.3　　インタプリタ型言語の動作

2.4.5 メモリの管理

Java 言語では，メモリ領域の確保の指示をするときにオブジェクトへの参照というやり方を取るので，不適切なメモリ領域を読み書きする危険は無い．そして，不要になったメモリ領域は自動的に開放するためにプログラマーが直接メモリの管理をしなくてすむ．この機能は，使用されないメモリ領域を自動的に回収するのでガーベッジコレクションという．

2.4.6 タイプチェック

ソースコードから中間コード（バイトコード）へコンパイルするときに型の不一致によるバグを防ぐことができる．これは，Ｊａｖａ言語には，タイプチェックの機能が含まれているためである．

2.4.7 例外処理

信頼性の高いソフトウェアの条件として，プログラムを実行する際に起こり得るエラーに対処できることが挙げられる．Java 言語では，例外の検出のプログラムに対する負担を軽減するために「例外処理」の概念を用いている．メソッドが投げる例外の種類をプログラムの中で宣言して，例外処理を行うようにしたり，エラーに対する処理を統一的に扱い，まとめて記述するためプログラム自体の構造が大変分かりやすくなる．

2.4.8 マルチスレッド

1 つのプログラムの中に複数の処理を同時に実行するプログラム技法のことをマルチスレッドプログラミングという．スレッドがお互いに独立して処理を実行するマルチプログラミングが行えるのは，Java 言語の特徴の 1 つである．

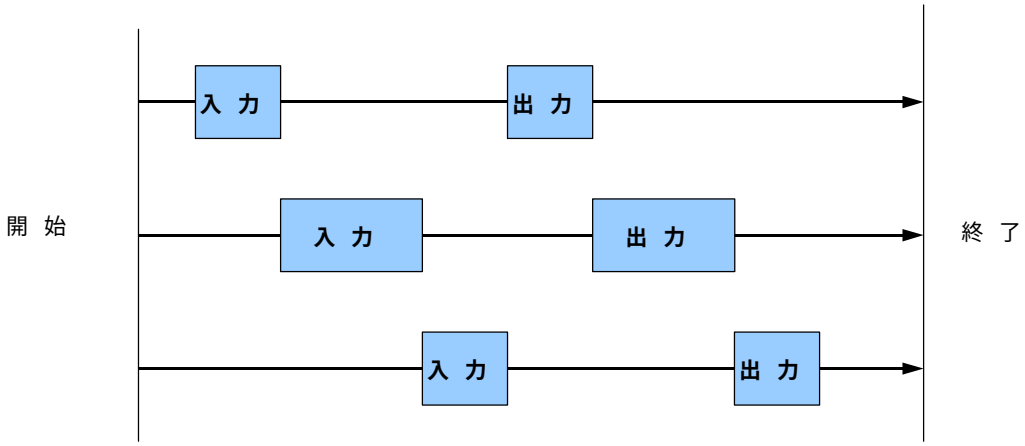


図 - 2.1 マルチスレッド

2.4.9 移植性

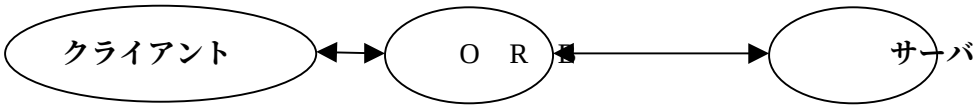
完成されたプログラムを別のオペレーティングシステムや CPU 上で動作するように修正する作業を移植といい，Java 言語はこの移植性が大変優れている．クラスファイルのフォーマットやバイトコードがオペレーティングシステムやＣＰＵに依存していないために，Java で作られたプログラムが動作する環境の上であるならコンパイルや修正しなくても実行できる．

2.4.10 実行効率

インタプリタ言語であるためにコンパイラ型言語よりも実行速度は劣る．そのための実行速度向上の方法として JIT(Just In Time compiler)を利用する方法がある．Java のインタプリタに読み込むときにバイトコードをネイティブなコードに変換する．

2.4.11 ネットワーク指向

Java 言語では，バイトコードを取り扱うので移植性のほかにも，ローカルのハードディスク上の中に存在するプログラムと同じように，ネットワーク上にある別のマシンの中のプログラムを読み込むことができる．リモートのオブジェクトをローカルのオブジェクトと同じように扱える ORB に対応したときにネットワークプログラミングの開発が効率的になる．



2.4.12 変数の種類

変数の種類	変数の説明
クラス変数	クラス定義やインタフェース定義でstaticが指定された変数で クラスで共通に使う変数
インスタンス変数	クラス定義の中でstatic画指定されていない変数でインスタンス毎に存在する
配列の構成要素	配列は名前のない変数が並んだもの
メソッドパラメータ	メソッド定義の中で宣言される
コンストラクタパラメータ	コンストラクタの定義の中で宣言される
例外ハンドラパラメータ	catchの引数として宣言される
ローカル変数	ブロック中やfor文の中での「初期化を行うところで宣言する

メモリ領域確保とメモリ領域開放

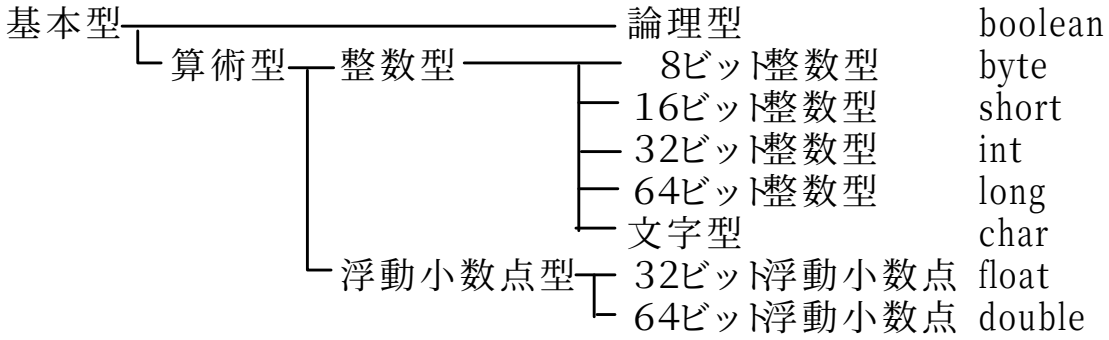
変数の種類	領域確保	領域開放
クラス変数	クラスがロードされたとき	クラスが不要になりアンロードされたとき
インスタンス変数	new演算子でインスタンスが 作り出されるとき	インスタンスがどこからも 参照されなくなったとき
配列の構成要素	new演算子で配列が作り出されるとき	配列への参照がなくなったとき
メソッドパラメータ	メソッドが呼び出されるとき	メソッドの実行終了時
コンストラクタパラメータ	コンストラクタが呼び出されたとき	コンストラクタの実行終了時
例外ハンドラパラメータ	例外が起きたとき	Catchの実行終了時
ローカル変数	プログラムの実行が宣言の部分に達したとき	変数の宣言を含む最も内側のブロックや for文の実行終了時

2.4.13 セキュリティ

Java 言語では，メモリに常駐するタイプのコンピュータウィルスを（メモリに読みこまれているオペレーティングシステムのないようを書き換えたり，実行ファイルを書き換えて潜入する）防ぐためにポインタを通してメモリの内容を直接書きかえれないようになっている．似たような文字列プログラムとしてウィルスプログラムを実行させるものも，クラスファイルを読み込むときにバイトコードのチェックによって不正なデータ変換やアクセスしてはいけないオブジェクトやフィールドにアクセスしてないか調べる．また，ネットワークから読みこんだプログラムは上書きできないようにハードディスク上の別の場所

に格納される. これらのチェックはバイトコードを使用していることが大きな意味を持つ. 他には，セキュリティマネージャによって暗証番号やパスワードの情報の漏洩をある程度防ぐことができる．このセキュリティマネージャのもとで実行すると勝手な読み書きの禁止や，プログラムを読みこんだホスト以外へのアクセスを禁止することで，勝手に情報を送信しようとすることを止めることができる.

2.4.14 基本型



論理型

true と false の２種類だけ値をとる基本型である．厳密な型のチェックを行うために，条件文では論理型以外のデータ型は指定できない.

整数型と浮動小数点型の範囲

型	サイズ	最小値	最大値
byte	8	-128	127
short	16	-32768	32767
int	32	-214783648	214783647
long	64	-9223372036854775808	9223372036854775807
char	16	¥u0000	¥uFFFF
float	32	$\pm 1.4 \times 10^{-45}$	$\pm 3.4 \times 10^{38}$
double	64	$\pm 4.9 \times 10^{-324}$	$\pm 1.7 \times 10^{308}$

2.4.15 参照型

参照型はデータを直接扱わず，データの格納場所を示す参照を扱うための型である．そ

して，参照型には，クラス型，インタフェース型，配列型の 3 種類ある．
Java 言語においてオブジェクトはクラスのインスタンスと配列の 2 種類あり，参照型の変数と格納できるオブジェクトとの関係を以下に示す．

クラス型の変数	そのクラスと子孫のクラスのインスタンスへの参照の格納
インスタンス型の変数	インタフェースを備えたクラスのインスタンスへの参照の格納
配列型の変数	その型と同じ配列への参照が格納
配列	Java.lang.Object の型の変数に代入できる
それ以外の参照型の変数	変数の型にかかわらず null を格納できる

配列

配列は同じ型の変数を複数個並べたもので，使用する前に配列をメモリ上に確保する必要があり，new 演算子を使用する．宣言においては，配列のサイズ指定しなくても良い．

2.4.16 文字列

Java 言語において文字列は String クラスのオブジェクトになるため, 文字列を他のオブジェクトと同じように操作できるし途中に NULL を含むこともできる．
1)基本データ型と同じく，プログラム中に文字列を加えることが可能である．
2)String クラスには普通のクラスで扱える= =，！=，= !といった演算子だけでなく，クラス固有の+演算子が用意されている.

String と StringBuffer

String クラスは, 固定文字列を扱うクラスで, 文字列の内容を変更できない．また, final が String クラスにはあるので子クラスを作成できない．

StringBuffer クラスは, String クラスと違い作成後に文字列の挿入や長さの変更ができる．また，final がついていないので子クラスを作成できる．

3章 CORBA

3.1. CORBA とその特徴

CORBA(Common Object Request Broker Architecture)はオブジェクト指向の分散処理環境を実現するための標準規格である．800 社を超えるメンバ企業により構成される，ソフトウェアコンソーシアムである OMG(Object Management Group)によりその仕様を制定，標準化を推進されている．その特徴として，インタフェース定義言語 IDL(Interface Definition Language)や ORB(Object Request Broker)が挙げられる．

CORBA 分散オブジェクト基盤を用いることにより，コンポーネントは，自主性，自己管理性，及び協調性に富んだものになる．単にコンポーネントを組み合わせ，拡張することで複雑なクライアント／サーバシステムを作り上げることが可能となる．

また分散環境においてクライアントはサーバの存在する位置，どのオペレーティングシステム上で動作しているか，またサーバオブジェクトの実装などを，CORBA を使うことで知る必要がなくなる(言語･位置･プラットホームの透過性)．クライアントが知るべきことはサーバオブジェクトが公開しているインタフェースであり，これがクライアントとサーバを結び付ける契約としての役割を果たすのである．このように CORBA は分散処理の透過性を，大規模な分散環境で，できるだけ多様な環境で実現しようとするものである．

3.1.1. OMG のオブジェクト管理アーキテクチャ

CORBA の前提となる規格として OMA(Object Management Architecture)がある．これは OMG が考える分散オブジェクト技術の全体像であり，構成要素は図に示すとおりである．

１）**アプリケーションインタフェース**

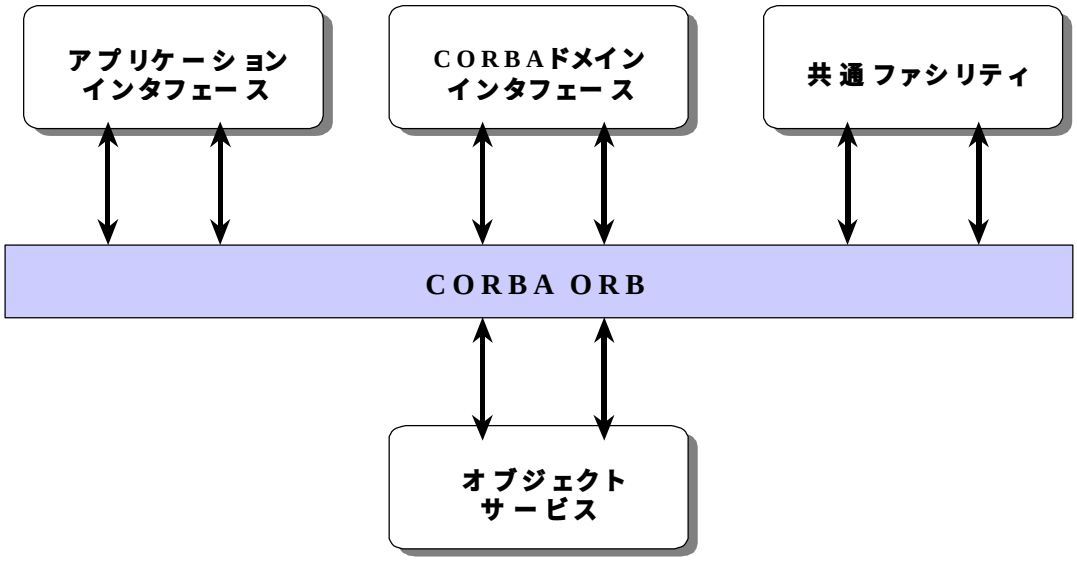


図 - 3.1　OMA の構成図

CORBA を基盤として，その他の標準化された様々なサービスを利用する，個別のアプリケーションのインタフェース.

２）CORBA ドメインインタフェース

ドメインインタフェースは，特定の産業・分野などがアプリケーションを開発するのに必要とされるサービスのインタフェースを規定する.

３）オブジェクトサービス

CORBA を利用して，分散オブジェクトシステムを構築するための基盤となるいくつかのサービス機能を提供するために，インタフェースを規定する.

４）共通ファシリティ

CORBA サービスがシステムに近いのに対し，共通ファシリティはアプリケーションに近いサービスを提供するためのインタフェースを規定している.

５）CORBA ORB

ORB(Object Request Broker)は，OMG の中心部でありオブジェクト間にクライアントとサーバの関係を確立するミドルウェアである. ORB によりクライアントオブジェクトは，同一マシン上あるいはネットワークをまたがって存在するメソッドを透過的に起動させることができる．ORB を用いることで，実行時にオブジェクトがお互いを見つけ出し，お互いのサービスを呼び出すことができる.

3.2 インタフェース定義言語IDL

分散オブジェクトシステムでは，クライアントアプリケーションとサーバオブジェクトは分離して開発，実行されるため両者間のインタフェースを一致させる必要がある．IDL（Interface Definition Language）はその両者のインタフェースを定義するものである. IDL でインタフェースを定義し，その IDL ファイルをコンパイルすることでクライアント，サーバのそれぞれの言語にマッピングされたスタブ，スケルトンと呼ばれるプロキシを生成する．このプロキシを使用することで実装言語に依存しない方法でオブジェクトを定義することができる.

IDL では，コンポーネントの属性，継承元となる親クラス，コンポーネントが発生させる例外，コンポーネントが起動する型付きのイベント，コンポーネントがサポートするメソッドなどが指定できる.

3.2.1 IDL の基本規則

IDL は大文字と小文字の区別がされ，定義の最後はセミコロン（;）で終わる．また Java

や C，C++と同様にモジュールのメンバの指定やインタフェースの定義は括弧（{ }）で括る．コメント文も Java や C，C++と同様である．

3.2.2 IDL の構造

IDL の構造として module(モジュール)がある．共通の目的を共有する IDL 定義同士をグループ化するために用いられる．

Module の例

```
1:  module Bank{
2:      interface Customer{
3:              …
4:      };
5:      interface Account{
6:              …
7:      };
8:      …
9:  };
```

3.2.3 IDL のデータ型

表‐3.2 に IDL のデータ型を示す．

3.2.4 interface 型

interface は，CORBA オブジェクトが提供するサービスを記述する．IDL の interface 型は，定義されたメソッドの実装を提供しないという点で Java の interface 型とよく似ている．また IDL の interface は属性や，オペレーションを含むことができる．

interface のメソッドは，オブジェクトの機能を定義するもので，メソッドシグネチャあるいは単にシグネチャと呼ばれるメソッド定義をする．メソッドシグネチャとは，メソッドが何をするのか，どのようなパラメータをメソッドが取るのか，そしてどのパラメータを出力として返すかを記述している．メソッドのパラメータは，in，out，inout として宣言できる．それぞれの意味として，in パラメータはメソッドへの入力として，out パラメータはメソッドからの出力として，inout パラメータはメソッドへの入力とメソッドからの出力として働く．

IDL の interface は，別の interface のメソッドや属性を継承できる．

3.2.5 exception 型

IDL は，開発者が例外を定義できるようにしている．また開発者がユーザ定義例外を作成できることに加え，CORBA は多くの標準例外，あるいはシステム例外を提供する．

CORBA と IDL は，CORBA が定義する標準例外と，ユーザ定義例外の例外処理をサポートする．

型	IDL 型	特徴	マッピングされる型

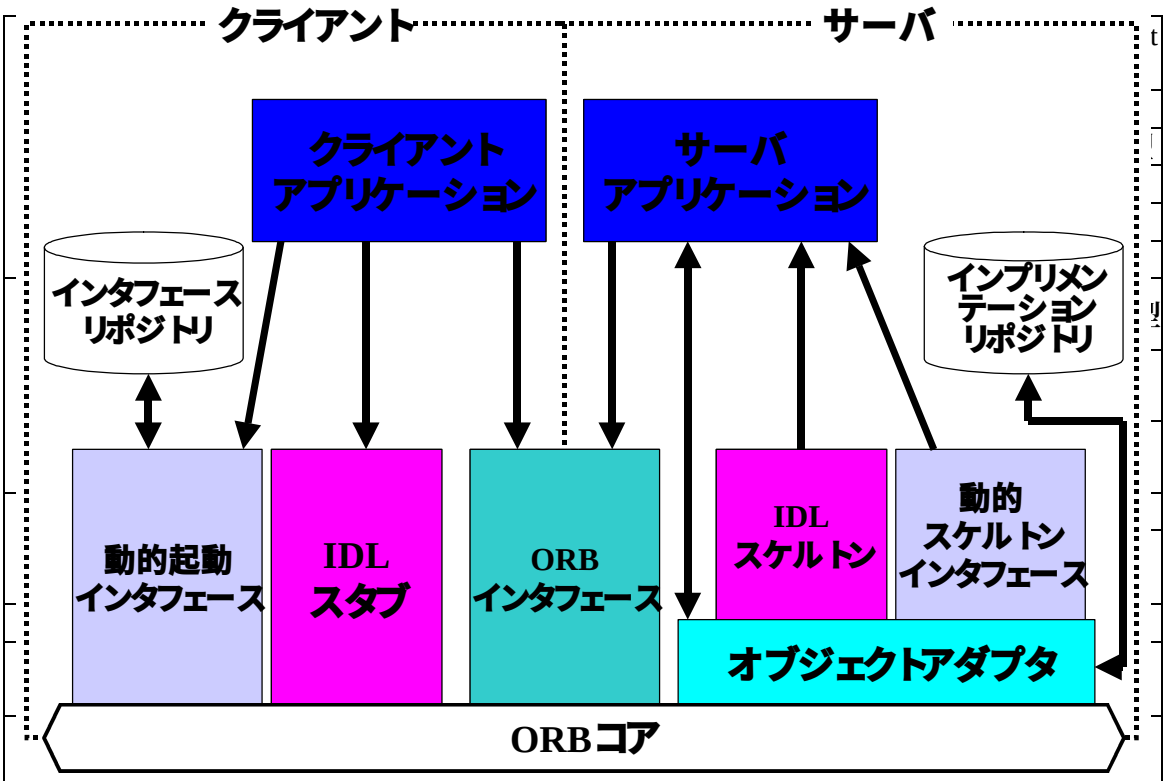


図 3.2 ORB の構造

列挙型	enum	る	
構造体型	struct		
共用型	union	異なる型の値を表す	
シーケンス型	Sequence	動的に大きさを変えられる 1 次元配列	

表 3.2 IDL のデータ型

3.2.6 他の IDL の構造

ユーザ定義の型名を生成するためにサポートされる typedef や，循環参照（2 つの interface が互いに，一方がもう一方の interface を参照する属性やメソッドを持った場合に生じる）問題の解となる前方宣言などがある．

3.3 CORBA ORB

ORB は，CORBA の基礎となるものである．ORB の主な役割は，アプリケーションコンポーネントがお互いの接続を確立できるようにし，オブジェクトリファレンスのためのリクエストを解決することである．

図 3.2 に ORB の構造を示し，次にクライアント／サーバにおける CORBA の果たす機能について述べる．

3.3.1 クライアント側から見たCORBA の機能

(1) IDL スタブ

オブジェクトサービスに対して静的インタフェースを提供．クライアントから見ると，スタブはリモート上にあるサーバオブジェクトのように振る舞う．つまりリモートにあるサーバオブジェクトの代理人を務める．スタブは，パラメータを含めたサーバオブジェクトへの操作を, サーバに送ることが可能なメッセージ形式にエンコード, またはデコードする．また，スタブ自身がヘッダファイルも保持しているため，低水準のプロトコルや，データのマーシャリングなどといった問題を気にすることなく，高水準言語を用いてサーバのメソッドを呼び出すことができる．

(2) 動的起動インタフェース

動的起動インタフェース DII（Dynamic Invocation Interface）は，実行時に起動するべきメソッドを探し出すことができる．CORBA の標準 API の定義により，サーバインタフェースを定義しているメタデータの探索，パラメータの作成，リモート呼び出しの発行，結果の返送を行うことが可能である．

(3) ORB インタフェース

アプリケーションにとって重要であると考えられるいくつかのローカルサービスの API（Application Program Interface）から構成されている．例えば，オブジェクトリファレンスを文字列に変換したり，あるいはその逆を行うための API が CORBA から提供されている．

(4) インタフェースリポジトリ

登録されているすべてのコンポーネントインタフェースと，そのコンポーネントインタフェースがサポートするメソッドおよびパラメータに関する記述（CORBA ではメソッドシグネチャと呼んでいる）を，取得したり，修正したりできる．この API によりコンポーネントはメタデータ情報の参照, 保管, 更新を動的に行うことができる．メタデータを使いこなすことによって, ORB 上のあらゆるコンポーネントは自己記述型インタフェースを備えることができる．

インタフェースリポジトリはオブジェクト定義を格納するオンラインデータサービスである．CORBA ではオブジェクト定義の情報が，リポジトリからユーザへどのようにして与えられるかを詳細に定義しており，これをリポジトリ内の情報を表現するインスタンスが属する一群のクラスを規定するという形で実現している．クラス階層は IDL 仕様を反映しているため，インタフェースリポジトリは IDL の記述と同じ形式で組織化されたオブジェクト群を管理する柔軟なオブジェクトデータベースとして成立している．

インタフェースリポジトリは，動的起動の際，メソッドシグネチャのチェック，異種 ORB 間の連携時などに必要となる．

１）インタフェースリポジトリクラス

インタフェースリポジトリはリポジトリ内の情報を表現するオブジェクトの集合として実装される．CORBA は８つの IDL 構造のそれぞれに対してインタフェースを定義している．表-3.3 にそれを示す．

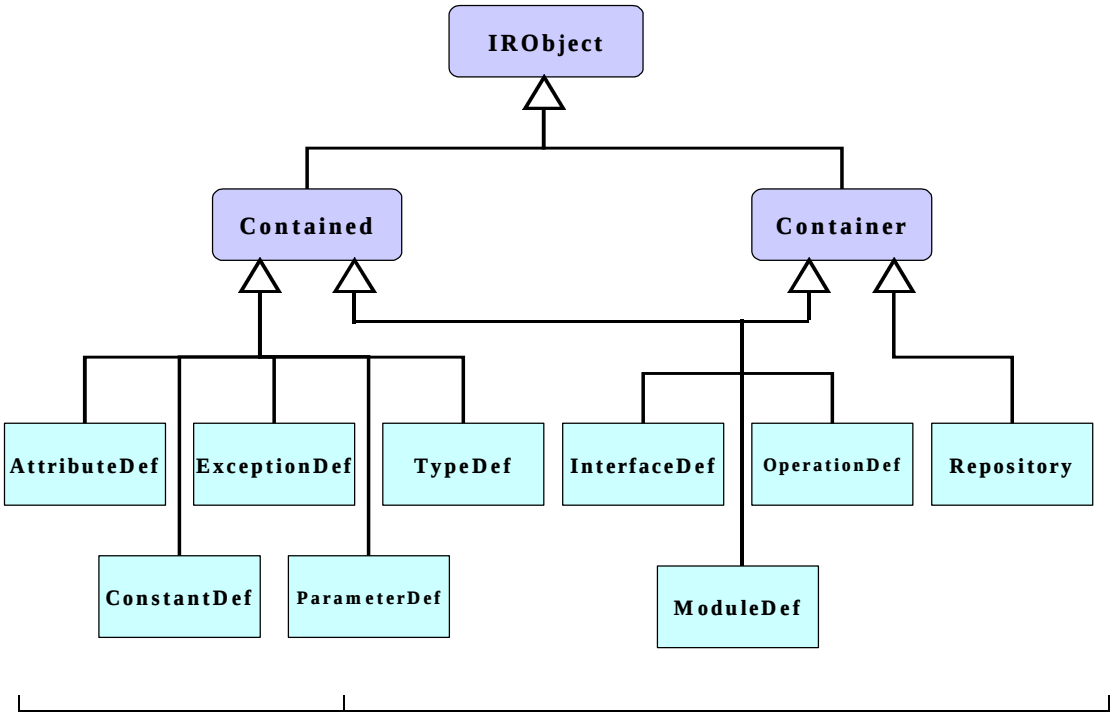
IDL の構造に対応するこれら８つのインタフェースに加えて，CORBA はリポジトリのネームスペースに含まれるすべてのモジュールのルートとして機能する Repository インタフェースを仕様で規定している．

２）インタフェースリポジトリのクラス階層

CORBA のインタフェースリポジトリは図 3.3 に示すように抽象スーパークラスである，IRObject，Contained，Container の３つを定義した．すべてのインタフェースリポジトリオブジェクトは IRObject インタフェースから派生している．このインタフェースは，destroy オペレーションを提供するとともに，オブジェクトの実際の型を識別するための属性オペレーションを提供する．コンテナとなるオブジェクトは探索オペレ

表 - 3.3　　IDL の構造に対応するインタフェース

Contained は他



3.3.2　サーバ側から見たCORBA の機能

（1） IDL スケルトン

サーバが公開した各サービスへの静的インタフェースを提供．IDL コンパイラを利用して作成される．

（2） 動的スケルトンインタフェース

動的スケルトンインタフェース DSI（Dynamic Skeleton Interface）は，サーバにおける実行時バインディングメカニズムを提供．これにより IDL を基に作成されたスケルトン（あるいはスタブ）を持たないコンポーネントに送られてくるメソッド呼び出しを処理．動的スケルトンは，送られてきたメッセージ内のパラメータ値により，ターゲットオブジェクトとメソッドを判断．これに対して通常のスケルトンは，特定のオブジェクトクラスに対応して定義され，IDL 定義の各メソッドに対応するメソッドインプリメンテーションがあることを前提としている．

クライアント側で説明した DII のサーバ側に相当するものがＤＳＩである．

（3） オブジェクトアダプタ

ORB の中核をなす通信サービスの最上位に位置し，サーバオブジェクトに代わりサービスの要求を受け付ける．サーバオブジェクトのインスタンスを作成したり，作成したオブジェクトへの要求の送信や，オブジェクト ID（CORBA ではオブジェクトリファレンスと呼ぶ）の割り当てを行ったりするためのランタイム環境を提供．CORBA 準拠の ORB は，基本オブジェクトアダプタ BOA（Basic Object Adapter）と呼ばれる標準アダプタをサポートすることを義務づけていた．

しかし，CORBA 2.2 で BOA は削除され，代わりに POA（Portable Object Adapter）が規定されるように変更された．これは今までの BOA が ORB 製品によって，その実装が非常に異なっていて，インタフェースやその使い方すら重要なところで違いがあり，サーバ側のプログラミングのポータビリティが実現できなかった．もう一つの理由として，BOA は本格的な CORBA システムを開発するための機能が不足していた．POA はこれらの問題を解決するために開発された，オブジェクトアダプタの新しい標準インタフェースである．しかし現在のところ主要な ORB 製品はまだ POA をサポートしていないのが現状のようだ．

図－3.3　インタフェースリポジトリクラス群における継承の階層

（4） インプリメンテーションリポジトリ

ランタイムリポジトリであり，サーバがサポートするクラス，およびインスタンス化されたオブジェクトとその ID に関する情報を格納している．ORB の実装に依存する

特別な情報を格納するための共通の領域としても利用される．トレース情報，監査履歴，セキュリティなどの管理情報などが格納される．

（5） ORB インタフェース
クライアント側で提供されるものと同様に，いくつかのローカルサービスに対応する API から構成されている．

3.4 CORBA サービス

CORBA は，アプリケーションにとって有用である多くのサービスを定義している．CORBA サービスはシステムレベルのサービスの集合であり，IDL で規定されたインタフェースを備えている．オブジェクトサービスは ORB の機能を拡張し，補うものと解釈できる．これらを使用することで開発者は，アプリケーション開発を容易にすることができる．

OMG は CORBA を含むすべての仕様と同様，サービスの実装までは定義せずサービスを使えるようにするインタフェースを提供している．

以下に各 CORBA サービスについて簡単に見ていくことにする．

3.4.1 コンカレンシコントロールサービス

コンカレンシコントロールサービスは，共有される CORBA オブジェクトにおける並列性を管理するためのインタフェースを提供．これは，サービスによってサポートされる数種類のロックを利用します．

3.4.2 イベントサービス

イベントサービスは，CORBA オブジェクトがイベントを送受信するメカニズムを提供する．これには次のような特徴がある．

1） イベントが目的のオブジェクトに届くことを保証する信頼できる配信
2） イベントの配信に関するプッシュとプルモデルのサポート．
3） サプライヤがイベントのコンシューマを知る必要がない場合，あるいはその逆の場合の匿名メッセージ
4） コンシューマがある種類のイベントに同意できるようにするパブリッシュ／サブスクライブに似たメカニズムであるイベントチャネル．

3.4.3 外部化サービス

外部化サービスは，オブジェクトを外部化（シリアライズ）したり，内部化するためのインタフェースを提供する．CORBA オブジェクトの値渡しのメカニズムにおいて，アプリケーションは外部化サービスが使用できる．

3.4.4 ライセンシングサービス

プロバイダは，ライセンシングサービスにより，サービスの利用を制御する方針を定義できる．時間，値のマッピング，カスタマの 3 種類のライセンス方法をサポートする．

3.4.5 ライフサイクルサービス

ライフサイクルサービスは，CORBA オブジェクトの生成，削除，コピー，移動といった機能を提供する．

3.4.6 ネーミングサービス

ネーミングサービスにより CORBA オブジェクトは名前により登録でき，そして特定される．このサービスは，ネーミングコンテキストの概念を使う．これは，一意な名前のセットを含む.

標準的なバインドメカニズムの一部のように, CORBA オブジェクトは名前を与えられ, その名前によって他のオブジェクトはそれらを見つけ出すことができる．この機能は，ネーミングサービスの縮小版と考えることができるが，実際のネーミングサービスはもっとスケーラブルである．

3.4.7 オブジェクトトレーダサービス

トレーダサービスは，ネーミングサービスに似ていて，これにより他のオブジェクトは CORBA オブジェクトを特定する．オブジェクトを特定するために名前を使うというよりは，むしろオペレーションの名前，パラメータ，結果の種類に基づいてサービスを探す.

ネーミングサービスとの大きな違いは，ネーミングサービスはサービスが外部に公開している名前を知っていれば特定のサービスを見つけられ，トレーダサービスは位置や機能あるいはイベント名に基づいてサービスを特定できる．

3.4.8 永続オブジェクトサービス

永続オブジェクトサービスは，オブジェクトの永続性を管理するためのインタフェース群を提供する．永続オブジェクトは，一定期間存続するオブジェクトである．つまり，アプリケーションが終了してもオブジェクトは存続する．使われない間オブジェクトはデータベース，ファイルといった永続的な格納場所におかれ，必要となった場合に取り出されたり，呼び出される．

3.4.9 プロパティサービス

オブジェクトは，プロパティサービスにより名前／値のペアといったプロパティを定義できる．このサービスを使って，コンポーネントの状態をプロパティに動的に対応づけることができる．

3.4.10 クエリサービス

クエリサービスは，オブジェクトでクエリが使えるようにする．オブジェクトに対する問い合わせ操作を提供する．クエリは属性値に基づいて，オブジェクトの動作を指定する属性を含む.

3.4.11 リレーションシップサービス

リレーションシップサービスによって，オブジェクト間の関係を示すことができる．これは，関連の種類やカーディナリティをチェックすることで完全な制約を考慮する．また関係するオブジェクトをコピーしたり，移動したり，削除するライフサイクルサービスとともに働く．このサービスは複雑なオブジェクト間の関係の管理を容易にする.

3.4.12 セキュリティサービス

分散環境におけるセキュリティの確保は，非分散環境に比べはるかに困難だが重要なもの. セキュリティサービスは，そのセキュリティ機能のためのインタフェースを明記する. その機能について次にまとめます.

1） ユーザが自分自身であることを証明する識別と認証.

2） ユーザがサービスあるいはオブジェクトにアクセスできるかどうかを決める権限とアクセス制御.

3） ユーザの動作の記録を提供するセキュリティ検査.

4） サービスに対するユーザの認証（またその逆），完全な保護，信頼できる保護を含む通信のセキュリティ.

5） 元のデータあるいは受け取ったデータが間違いないことを証明する，電子署名に似た能力を提供する否認不可.

6） さまざまなセキュリティ方針のアドミニストレーション.

3.4.13 時間サービス

時間サービスによって，ユーザは現在の時間を得ることができる．これは，イベントの順番を決定したり，時間に基づいてイベントを発生させることができる.

3.4.14 トランザクションサービス

トランザクションの機能をサポートするインタフェースを提供する．トランザクション処理は次の４つの特性を満たす必要がある.

1） それ以上分割できない処理単位では，すべての処理が成功か失敗のいずれかでなければならないという原子性（Atomicity）

2） トランザクションの実行結果として，すべてのオブジェクトの状態は，アプリケーションにとって一貫性（Consistency）のあるものでなければならない.

3） 並行して実行される複数のトランザクションは，分離して実行されなければならない．つまりあるトランザクションは，別のトランザクションの中間結果に依存した処理を行ってはならない．これを分離性（Isolation）という.

４）　トランザクションが成功して完了したときの結果は失われてはならないという耐久性（Durability）．トランザクションの結果を永続的な記憶域に書き込む必要がある．

3.4.15　スタートアップサービス

ORB が起動された際にリクエストを自動的に起動することを可能にする．

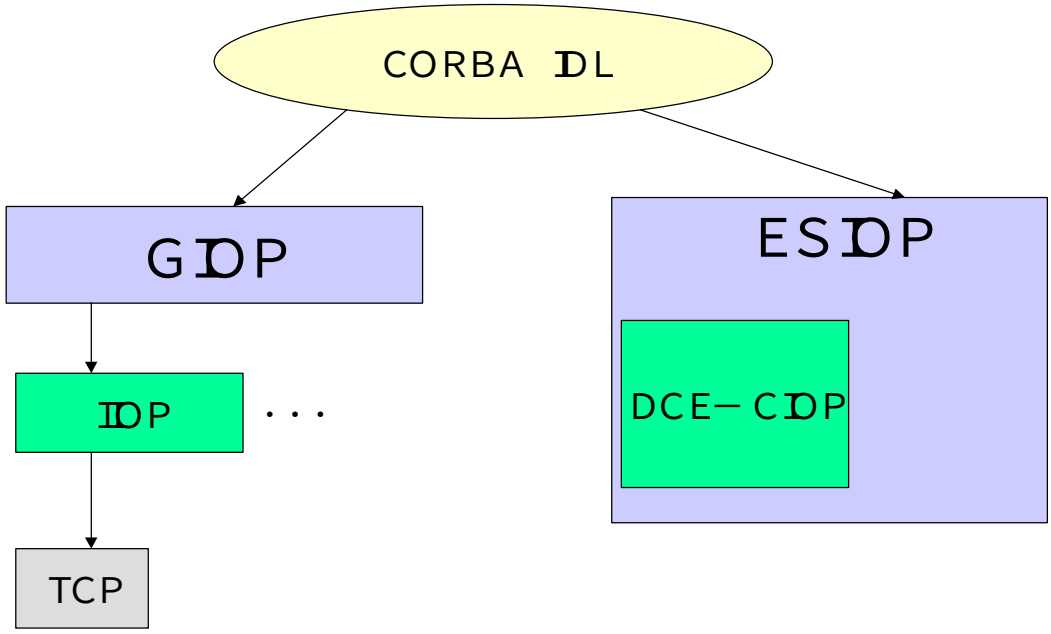
3.4.16　コレクションサービス

分散環境で共通に用いられるコレクション（オブジェクトの集合を保持する）を作成，操作したりするためのインタフェースを提供する．

3.5　CORBA インタオペラビリティ

クライアントとサーバが別の ORB 環境であると CORBA1.0 では処理が行えなかった．そこで，ORB 製品間の相互運用性実現のため標準化されたのが IIOP（Internet Inter-ORB Protocol）である．

オブジェクト間の通信を実現するのには共通のプロトコルが必要であり，そのプロトコルはメッセージ交換規則や転送構文を含む必要がある．CORBA2.0 以降では共通の上位プ



図－3.4　インタオペラビリティアーキテクチャ

ロトコルを定め，下位プロトコルにマッピングするアプローチを採っている．そして，トランスポートレイヤでの上位プロトコルは GIOP(Ｇｅｎｅｒａｌ Inter-ORB Protocol) に定められ，トランスポートレイヤとして TCP を扱うときには IIOP が必須のプロトコルである．また，あるアプリケーションレイヤでは ESIOP（Environment Specific Inter－ORB Protocol）を用いて DCE-RPC のときには DEC-CIOP を使う．ブリッジを使用することでプロトコルの変換をしインタオペラビリティを考慮しない製品を扱うことも可能である．

（1） IOR

分散オブジェクト環境で必要なものに分散オブジェクトリファレンスがある．この分散オブジェクトリファレンスの相互運用性を考慮し，表現のためのフォーマットが IOR（Interoperable Object Reference）である．

（2） CDR

CORBA におけるパラメータやリターン値のデータ転送のフォーマットは CDR（Common Data Representation）で規定されている．データは転送を行う前に実行環境のフォーマットから CDR に変換し，送り先の実行環境のフォーマットに変換することによって利用する．

以下にその特徴を示す．

1） IDL のデータ型をすべてサポートしている．

2） バイトオーダは，Big Endian と Little Endian の両方ともサポートしている．使用時には指定する．

3） データ型を表現するためのデータ長を固定し，簡単に処理できるバイト長にしてある．型としてデータ長を決めることができない String 型，wString 型，Sequence 型は，CDR 内にデータ長を含む．

一般 ORB 間プロトコルGIOP

CORBA では，コネクション型のトランスポートプロトコルを使用して，リクエストの転送を実現するアプリケーションレベルのプロトコルとして一般 ORB 間プロトコル GIOP（General Inter-ORB Protocol）を定めている．

GIOP は次の 3 つの要素から構成されている．

1） 共通データ表現 CDR

OMG IDL のデータ型からネットワーク上でのデータ表現へのマッピングを定めている．

2） メッセージ・フォーマット

オブジェクト間通信で使用されるメッセージのフォーマットを定めている．

3） メッセージ転送規則

下位トランスポートのコネクションの使い方，メッセージの発行の順序規則を定めている．GIOP の特徴は，簡単なプロトコルであり，任意のトランスポートにマッピングでき，

いろんな ORB の実装ができる．

インターネットORB 間プロトコルIIOP

IIOP が行うのは，GIOP の TCP/IP へのマッピングである．サーバとの間にコネクションを確立して GIOP で定義されたメッセージを送受信する．

環境固有ORB 間プロトコルESIOP

ESIOP （Environment Specific Inter－ORB Protocol）別の特定のネットワークとの円滑な相互運用を目的としている．

（3） DCE-CIOP

ESIOP の最初のプロトコル仕様が DCE-CIOP である．4 つの規則がある．

1） 転送構文

GIOP でマーシャリングされたメッセージをバイト列として転送する．

2） メッセージのフォーマット

3） プロファイルデータ

サーバのストリングバインディング・サーバの名前情報を含む．

4） オブジェクト探索のメカニズム

クライアント側の ORB がサーバオブジェクトを探索するメカニズムを規定している．

3.6 CORBA オブジェクトの開発手順

クライアント／サーバ間の，静的起動と動的起動のどちらの場合でも，クライアントはオブジェクトリファレンス（オブジェクト ID）にアクセスし，リクエストを出し，メソッドを起動してサービスを実行する．クライアントでは静的起動と動的起動を関知しない．

サーバオブジェクトの実装，オブジェクトアダプタ，ORB などはすべてクライアントの静的起動，動的起動に関わらず透過的である．静的なインタフェースは IDL プリコンパイラによって，スタブという形で直接作成される．静的なスタブのインタフェースはコンパイル時にクライアントプログラムに結合されるため，メソッドの動的起動に比べて次のような利点がある．

・プログラミングが容易

リモートのメソッドでも単純に名前で起動し，パラメータを渡すだけで呼び出すことができる．

・型のチェックが厳しい

コンパイル時に強制的に行われる．

・コードの可動性が良い
　コードを読むことで何がしたいかわかる．
・性能が良い

対照的に，メソッドの動的起動は，もっと柔軟性のあるものとなっていて，クライアントのコードを変更せずにシステムに新しいクラスの追加などの変更をサーバ側に加えることができる．メソッドの動的起動は非常に大変な作業となる．メソッドを起動させることは簡単であるが，リクエストの作成が大変となる．またリクエストの作成と，リモートメソッドの起動には複数のやり方があり，これにより複雑さが増す．動的起動は，柔軟性と拡張性を得るが，それには複雑化と性能劣化という代償が必要となる．

3.6.1　メソッドの静的起動

CORBA における静的起動の流れを以下に示し，図‐3.5 に開発手順を示す．

1）　IDL を用いてオブジェクトのインタフェースを定義する．
　サーバにおいて使用可能なオペレーションと，その起動方法を伝えるための手段である IDL を使い，オブジェクトの型とそれが持つ属性，エクスポートするメソッドと，そのメソッドが持つパラメータのそれぞれを定義する．

2）　ターゲットとなる言語のプリコンパイラを通し，IDL ファイルをコンパイルする．
　実装言語のプリコンパイラにより IDL をコンパイルし，スタブ，スケルトンを生成する．

3）　クライアント，サーバの実装．
　クライアントとサーバのコードの記述．

4）　クライアント，サーバのコンパイル．
　クライアントとサーバ，それぞれの実装言語のコンパイラにより，コンパイルする．

5）　クラスの定義をインタフェースリポジトリにバインドする．
　一般的には，IDL の情報を，ユーティリティを使ってプログラムが実行時にアクセスするインタフェースリポジトリにバインド，もしくはロードする．

6）　実行時オブジェクトをインプリメンテーションリポジトリに登録する．
　オブジェクトアダプタは，オブジェクトリファレンスと，サーバにインスタンス化するすべてのオブジェクトの型を，インプリメンテーションリポジトリに登録する．

7）　サーバ上のオブジェクトをインスタンス化する．

3.6.2　動的起動

スタブを使用しないで，実行時にバインディングするアプローチ．動的起動インタフェース DII（Dynamic Invocation Interface）を使うと，システムの柔軟性と拡張性を保つことができる．サーバは新しいサービスやインタフェースを，それらが利用可能になった時いつでも提供でき，クライアントはこれらのインタフェースを実行時に発見しその呼び出し方を知る．

オブジェクトのメソッドを動的に起動するには，前もってオブジェクトを見つけそのリファレンスを取得する必要がある．これによりオブジェクトのインタフェースを検索し，リクエストを動的に作り上げることができる．リクエストには，インタフェースリポジトリから取得したメソッドとそのパラメータを記述する．

クライアントがリモートのオブジェクトを見つける手段として次のような方法がある．

・クライアントに対し，文字列化したオブジェクトリファレンスを提供．この文字列を実在のオブジェクトリファレンスに変換してから接続する．

・CORBA のネーミングサービスを使い名前からオブジェクトを探し出す．

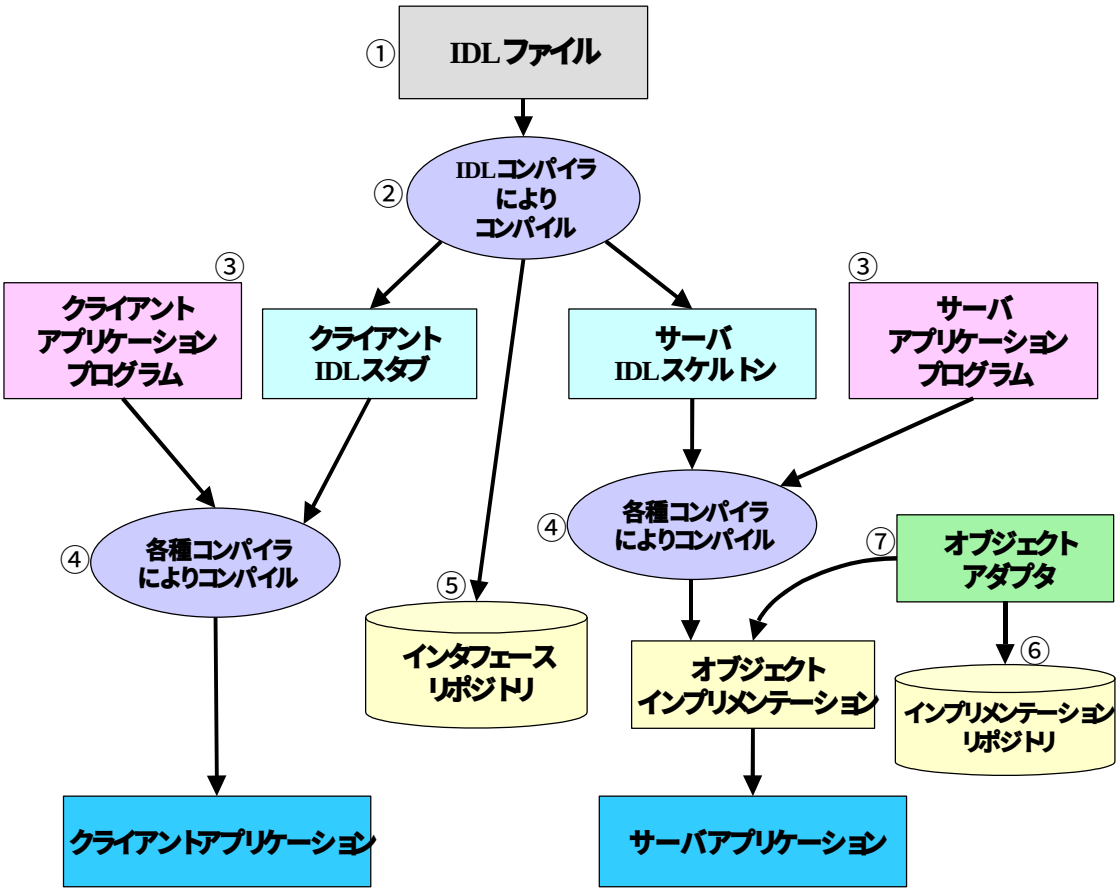
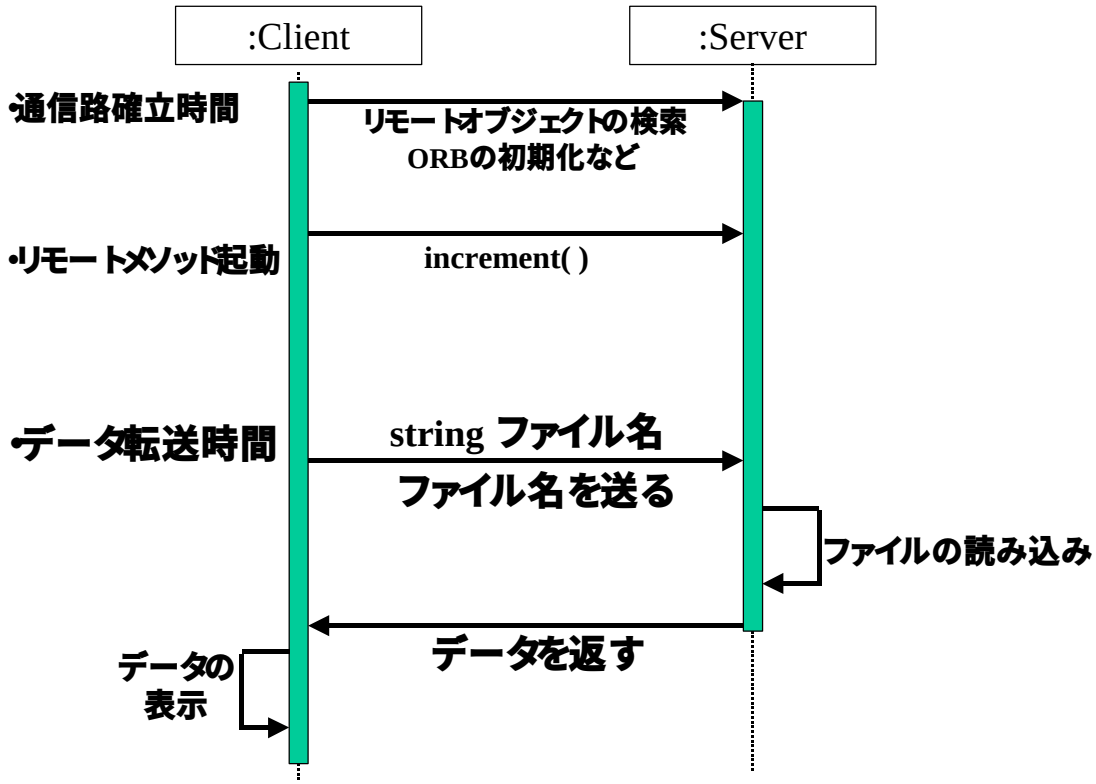


図 - 3.5　CORBA オブジェクト開発手順

・CORBA のイエローページであるトレーダサービスから見つけ出す．

これらの方法により，クライアントはリモートオブジェクトを見つけ出し，リモートメソッドを起動させる．

4章　分散オブジェクト環境CORBA の性能評価



図‐4.1　性能評価シーケンス図

分散オブジェクト環境における性能評価として，通信路確立時間，リモートメソッド起動時間，データ転送時間などの評価方法が挙げられる．今回，様々な条件での CORBA におけるデータ転送時間の違いをみることによってその性能を評価する．図‐4.1 に性能評価のシーケンス図を示す．

4.1 データ転送時間

データ転送時間とは，クライアントがサーバに対してリクエストを出し，サーバからクライアントにデータを送信してクライアントがデータを受け終えるまでの所要時間とする．
クライアント側でのファイルに書き出す時間と，サーバ側での処理時間は含まない．

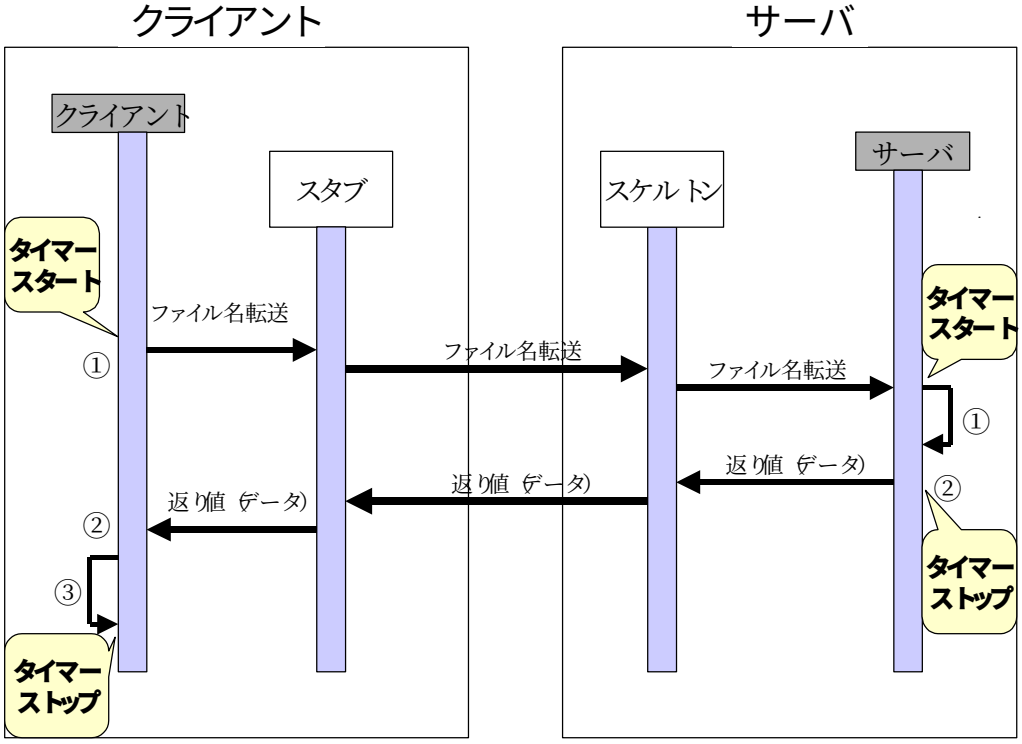


図-4.2 データ転送のシーケンス図

図-4.2 のシーケンス図を以下に説明する．

1）クライアント側

① ファイル名を指定し，リモートメソッドを起動

② データを受ける

③ データをファイルへ書き込む

2）サーバ側

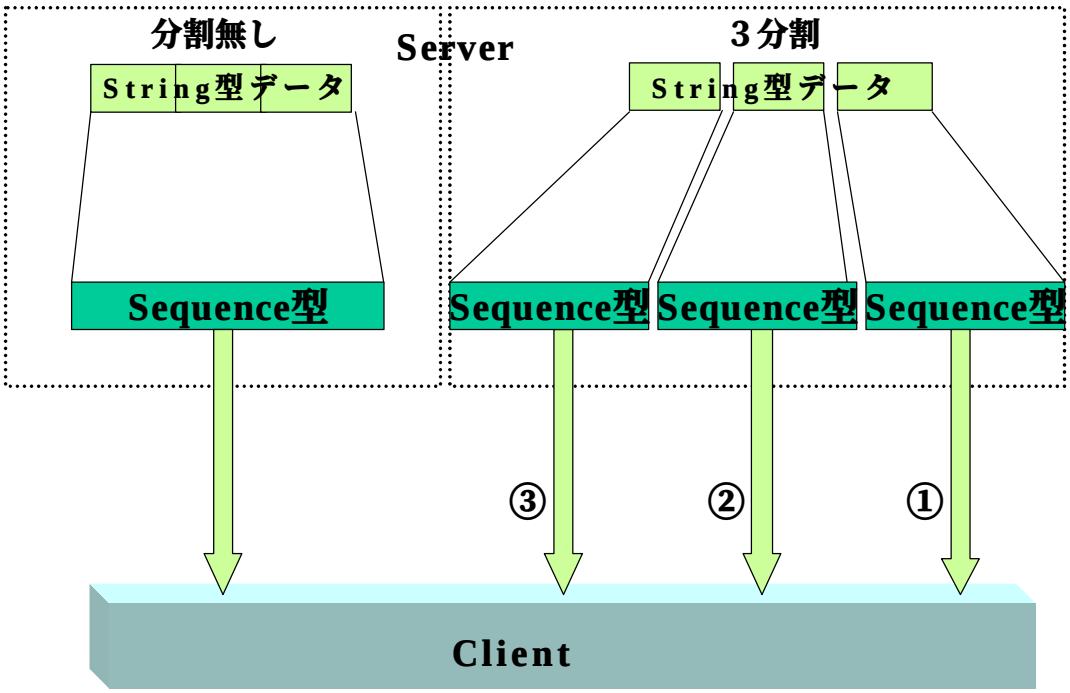
① 指定されたデータをファイルから読み込む

② 型を指定してデータを転送

4.2 CORBA の Sequence 型と String 型

4.2.1 CORBA の Sequence 型

Sequence は，指定されたデータ型の 1 次元可変長配列として扱う．実際に送受信されるデータサイズは実行時に決まるので，可変長データを効率良く転送できる．IDL によって Sequence を可変長のデータとして扱うときには要素数は指定せずに扱う．最大サイズを指定する Sequence 型を Bouded Sequence といい，最大サイズを指定しない Sequence 型を unBouded Sequence という．変数をオブジェクト型で宣言して，CreateType や CreateTypeById で各要素のインスタンスを作らなくてはいけない場合がある．それは，



Sequence が構造体，共用体，any 型，オブジェクト，TypeCode を扱うときである．

文字列型 String[8]の要素データを IDL において Sequence〈string,3〉のように 1 次元配列 3 要素に分割してクライアントアプリケーションにデータ転送を行うとき，9 要素のデータを 3 要素にまとめて送ることができる．

図‐4.3 に３分割したときのデータ転送の仕方を示す．分割しないときには，データを 1 つの Sequence 型として転送する．データ分割を行うと，分割したデータが 1 つ 1 つの Sequence 型として転送される．１つのデータを転送し終わると，次のデータを転送していく．図‐4.3 では，①の分割したデータが転送されると②を，②が転送されると③が転送される．

Sequence 型（C++）	IDL マッピング	Sequence 型は，Sequence 長_length と最大長_maximum とバッファポインタ_buffer を private データのに持った C++の Sequence クラスと var 型クラスにマッピングされる．
	クライアントアプリケーションでの処理	new 演算子でインスタンスの領域を得て，delate 演算子で開放する．allocbuf 関数で Sequence 要素の配列を割り当てる．
	サーバアプリケーションでの処理	データをクライアントアプリケーションに転送するときにはサーバで領域確保を行う．
Sequence 型（Java）	IDL マッピング	ＩＤＬの Sequence は，ＩＤＬＴｙｐｅパッケージの下に定義されている Sequence を含む型のデータ型の配列(value)とコンストラクタ(Sequence+型名)を持つクラスから継承した Sequence と同一名の Java クラスにマッピングされる．
	クライアントアプリケーションでの処理	new 演算子によって Sequence 型のデータ領域を割り当てる．Sequence にデータを割り当てるためには，個数指定によりインスタンスデータ value に領域が割り当てられ，また，value において Sequence データを参照できる．

4.2.2 CORBA の文字列型（String 型）

String 型は，¥0 で終了する文字列であり，固定長文字列は長さ指定をする Bounded String として，可変長文字列は長さ指定をしない unBounded String として扱う．

String 型（C++）	IDL マッピング	String 型で const 宣言されているものは C++の CORBA::char*にそれ以外は String_var クラスにマッピングされる．
	クライアントアプリケーションでの処理	サーバアプリケーションにデータを転送するときには，文字列を格納する領域と終端文字を‘¥0‘を含む領域を確保して，使用が終了したら開放する．
	サーバアプリケーションでの処理	データをクライアントアプリケーションに転送するときにはサーバで領域確保を行う．領域はスケルトンによって自動的に開放を行う．

図‐4.3 Sequence 型の分割指定での転送

String 型（Java）	IDL マッピング	Java の java.lang.String クラスには，長さの指定がないので長さ指定した String を扱う場合には IDL 定数宣言をすることで長さを指定し使用する.
	クライアントアプリケーションでの処理	サーバアプリケーションにデータを転送するときには演算子 new を用いて領域を確保する.

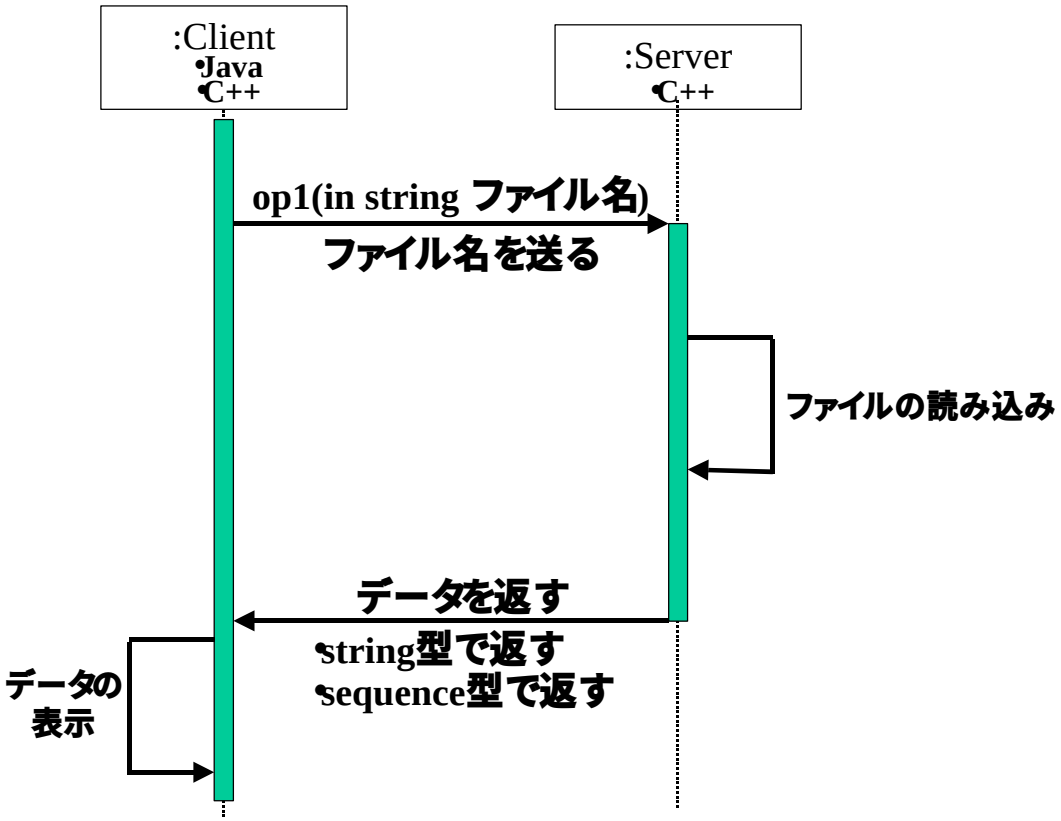
4.2.3 String 型と Sequence 型の違いについて

String 型の C++マッピングは，リモートコールのためにマーシャリングするためのクライアントスタブは String 型の長さを含む必要がある．そのため，String_length（String 型の長さ）が必要であり，処理のオーバーヘッドを String 型に付加することになる．String 型は，クライアントスタブで String_length を呼び出さなくてはいけないが，Sequence 型は長さが C++マッピングにおいて付加されるため，String_length 関数を呼び出す処理の差が性能の差として現れる.

4.3　性能評価の方法

CORBA におけるデータ転送時間が，データを分割して転送したとき，その分割数の最大要素数を IDL で指定した場合と，指定せずにサーバでデータを分割して転送したときの違いを評価する．またデータを転送する際のデータ型による違いなどから性能を評価する．転送するデータは，サーバ側に文字列の入ったファイルを置き，その文字列を転送する．ファイルのサイズは 1Kbyte～２Mbyte まで変化させ，平均転送時間から，転送速度を調べる．また実装言語の Java と C++による転送速度の違いを評価する．実装は，クライアントを Java と C++で実装し，サーバについては使用した CORBA 製品（Object Director 試用版）が Java でサーバを実装できないため C++で実装した．

測定する回数として誤差などを考慮し，1 つのデータに対し 12 回ずつの記録をとる．そ



のうち最初の１，２回で値の大きくはずれているようなものを除き平均転送時間とする．得られた平均転送時間より，転送速度を得る．

また LAN の転送速度は１０Mbps と 100Mbps の 2 種類を使用した.

4.3.1　データ分割によるデータ転送時間の評価

転送時のデータ型を Sequence の String 型とする．データを一括して転送する方法と，いくつかに分割して転送した場合についてその評価をする．分割した場合には，データ分割の最大要素数を IDL で指定することができ, IDL 指定があるものとサーバ側で分割数の最大要素数を引数として与えてデータを転送する．分割サイズは 1（分割なし），10，100，1000，10000，20000 の 5 種類とする．このデータ分割の最大要素数を IDL で指定（Bounded）したものとサーバ側で引数として与える場合（UnBounded）とを比較し評価する.

一括して転送する場合には分割サイズを 1 として分割要素を IDL 指定する場合と IDL 指定しない場合の転送速度の評価する.

以下に IDL におけるデータ分割の最大要素数の指定方法を示す.

Sequence＜string，サイズ＞　A　　：サイズ指定あり

Sequence＜string＞　　B　　：サイズ指定なし

この評価では，データを分割して転送することで，その転送時間を評価するため，転送するデータのサイズは，使用するデータで最も大きい 2Mbyte のファイルを使用しその転送時間を測定する.

4.3.2　Sequence 型と String 型でのデータ転送時間の評価

データを転送する際のデータ型による転送時間から転送速度を評価する．データ型として文字列を扱える Sequence 型と String 型について測定し評価する.

Sequence 型は基本データ型の一次元配列である．Sequence 型の基本データ型を変えることで，　データ転送時間に違いがでるか測定し，評価する．今回，転送するデータが文字列なので基本データ型として，文字型の char 型と文字列型の String 型を用いる.

Sequence＜string＞型については，IDL におけるデータ分割の最大要素数の指定せず，データを一括して転送するものを比較対象とする.

4.3.3　実装言語の違いによる評価

CORBA のデータ転送での，実装言語による違いを評価する．使用した CORBA 製品（Object　Director 試用版）が Java でサーバを実装できないためサーバは C++で実装した．この評価ではクライアントの実装言語を変えることでその違いを比較する．実装言語

図 - 4.3 データ転送における性能評価シーケンス

については，Java と C++で実装した.

Sequence＜sting＞型を評価対象として，データ分割して分割要素の指定をしたものと，していないものの 2 種類について，クライアントを Java と C++で実装してそれぞれ測定し評価する．

4.4 実行環境

計測環境を表‐4.1 に示す．

表‐4.1　性能評価の実行環境

クライアント	サーバ
CPU ： P55C‐200 メインメモリ ： 32MB	CPU ： P6‐200 （内部キャッシュ 256KB） メインメモリ ： 64KB
OS ： Windows98	OS ： WindowsNTServer4.0
JavaVM ： JDK1.1.6	JavaVM ： JDK1.1.6
Java Compiler ： JDK1.1.6	Java Compiler ： JDK1.1.6
C++ Compiler ： Visual C++5.0 Enterprise Edition	C++ Compiler ： Visual C++5.0 Enterprise Edition
SymfoNET/ObujectDirector Sever V3.0 for WindowsNT 試用版	

5章　評価結果

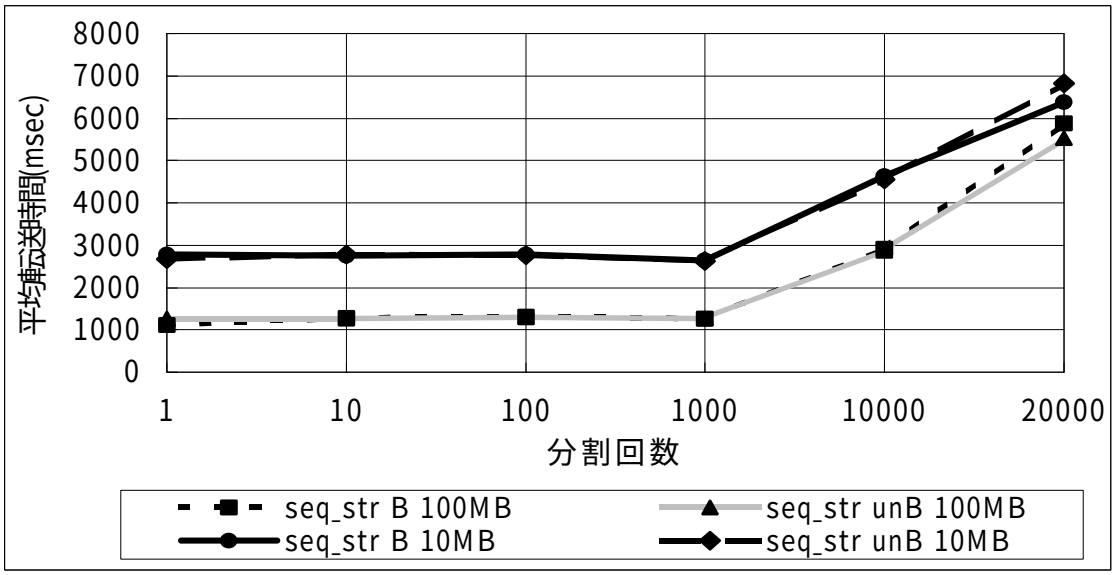


図-5.1　データ分割時の平均転送時間

CORBA のデータ転送時間における評価結果を示す．データ分割による違い，Sequence 型における基本データ型の違い，Sequence 型と String 型との比較，および実装の違いにおける比較について評価した．LAN は 10Mbps，100Mbps の２つを用いた．

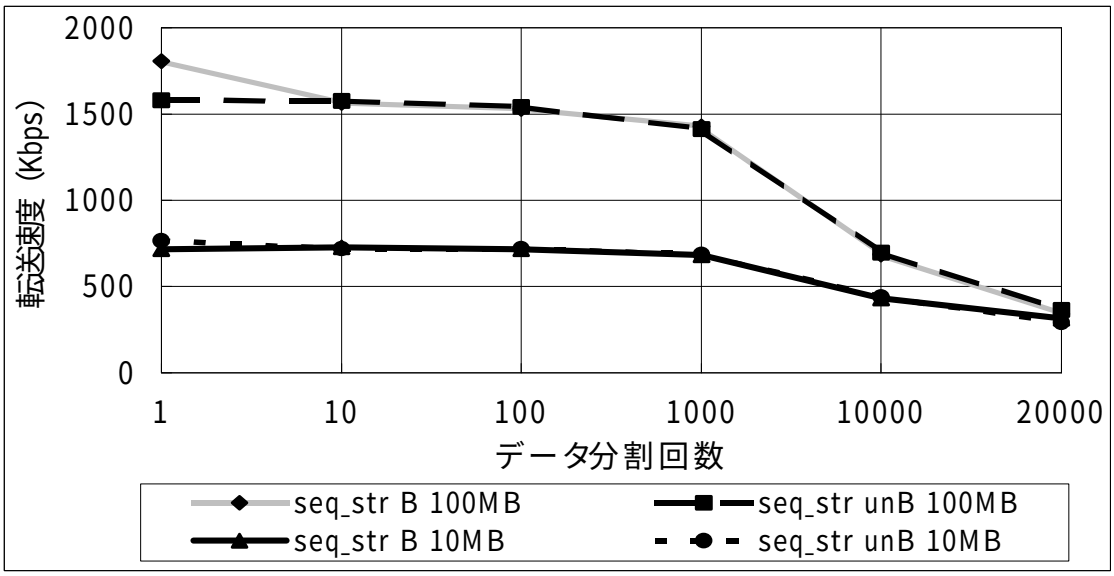
5.1 データ分割によるデータ転送時間の評価

図‐5.1 にデータ分割時の平均転送時間，図‐5.2 にデータ転送速度を示す．

結果より，分割要素数の指定による違いはほとんど見られない．

結果を見るとデータ分割数が 1000 までは LAN の性能に関わらず転送速度に変化は見られない．分割数が 10,000 になると極端に転送時間がかかり転送速度が遅くなっている．また，20,000 分割したときには LAN の性能による転送時間の有意の差は見られない．

図‐5.3 に対数近似曲線のグラフを示す．



図‐5.2 データ分割時の転送速度

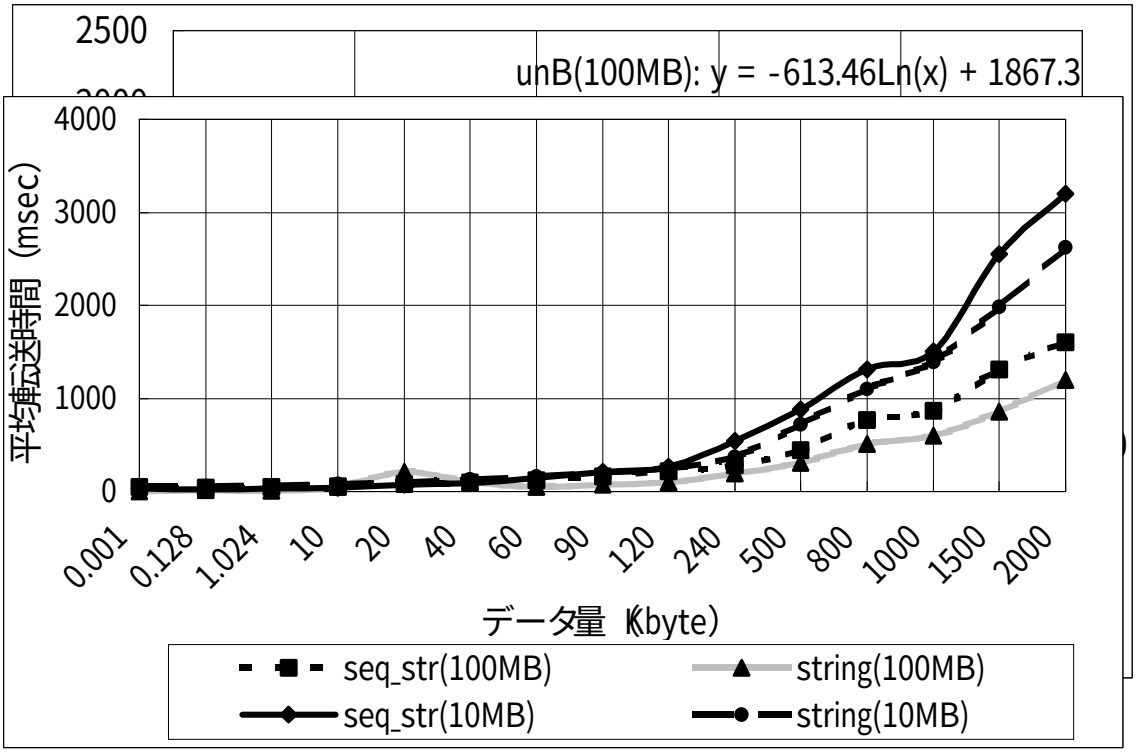


図-5.4　データ型の違いによる平均転送時間

5.2　Sequence 型と String 型でのデータ転送時間の評価

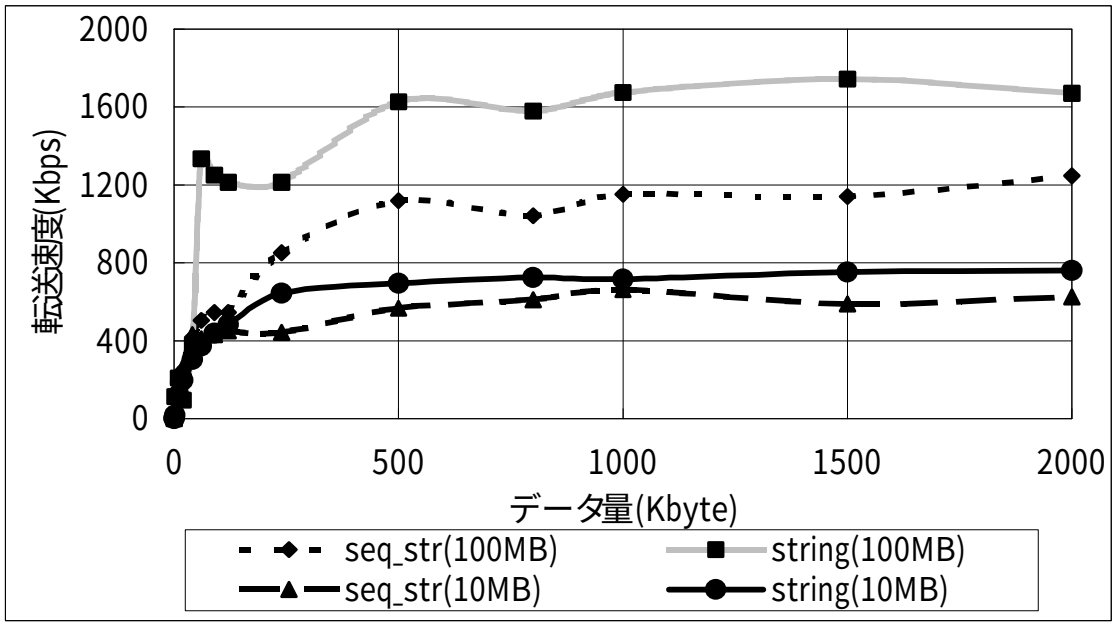


図-5.6　データの違いによるデータ転送速度

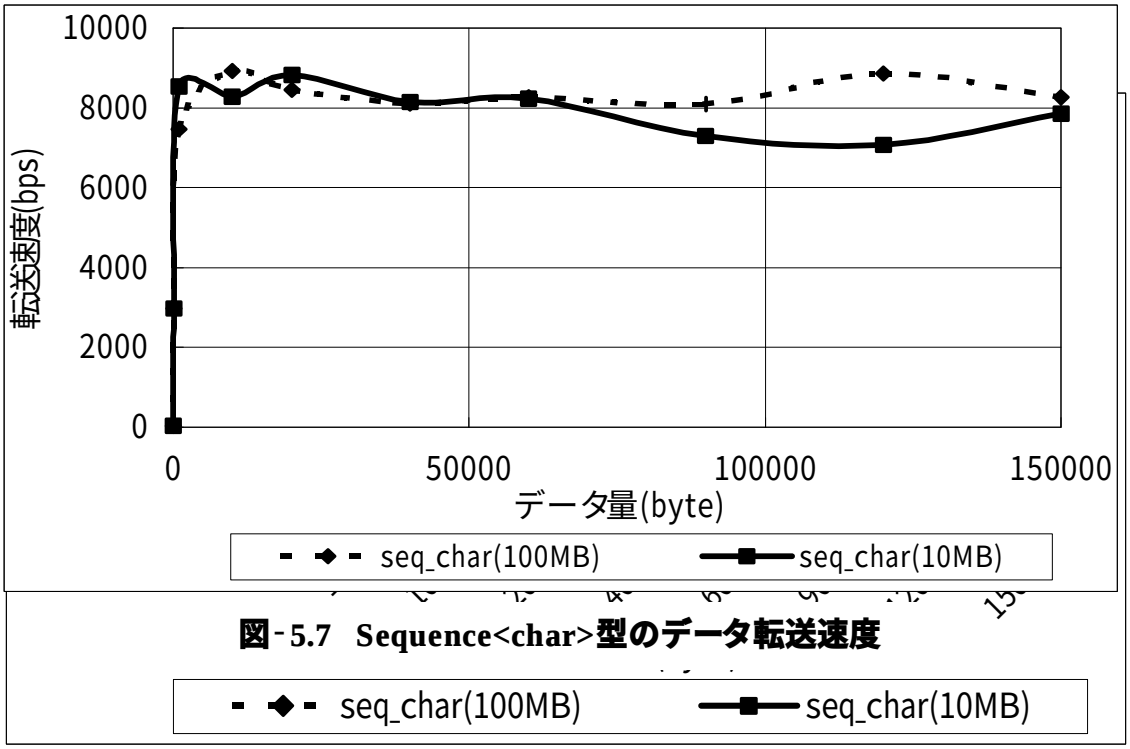
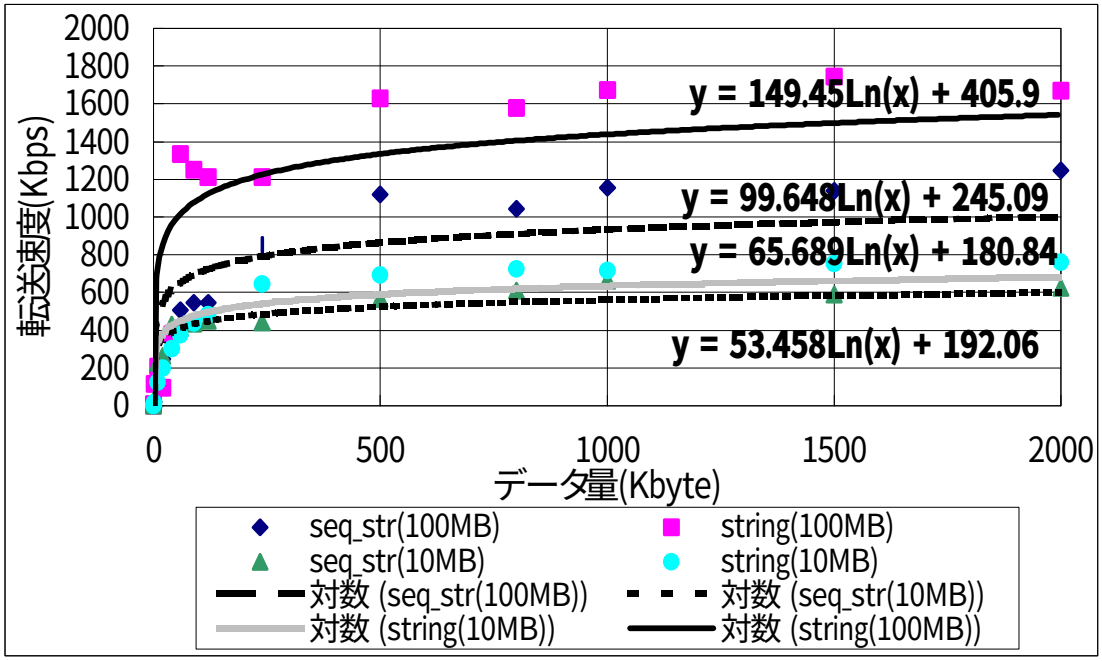


図-5.7　Sequence<char>型のデータ転送速度

図-5.5　Sequence<char>型の平均転送時間



図‐5.8　データの違いによるデータ転送速度

図‐5.4，図‐5.5 に Sequence 型と String 型の平均転送時間，図‐5.6，図‐5.7 にデータ転送速度を示す．Sequence<char>型については，実装が上手くいかなかったためか，データ量が最大で約 150Kbyte しか転送することができなかった．また平均転送時間，転送速度についても他のデータより遅かったため，抜き出したグラフを別に示した．図‐5.8 に図‐5.6 についての対数近似曲を示す．

全体的に見て LAN の性能に伴い転送速度も高くなっていることが分かる．また LAN の性能に関わらず，Sequence<string>型よりも String 型の方が転送速度は速くなっている．LAN の性能が上がると転送速度は 2 倍になる．

いずれの場合も 2Mbyte までの測定で，転送速度の著しい低下は見られない．

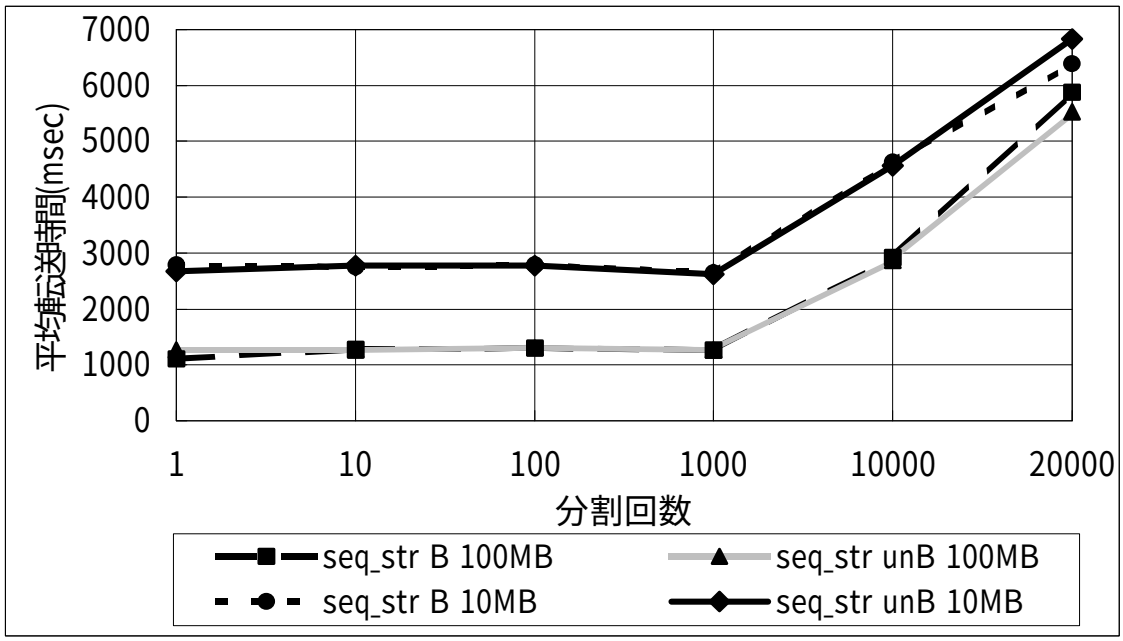


図-5.9　実装別平均転送時間（Java）

5.3　実装言語の違いによるデータ転送時間の評価

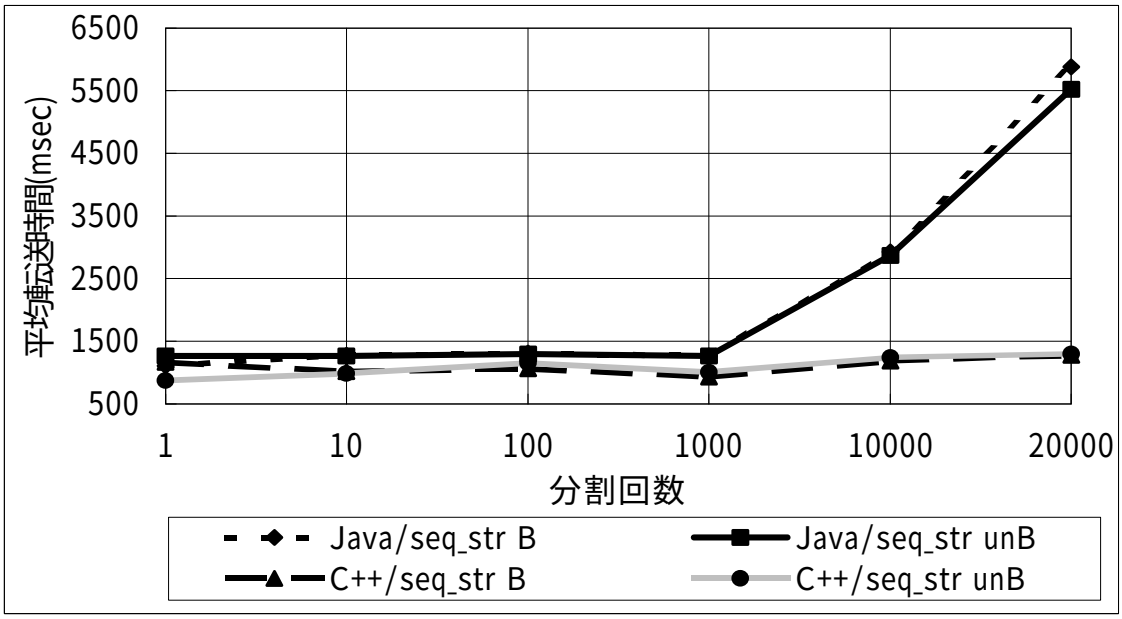


図-5.11 LAN の性能別平均転送時間（100MB）

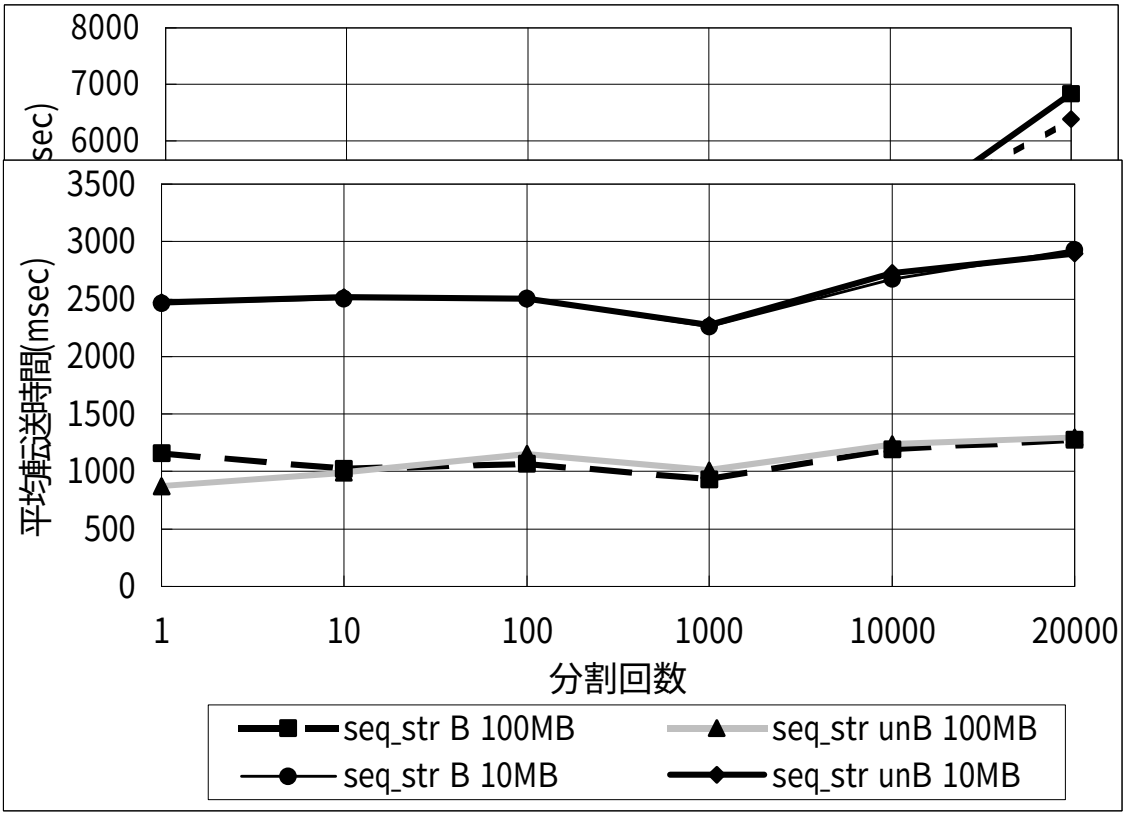
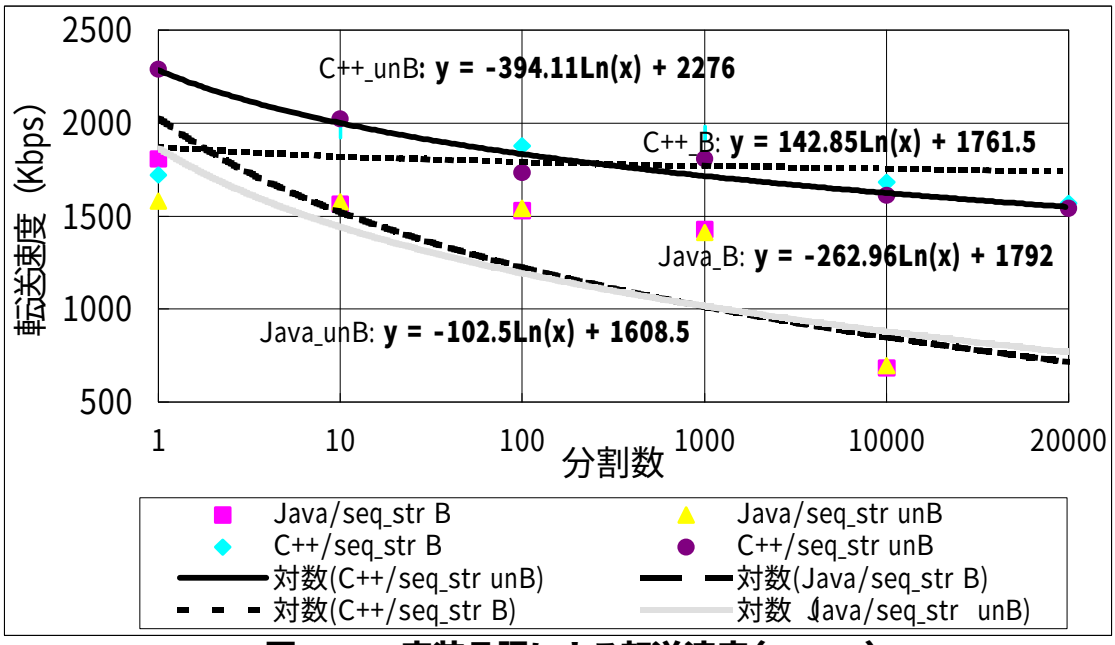
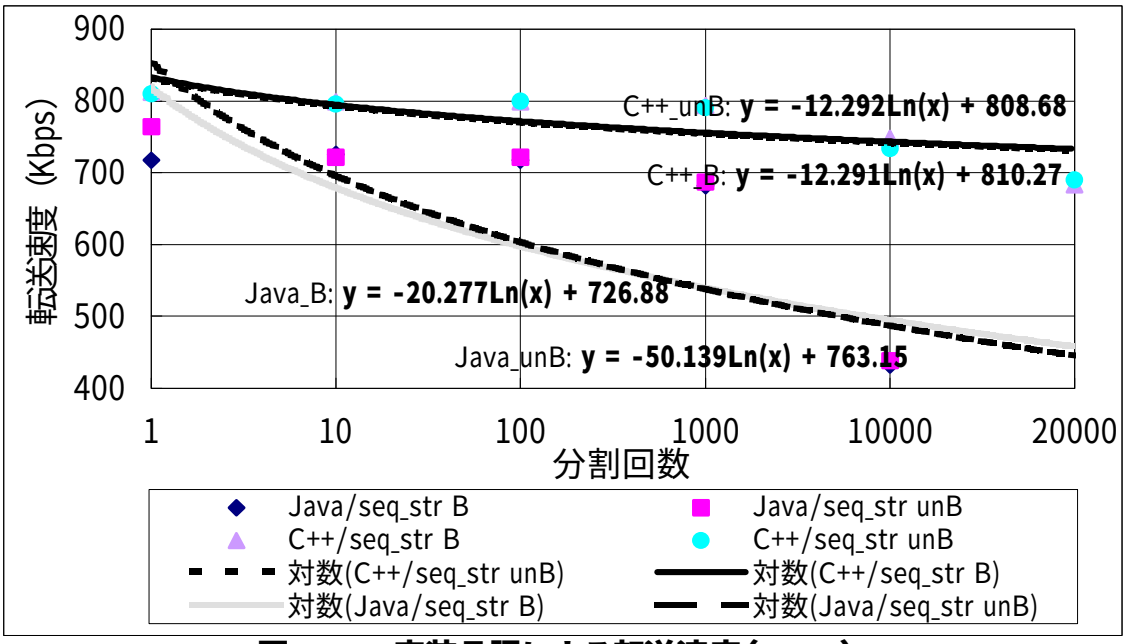


図-5.10 実装別平均転送時間（C++）



図‐5.13　実装言語による転送速度(100MB)



図‐5.14　実装言語による転送速度(10MB)

図‐5.9，図‐5.10 に実装言語別のデータ分割時における平均転送時間を，図‐5.11，図‐5.12 に LAN の性能別平均転送速度を示す．

実装言語別に見ると Java，C++とも 100MB の LAN の方が平均転送時間は早いことが分かる．また Java については 20000 分割した時，LAN の性能による転送速度の有意の差は無くなってきている．

LAN の性能別で見ると，100MB，10MB ともに C++の方が平均転送時間は速い．

図‐5.13，図‐5.14 に LAN 別に転送速度の，対数近似曲線をとったグラフを示す．
C++の分割要素の指定無しで他の転送速度とは異なり，転送速度の低下は見られない．

6章 考察

データ転送時間とは，クライアントがサーバに対してリクエストを出し，サーバからクライアントにデータを送信してクライアントがデータを受け終えるまでの所要時間とする．クライアント側でのファイルに書き出す時間と，サーバ側での処理時間は含まない．データ転送時間の平均をとったものを平均転送時間とする．また，得られた平均転送時間より，転送速度を得る．

6.1 データ分割によるデータ転送時間

図 5.1，図 5.2，図 5.3 に Sequence〈string〉型の分割の指定をする場合と，指定しない場合の比較を示す．

6.1.1 分割する最大要素数の指定

データ転送を行うときに，転送データの最大要素数をスタブが認知していないならば，最大要素数を認知するためにスタブで処理をする時間が必要となり，反対に転送データの最大要素数をあらかじめ認知していたならばスタブでの処理を必要としない分データ転送は向上すると考えられる．

しかし，図 5.1，図 5.2，図 5.3 より，転送データの最大要素数を指定することによってデータ転送速度が速くはならなかった．これは，スタブがサーバから送られてくるデータの最大要素数を指定しなくてもデータ転送時間に影響を与えないことを意味している．

6.1.2 1000 分割から 20000 分割までのデータ転送時間

図 5.1，図 5.2，図 5.3 において，転送データの分割数が 1000 を超えてから LAN の性能による転送速度の差が徐々に低下している．分割数が 1000 分割を超えると転送する 1 つのデータ要素があまりに小さいため LAN の性能に依存しなくなる．すなわち，スタブとスケルトン間のデータ転送時間よりも，スタブへアクセスする時間の影響が大きくなっているからと考えられる．

6.1.3 1000 分割までのデータ転送時間

転送データの分割数が 1000 までは，スタブへアクセスする時間よりも LAN の性能によるデータ転送速度の違いがよくでている．これは，1 回に転送するデータ量が大きいため

にデータ転送速度は LAN の性能に左右され，スタブへアクセスする時間よりもスタブとスケルトン間のデータ転送速度の影響が大きくなっているからと考えられる．

Sequence 型を使用して一括でデータ転送を行う場合には，データの最大要素数を指定せずに送れば良い.

分割する1つの要素のデータサイズが小さすぎると LAN の性能を生かせなくなる．よって，データの大きさによって適切な最大分割要素数を考える必要がある.

6.2 Sequence 型と String 型でのデータ転送時間

図 5.4，図 5.6，図 5.8 に Sequence〈string〉型と String 型のデータ転送の比較を示す．ここでは要素指定のしていない Sequence〈string〉型を使用する．また，図 5.5，図 5.7 に Sequence〈char〉型のデータ転送速度を示す．しかし，Sequence〈char〉型は，データ量 150byte までしか動作しなかったので，今回の考察対象からは省くことにした．

6.2.1 Sequence 型と String 型の比較

Sequence 型は，１度スタブにアクセスを行ったら後は１度にデータを送ることが可能となり，通信プロトコルには依存しない．そのため，通信プロトコルにデータ転送の方法が依存する String 型よりも時間をかけずにデータ転送を行えると考えられる．

しかし，図-5.4 から Sequence 型は String 型よりも平均転送時間が増大している．また，図 5.6 や図‐5.8 からもデータ転送速度は，Sequence 型の方が String 型よりも転送速度が遅い．Sequence〈string〉型では，可変長データ型のためにマーシャリング／アンマーシャリングにおいて各文字列に対するメモリの確保を行う必要があることや，Sequence 型は要素として複雑な型を持つことができる．例えばその型が構造体ならば識別するための処理がマーシャリング／アンマーシャリングにおいて行われる．よって，Sequence 型のデータ転送では，マーシャリング／アンマーシャリングで時間がかかっているため String 型よりデータ転送に時間がかかったと考えられる．

6.2.2 LAN の性能

図‐5.6，図‐5.8 で LAN の性能に伴ってデータ転送速度はおよそ２倍に高くなっている．これは，
マーシャリング／アンマーシャリングにかかる時間よりもスタブとスケルトン間でのデータ転送時間が大きな影響を与えていると考えられる．

構造体のような複雑なデータを転送する場合には Sequence 型が適しているが，基本的なデータを転送する場合に Sequence 型を用いるとマーシャリング／アンマーシャリングの時間が大きくかかわる．そのため，基本的なデータを転送する場合には，通信プロトコルを考慮し適切な型を選択する必要がある．

6.3 実装言語の違いによるデータ転送時間

図‐5.11，図‐5.12，図‐5.13，図‐5.14 では実装言語別で LAN の性能別による比較を示す．
また，図-5.9，図-5.10 では実装言語別のデータ分割時の平均転送時間を示す．

6.3.1 実装言語が Java 言語の場合

図-5.11，図-5.12 より，分割数が 1 から 1000 までは，実装言語が Java 言語や C++言語のどちらであってもデータ転送時間はほぼ同じであるが，1000 を超えると実装言語が Java 言語である方は急激に平均データ転送時間は増加する．これは，分割数が 1000 回を超えるとスタブとスケルトン間の転送時間よりも Java で実装したスタブでの処理に時間がかかっていると考えられる．
Java 言語は，インタプリタ型言語であるからプログラムの実行を仮想マシンである JVM(Java Virtual Mchine)を用いて行うために，コンパイラ型言語である C++言語が直接 CPU でプログラムを実行するよりも実行速度が遅いことが原因である．

6.3.2 実装言語が C++言語の場合

図-5.10 より，分割数を 1 から 20000 まで増やしても，データ転送時間はほぼ同じである．これは，実装言語が C++言語の場合はコンパイラ型言語であるために実行速度の低下はないためと考えられる．

6.3.3 Java 言語と C++言語

C++言語の場合は，分割数を 1 から 20000 まで増やしてもデータ転送時間はほぼ同じであるが，Java 言語の場合は転送データの分割数が 1000 を超えてから LAN の性能による転送速度の差が徐々に減少している．分割数が 1000 分割を超えると転送する１つのデータ要素があまりに小さいため LAN の性能に依存しなくなる．すなわち，スタブとスケルトン間のデータ転送時間よりも，スタブへアクセスする時間の影響が大きくなっているからと考えられる．しかし，C++言語では分割数を増やしても言語実行速度が速いために平均転送時間への影響は少ない．

速度の観点から見ると Java 言語を用いると処理速度が遅いため，C++言語を用いた方がデータ転送を速く行うのに有効である．

7章 まとめ

本研究では，分散オブジェクト環境 CORBA を用いたときにデータを適切に転送する方法を検討した．

データを分割の指定をして転送するべきかについては，どちらもデータ転送速度は変わらないため指定しなくてもよい．分割が必要な場合は，分割する１つの要素のサイズが小さ過ぎると LAN の性能が生かせないので，転送するデータのサイズによって適切な最大要素数を考えた上で分割する必要がある．

構造体のような複雑なデータを転送する場合は，通信プロトコルに依存しない Sequence 型が適している．しかし，単純なデータ型を転送するときには Sequence 型を使用するとマーシャリング／アンマーシャリングの時間が大きい．よって，単純なデータを転送する場合は，通信プロトコルを考慮し適切な型を選ばなくてはいけない．

分散オブジェクト環境を用いてデータの転送を行う場合は，データの内容を一括して送るべきか，通信プロトコル上でどのように扱われるか，とりうるプロトコルと転送するデータ条件との検討を行った上でデータ転送の方法を選択し行うことが重要である．

参考文献

[1] R. Orfali and D. Harkey, *Client/ Server Programming with Java and CORBA*, John Wiley & Sons,Inc, 1997［並河英二ほか(訳), Java & CORBA C/S プログラミング, 日経 BP 社, 1997］.

[2] R. Otte and P. Patrick and M. Roy, *Understanding CORBA: The Common Object Request Broker Architecture*, Prentice Hall PTR, 1996［MISCO オブジェクト指向研究会(訳), 分散オブジェクト指向 CORBA, プレンティスホール出版, 1996］.

[3] J. Rosenberger, *Sam's Teach Yourself CORBA in 14days*, SAMS-PROGRAMMING an imprint of Macmillan Computer Publishing USA, 1998［オブジェクト指向研究会(訳), 例題で学ぶ分散オブジェクト指向 CORBA, プレンティスホール出版, 1998］.

[4] 松野良蔵, *Java+CORBA 分散オブジェクトシステム構築*, 翔泳社, 1999.

[5] 青山幹雄ほか, コンポーネントウェア, 共立出版, 1998.

[6] Dr. T. J. Mowbray and R. C. Malveau, *CORBA Design Patterns*, John Wiley & Sons, 1997[大谷真ほか(訳), CORBA Design Pattern-アーキテクチャからプログラミングまで-, IDG コミュニケーションズ, 1998]

[7] Dirk Slama, Jason Garbis and Perry Russel, *Enterprise CORBA*, Prentice-Hall, 1999[(株)東洋情報システム+NTT 情報流通プラットフォーム研究所(訳), Enterprise CORBA-CORBA による実用システムの設計ガイドライン-, ピアソン・エデュケーション, 1999].

[8] 小野沢博文, *分散オブジェクト指向技術 CORAB*, ソフト・リサーチ・センター, 1996.

[9] J. Farley, *Java Distributed Computing*, O'Reilly & Associates, 1998［豊福剛(訳), Java 分散コンピューティング, オライリー･ジャパン, 1998］.

[10] 神吉達郎, *Java プログラミング入門*, オーム社, 1996.

[11] 山地秀美, はじめての VisualC++5.0, 技術評論社, 1998.

[12] 山下浩, 黒羽裕章, 黒岩健太郎, *C++プログラミングスタイル*, オーム社, 1992.

[13] 小池慎一, 森博, 山住富也, *改訂新版はじめての TURBO C+＋*, 技術評論社, 1994.

[14] MISCO オブジェクト指向研究会, オブジェクトモデリング表記法ガイド, プレンティスホール出版, 1998.

付録　データ転送のプログラムソース

(1) データ分割 (分割要素指定あり, 10 分割)

Java／Sequence<string>型
IDL プログラムソース

```
module ODsample{
        interface  seqtest{
                typedef sequence<string,10>   sampleseq;
                        //分割要素の指定
                sampleseq op1(in string str1);
        };
};
```

クライアントプログラムソース

```
import java.lang.*;
import java.awt.*;
import java.io.*;

public class RemoteCall {
         long TUSINstartTime = System.currentTimeMillis();
        public void initialize() throws CORBA.Exception{
                try {
                        odlib = new ODLib();
                        odlib.initialize();
                        long NameStartTime = System.currentTimeMillis();
                        CORBA.Object obj = odlib.getobj("ODsample::seqtest");
                        long NameStopTime = System.currentTimeMillis();
                        target = ODsample.seqtest_var.narrow(obj);
                        System.out.println("NamingService    time"    +    ((NameStopTime    -
NameStartTime)) + " msecs");
                }
                catch( CORBA.Exception e ){
                        throw(e);
                }
        }
        long TUSINstopTime = System.currentTimeMillis();
        void call( Box in_box, Box out_box, Box inout_box,
                                Box ret_box, Label status )
        {
                CORBA.String str_in, tmp_str;
              java.lang.String str;
              java.lang.String data = "";
              java.lang.String cmt = " data is wrong.";
```

```
                try {
                  data = "In";
                  str = in_box.get_text();
                  if( str.length() <= 0 ) {
                            status.setText(data + cmt);
                            return;
                   }
                   str_in = new CORBA.String(str);
                   ODsample.seqtestPackage.sampleseq ret_seq;
                   long startTime = System.currentTimeMillis(); //startTime
                            ret_seq = target.op1(str_in);
                   long startTime2 = System.currentTimeMillis(); //startTime
                   try{
                     PrintWriter dos = new PrintWriter(new BufferedWriter(new
FileWriter("file.txt")));
                     for(int j=0;j<10;j++){   //  分割数分データのファイルへの書き込み
                      dos.write(ret_seq.value[j].value);
                     }
                     dos.close();
                     }catch(Exception e){System.out.println(e);}
                     long stopTime = System.currentTimeMillis();

                   System.out.println("Tusinnro Time = " + ((TUSINstopTime - TUSINstartTime)) + "
msecs");
                   System.out.println("Transfer Time = " + ((startTime2 - startTime)) + " msecs");
                    System.out.println("Client Time = " + ((stopTime - startTime2)) + " msecs");
                    long TUSINROtime = TUSINstopTime - TUSINstartTime;
                    long Transfetime = startTime2 - startTime;
                    long ClientTime = stopTime - startTime2;
                    String turo = new String("通信路時間= ");
                    String   str_b = new String();
                try{
                     FileOutputStream fos = new FileOutputStream("data.txt",true);
                     DataOutputStream dos = new DataOutputStream(fos);
                     PrintStream pos = new PrintStream(dos);

                       pos.print("データ量= ");
                       pos.print(str);
                       pos.print("        ¥t");
                       pos.print(TUSINROtime);
                       pos.print("¥t");
                       pos.print(Transfetime);
                       pos.print("        ¥t");
                       pos.print(ClientTime);
                       pos.print("¥n");
                       str_b = str;

                      dos.close();
                      fos.close();
                      System.out.println("Next");
                      }catch(Exception e){System.out.println(e);}
                }
                            catch(NumberFormatException e){
                       status.setText(data + cmt);
                       return;
                }
                catch( CORBA.Exception err){
                            status.setText("op1 error");
                            return;
                }
```

```
                status.setText("");
        }
        static ODLib odlib;
        static ODsample.seqtest target;
}
```

サーバプログラムソース

```
#include <stdlib.h>
#include <windows.h>
#include <string.h>
#include <fstream.h>
#include "orb_cplus.h"
#include <time.h>       // 時間計測用
#include <sys¥timeb.h>// 時間計測用
#include "simple.H"
#include "benchtime.h"
#include <iostream>

/* ---------------------------------------------------------------- */
static      CORBA::ORB_ptr                                  orb;
static      CORBA::BOA_ptr                                  boa;
static      CORBA::Environment_ptr                  env;
static      FJ::ImplementationRep_ptr       impl_rep;
static      CORBA::ImplementationDef_ptr    impl;
static      CORBA::Object_ptr                       o;
/* ---------------------------------------------------------------- */
char fn[2100000];      //分割時の 1 要素サイズ
FILE *fname;
HighResTimer timer;
ODsample::seqtest::sampleseq*
ODsample_seqtest_impl::op1(
     const CORBA::Char*       str1,
           CORBA::Environment & )

           throw( CORBA::Exception )
{
timer.startCounter();

        if((fname=fopen(str1,"r"))==NULL){
            printf(" ファイルのオープンに失敗 ¥n");
           return(0);
            }
        if(fgets(fn,210000,fname)==NULL){
            printf("ファイルの取り込みの失敗¥n");
        }
        fclose(fname);
        CORBA::String_var *r = ODsample::seqtest::sampleseq::allocbuf(10);
                                                                    //分割要素サイズ
                for(int i=0;i<10;i++){      //分割数分のデータ読み込み
                r[i] = fn;
                }
          ODsample::seqtest::sampleseq_ptr ret;
          ret = new ODsample::seqtest::sampleseq(10,r,CORBA_TRUE);

          int tm = timer.getCounter();
        cout << "New Server time is " << tm/1000 << "ms" << endl;
        return(ret);
```

```
}
/* ------------------------------------------------------------------------ */

int
main( int argc, char *argv[] )
{
        int         current_argc = argc;
        char        buf[128];

        env = new CORBA::Environment;

        try {
                orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, *env );

                boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, *env );

                o = orb->resolve_initial_references(
                                CORBA_ORB_ObjectId_ImplementationRepository,
                                *env );
                impl_rep = FJ::ImplementationRep::_narrow( o );

                o = impl_rep->lookup_id( _IMPL_ODsample_seqtest, *env );
                impl = CORBA::ImplementationDef::_narrow( o );

                boa->impl_is_ready( impl, *env );
        }
        catch ( CORBA::SystemException se ) {
                printf("SystemException raised!¥n");
                printf("exception-id = %s¥n", se.id());
                ExitProcess(0);
        }
        catch ( CORBA::Exception e ) {
                printf("Exception raised!¥n");
                printf("exception-id = %s¥n", e.id());
                ExitProcess(0);
        }
        CORBA::release( env );
        CORBA::release( boa );
        CORBA::release( orb );

        ExitThread(0);
        return(0);
}
```

C++／Sequence<string>型
IDL プログラムソース

```
module ODsample{
        interface       seqtest{
                typedef sequence<string,10>     sampleseq;
                        //分割要素の指定
                sampleseq op1(in string seq1);
        };
};
```

クライアントプログラムソース

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <string.h>
#include  <iostream.h>
#include  <windows.h>
#include <time.h>       // 時間計測用
#include <sys¥timeb.h>// 時間計測用
#include  "orb_cplus.h"
#include  "CosNaming_cplus.h"
#include "benchtime.h"// 時間計測用
#include  "simple.H"

CORBA::Environment_ptr     env;

/* ---------------------------------------------------------------------- */
void
print_seqvar( const ODsample::seqtest::sampleseq &seq )
{
            char *ftxt;
          FILE *fn;
            ftxt = "file.txt";
            fn=fopen(ftxt,"w");
        for( int i = 0; i < 10; i++ ) {
                  //分割数分のデータ書き込み
            fputs(seq[i],fn);
        }
                fclose(fn);
}

void
timer_w(int timer_c){
        FILE *abc;
        char *txt;
        txt="data.txt";
        abc=fopen(txt,"a");
    fprintf(abc,"%d¥n",timer_c);
    printf("¥n");
    fclose(abc);
}
```

```
HighResTimer timer;
int
main( int argc, char *argv[] )
{
        CORBA::ORB_ptr          orb;
        CORBA::BOA_ptr          boa;

        int     current_argc = argc;

        env = new CORBA::Environment;

        try {
                orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, *env );

                boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, *env );

                CORBA::Object_ptr obj = orb->resolve_initial_references(
                                                        CORBA_ORB_ObjectId_NameService,
                                                        *env );

                CosNaming::NamingContext_ptr NamingContext_obj =
                                CosNaming::NamingContext::_narrow( obj );
                CORBA::release( obj );

                CosNaming::NameComponent_var *name_component_var =
                        CosNaming::Name::allocbuf(1);
                name_component_var[0]->id =
                                (const CORBA::Char *)"ODsample::seqtest";
                name_component_var[0]->kind = (const CORBA::Char *)"";
                CosNaming::Name_ptr name =
                        new CosNaming::Name(1,1,name_component_var, CORBA_TRUE);
                obj = NamingContext_obj->resolve( *name, *env );
                CORBA::release( NamingContext_obj );
                delete name;


                ODsample::seqtest_ptr       ap = ODsample::seqtest::_narrow( obj );
                ODsample::seqtest_var       av = ap;

                char str[10];
              printf("ファイル名＞");
                scanf("%s",str);
              CORBA::Char* seq1;

                strcpy(seq1,str);
                int k=0;
            for(int j=0;j<12;j++){          //データを取る回数分繰り返す
               ODsample::seqtest::sampleseq_ptr ret;

                timer.startCounter();
                ret = av->op1(seq1,*env);
                k = j+1;
                printf("%d",k);
                int tm = timer.getCounter();

                cout << "New Server time is " << tm/1000 << "ms" << endl;
                print_seqvar(*ret);

              int timeC = tm/1000;
```

```
                timer_w(timeC);

                  CORBA::string_free( seq1 );

                delete      ret;
              }
          }
          catch( CORBA::SystemException          &se ) {
                        printf( "SystemException raised!¥n" );
                        printf( "exception-id = %s¥n", se.id() );
                        ExitProcess(0);
          }
          catch( CORBA::Exception      &e ) {
                        printf( "Exception raised!¥n" );
                        printf( "exception-id = %s¥n", e.id() );
                        ExitProcess(0);
          }
          CORBA::release( env );
          CORBA::release( boa );
          CORBA::release( orb );
          return (0);
}
```

サーバプログラムソース

```
#include <stdlib.h>
#include <windows.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include <time.h>       // 時間計測用
#include <sys¥timeb.h>// 時間計測用
#include "orb_cplus.h"
#include "benchtime.h"// 時間計測用
#include "simple.H"
/* ---------------------------------------------------------------------- */
static      CORBA::ORB_ptr                                          orb;
static      CORBA::BOA_ptr                                          boa;
static      CORBA::Environment_ptr                          env;
static      FJ::ImplementationRep_ptr               impl_rep;
static      CORBA::ImplementationDef_ptr            impl;
static      CORBA::Object_ptr                                       o;
/* ---------------------------------------------------------------------- */
int k=0;
FILE *fname;
char fn[210000];          //分割時の 1 要素サイズ
HighResTimer timer;
ODsample::seqtest::sampleseq*
ODsample_seqtest_impl::op1(
          const CORBA::Char*            seq1,
          CORBA::Environment & )
          throw( CORBA::Exception )
{
timer.startCounter();
                  printf("¥n");
                  printf("10 分割");
                printf( "seq1 = [%s]¥n", seq1 );
          k=k++;
                  printf("%d",k);
        CORBA::String_var *r = ODsample::seqtest::sampleseq::allocbuf(10);
```

```
                                                                    //分割要素サイズ
        if((fname=fopen(seq1,"r"))==NULL){
            printf("  ファイルのオープンに失敗 ¥n");
        }
        if(fgets(fn, 210000 ,fname)==NULL){
              printf("読み込み失敗");
          }

          for(int i=0;i<10;i++){          //分割数分のデータの読み込み
              r[i] = fn;
          }
        ODsample::seqtest::sampleseq* ret;
          ret = new ODsample::seqtest::sampleseq(10,r,CORBA_TRUE);

          int tm = timer.getCounter();

          fclose(fname);
          cout << "New Server time is " << tm/1000 << "ms" << endl;
        return(ret);
}
/* ---------------------------------------------------------------------- */
int
main( int argc, char *argv[] )
{
          int         current_argc = argc;
          env = new CORBA::Environment;

          try {
                    orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, *env );
                    boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, *env );
                    o = orb->resolve_initial_references(
                              CORBA_ORB_ObjectId_ImplementationRepository, *env );
                    impl_rep = FJ::ImplementationRep::_narrow( o );

                    o = impl_rep->lookup_id( _IMPL_ODsample_seqtest, *env );
                    impl = CORBA::ImplementationDef::_narrow( o );

                    boa->impl_is_ready( impl, *env );
          }
          catch ( CORBA::SystemException se ) {
                    printf("SystemException raised!¥n");
                    printf("exception-id = %s¥n", se.id());
                    ExitProcess(0);
          }
          catch ( CORBA::Exception e ) {
                    printf("Exception raised!¥n");
                    printf("exception-id = %s¥n", e.id());
                    ExitProcess(0);
          }
          CORBA::release( env );
          CORBA::release( boa );
          CORBA::release( orb );

          ExitThread(0);
          return(0);
}
```

(2) データ分割 (分割要素指定なし, 10 分割)

Java／Sequence<string>型

IDL プログラムソース

```
module ODsample{
        interface  seqtest{
                typedef sequence<string>       sampleseq;
                sampleseq op1(in string str1);
        };
};
```

クライアントプログラムソース

```
import java.lang.*;
import java.awt.*;
import java.io.*;

public class RemoteCall {
        long TUSINstartTime = System.currentTimeMillis();
      public void initialize() throws CORBA.Exception{
            try {
                  odlib = new ODLib();
                  odlib.initialize();
                  long NameStartTime = System.currentTimeMillis();
                  CORBA.Object obj = odlib.getobj("ODsample::seqtest");
                  long NameStopTime = System.currentTimeMillis();
                  target = ODsample.seqtest_var.narrow(obj);
                  System.out.println("NamingService    time"    +    ((NameStopTime    -
NameStartTime)) + " msecs");
            }
            catch( CORBA.Exception e ){
                  throw(e);
            }
      }
      long TUSINstopTime = System.currentTimeMillis();
      void call( Box in_box, Box out_box, Box inout_box,
                        Box ret_box, Label status )
      {
            CORBA.String str_in, tmp_str;
          java.lang.String str;
          java.lang.String data = "";
          java.lang.String cmt = " data is wrong.";

          try {
            data = "In";
            str = in_box.get_text();
            if( str.length() <= 0 ) {
                  status.setText(data + cmt);
                  return;
             }
```

```
                str_in = new CORBA.String(str);

                ODsample.seqtestPackage.sampleseq ret_seq;
                long startTime = System.currentTimeMillis(); //startTime
                        ret_seq = target.op1(str_in);
                long startTime2 = System.currentTimeMillis(); //startTime

                try{
                  PrintWriter dos = new PrintWriter(new BufferedWriter(new
FileWriter("file.txt")));

                   for(int j=0;j<10;j++){      //分割数分のファイルへのデータ書き込み
                       dos.write(ret_seq.value[j].value);
                   }
                  dos.close();
                  }catch(Exception e){System.out.println(e);}
                  long stopTime = System.currentTimeMillis();

                System.out.println("Tusinnro Time = " + ((TUSINstopTime - TUSINstartTime)) + "
msecs");
                System.out.println("Transfer Time = " + ((startTime2 - startTime)) + " msecs");
                 System.out.println("Client Time = " + ((stopTime - startTime2)) + " msecs");
                 long TUSINROtime = TUSINstopTime - TUSINstartTime;
                 long Transfetime = startTime2 - startTime;
                 long ClientTime = stopTime - startTime2;
                 String turo = new String("通信路時間= ");
                 String   str_b = new String();
            try{
                 FileOutputStream fos = new FileOutputStream("data.txt",true);
                 DataOutputStream dos = new DataOutputStream(fos);
                 PrintStream pos = new PrintStream(dos);

                   pos.print("データ量= ");
                   pos.print(str);
                   pos.print("        ¥t");
                   pos.print(TUSINROtime);
                   pos.print("¥t");
                   pos.print(Transfetime);
                   pos.print("        ¥t");
                   pos.print(ClientTime);
                   pos.print("¥n");
                   str_b = str;

                  dos.close();
                  fos.close();
                  System.out.println("Next");
                  }catch(Exception e){System.out.println(e);}
            }
                        catch(NumberFormatException e){
                    status.setText(data + cmt);
                    return;
            }
            catch( CORBA.Exception err){
                    status.setText("op1 error");
                    return;
            }
            status.setText("");
        }
        static ODLib odlib;
        static ODsample.seqtest target;
```

```
}
```

サーバプログラムソース

```
#include <stdlib.h>
#include <windows.h>
#include <string.h>
#include <fstream.h>
#include "orb_cplus.h"
#include <time.h>       // 時間計測用
#include <sys¥timeb.h>// 時間計測用
#include "simple.H"
#include "benchtime.h"
#include <iostream>

/* ---------------------------------------------------------------------- */

static      CORBA::ORB_ptr                                          orb;
static      CORBA::BOA_ptr                                          boa;
static      CORBA::Environment_ptr                          env;
static      FJ::ImplementationRep_ptr               impl_rep;
static      CORBA::ImplementationDef_ptr            impl;
static      CORBA::Object_ptr                               o;

/* ---------------------------------------------------------------------- */
char fn[210000];      //分割時の 1 要素サイズ
FILE *fname;
HighResTimer timer;
ODsample::seqtest::sampleseq*
ODsample_seqtest_impl::op1(
     const CORBA::Char*       str1,
           CORBA::Environment & )
           throw( CORBA::Exception )
{
timer.startCounter();

          if((fname=fopen(str1,"r"))==NULL){
              printf("  ファイルのオープンに失敗  ¥n");
          }
          if(fgets(fn,210000,fname)==NULL){
              printf("ファイルの取り込みの失敗¥n");
          }
          fclose(fname);
          CORBA::String_var *r = ODsample::seqtest::sampleseq::allocbuf(10);
                                                                  //分割要素サイズ
            for(int i=0;i<10;i++){          //分割数分のデータ読み込み
                        r[i] = fn;
            }
            ODsample::seqtest::sampleseq_ptr ret;
            ret = new ODsample::seqtest::sampleseq(10,10,r,CORBA_TRUE);
                                                  //分割要素数の指定
            int tm = timer.getCounter();
          cout << "New Server time is " << tm/1000 << "ms" << endl;
          return(ret);
}
```

```
/* ------------------------------------------------------------------------ */

int
main( int argc, char *argv[] )
{
        int         current_argc = argc;
        char        buf[128];

        env = new CORBA::Environment;

        try {
                orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, *env );

                boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, *env );

                o = orb->resolve_initial_references(
                                CORBA_ORB_ObjectId_ImplementationRepository,
                                *env );
                impl_rep = FJ::ImplementationRep::_narrow( o );

                o = impl_rep->lookup_id( _IMPL_ODsample_seqtest, *env );
                impl = CORBA::ImplementationDef::_narrow( o );

                boa->impl_is_ready( impl, *env );
        }
        catch ( CORBA::SystemException se ) {
                printf("SystemException raised!¥n");
                printf("exception-id = %s¥n", se.id());
                ExitProcess(0);
        }
        catch ( CORBA::Exception e ) {
                printf("Exception raised!¥n");
                printf("exception-id = %s¥n", e.id());
                ExitProcess(0);
        }

        CORBA::release( env );
        CORBA::release( boa );
        CORBA::release( orb );

        ExitThread(0);
        return(0);
}
```

C++／Sequence<string>型

IDL プログラムソース

```
module ODsample{
        interface       seqtest{
                typedef sequence<string> sampleseq;
                sampleseq op1(in string seq1);
        };
};
```

クライアントプログラムソース

```
#include  <stdio.h>
#include  <stdlib.h>
#include  <string.h>
#include  <iostream.h>
#include  <windows.h>
#include <time.h>       // 時間計測用
#include <sys¥timeb.h>// 時間計測用
#include  "orb_cplus.h"
#include  "CosNaming_cplus.h"
#include "benchtime.h"// 時間計測用
#include  "simple.H"

CORBA::Environment_ptr     env;
/* ------------------------------------------------------------------------ */
void
print_seqvar( const ODsample::seqtest::sampleseq &seq )
{
            char *ftxt;
           FILE *fn;
            ftxt = "file.txt";
            fn=fopen(ftxt,"w");
        for( int i = 0; i < 10; i++ ) {
                  //分割数分のデータ書き込み
            fputs(seq[i],fn);
        }
                fclose(fn);
}

void
timer_w(int timer_c){
          FILE *abc;
          char *txt;
          txt="data.txt";
          abc=fopen(txt,"a");
     fprintf(abc,"%d¥n",timer_c);
     printf("¥n");
     fclose(abc);
}

HighResTimer timer;
int
```

```
main( int argc, char *argv[] )
{
        CORBA::ORB_ptr          orb;
        CORBA::BOA_ptr          boa;

        int     current_argc = argc;

        env = new CORBA::Environment;

        try {
                orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, *env );

                boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, *env );

                CORBA::Object_ptr obj = orb->resolve_initial_references(
                                                        CORBA_ORB_ObjectId_NameService,
                                                        *env );

                CosNaming::NamingContext_ptr NamingContext_obj =
                                CosNaming::NamingContext::_narrow( obj );
                CORBA::release( obj );

                CosNaming::NameComponent_var *name_component_var =
                        CosNaming::Name::allocbuf(1);
                name_component_var[0]->id =
                                (const CORBA::Char *)"ODsample::seqtest";
                name_component_var[0]->kind = (const CORBA::Char *)"";
                CosNaming::Name_ptr name =
                        new CosNaming::Name(1,1,name_component_var, CORBA_TRUE);
                obj = NamingContext_obj->resolve( *name, *env );
                CORBA::release( NamingContext_obj );
                delete name;


                ODsample::seqtest_ptr           ap = ODsample::seqtest::_narrow( obj );
                ODsample::seqtest_var           av = ap;

                char str[10];
              printf("ファイル名＞");
                scanf("%s",str);
              CORBA::Char* seq1;

                strcpy(seq1,str);
                int k=0;
            for(int j=0;j<12;j++){              //データを取る回数分繰り返す
               ODsample::seqtest::sampleseq_ptr ret;

                timer.startCounter();
                ret = av->op1(seq1,*env);
                k = j+1;
                printf("%d",k);
                int tm = timer.getCounter();

                cout << "New Server time is " << tm/1000 << "ms" << endl;
                print_seqvar(*ret);

              int timeC = tm/1000;
              timer_w(timeC);

                CORBA::string_free( seq1 );
```

```
                delete      ret;
            }
        }
        catch( CORBA::SystemException       &se ) {
                printf( "SystemException raised!¥n" );
                printf( "exception-id = %s¥n", se.id() );
                ExitProcess(0);
        }
        catch( CORBA::Exception     &e ) {
                printf( "Exception raised!¥n" );
                printf( "exception-id = %s¥n", e.id() );
                ExitProcess(0);
        }
        CORBA::release( env );
        CORBA::release( boa );
        CORBA::release( orb );
        return (0);
}
```

サーバプログラムソース

```
#include <stdlib.h>
#include <windows.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include <time.h>       // 時間計測用
#include <sys¥timeb.h>// 時間計測用
#include "orb_cplus.h"
#include "benchtime.h"// 時間計測用
#include "simple.H"
/* ---------------------------------------------------------------- */
static      CORBA::ORB_ptr                                  orb;
static      CORBA::BOA_ptr                                  boa;
static      CORBA::Environment_ptr                  env;
static      FJ::ImplementationRep_ptr           impl_rep;
static      CORBA::ImplementationDef_ptr        impl;
static      CORBA::Object_ptr                       o;
/* ---------------------------------------------------------------- */
int k=0;
FILE *fname;
char fn[210000];        //分割時の 1 要素サイズ
HighResTimer timer;
ODsample::seqtest::sampleseq*
ODsample_seqtest_impl::op1(
        const CORBA::Char*          seq1,
        CORBA::Environment & )
        throw( CORBA::Exception )
{
timer.startCounter();
                printf("¥n");
                printf("10 分割");
            printf( "seq1 = [%s]¥n", seq1 );
        k=k++;
                printf("%d",k);
      CORBA::String_var *r = ODsample::seqtest::sampleseq::allocbuf(10);
                                                              //分割要素サイズ
      if((fname=fopen(seq1,"r"))==NULL){
         printf(" ファイルのオープンに失敗 ¥n");
```

```
        }
        if(fgets(fn, 210000 ,fname)==NULL){
            printf("読み込み失敗");
        }

        for(int i=0;i<10;i++){          //分割数分のデータの読み込み
            r[i] = fn;
        }
        ODsample::seqtest::sampleseq* ret;
        ret = new ODsample::seqtest::sampleseq(10,10,r,CORBA_TRUE);
                                                    //分割要素数の指定
        int tm = timer.getCounter();

        fclose(fname);
        cout << "New Server time is " << tm/1000 << "ms" << endl;
        return(ret);
}
/* ------------------------------------------------------------------------ */
int
main( int argc, char *argv[] )
{
        int         current_argc = argc;
        env = new CORBA::Environment;

        try {
                orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, *env );
                boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, *env );
                o = orb->resolve_initial_references(
                        CORBA_ORB_ObjectId_ImplementationRepository, *env );
                impl_rep = FJ::ImplementationRep::_narrow( o );

                o = impl_rep->lookup_id( _IMPL_ODsample_seqtest, *env );
                impl = CORBA::ImplementationDef::_narrow( o );

                boa->impl_is_ready( impl, *env );
        }
        catch ( CORBA::SystemException se ) {
                printf("SystemException raised!¥n");
                printf("exception-id = %s¥n", se.id());
                ExitProcess(0);
        }
        catch ( CORBA::Exception e ) {
                printf("Exception raised!¥n");
                printf("exception-id = %s¥n", e.id());
                ExitProcess(0);
        }
        CORBA::release( env );
        CORBA::release( boa );
        CORBA::release( orb );

        ExitThread(0);
        return(0);
}
```

(3) String 型のプログラムソース（Java）

IDL プログラムソース

```
module ODsample{
        interface       stringtest{
                string  op1(in string str1);
        };
};
```

クライアントプログラムソース

```
import java.lang.*;
import java.awt.*;
import java.io.*;
public class RemoteCall {
         long TUSINstartTime = System.currentTimeMillis();
        public void initialize() throws CORBA.Exception{
                try {
                        odlib = new ODLib();
                        odlib.initialize();
                        long NameStartTime = System.currentTimeMillis();
                        CORBA.Object obj = odlib.getobj("ODsample::stringtest");
                        long NameStopTime = System.currentTimeMillis();
                        target = ODsample.stringtest_var.narrow(obj);
                         System.out.println("NamingService time" + ((NameStopTime -
NameStartTime)) + " msecs");
                }
                catch( CORBA.Exception e ){
                        throw(e);
                }
        }
        long TUSINstopTime = System.currentTimeMillis();
        public void call( Box in_box, Box out_box, Box inout_box,
                                Box ret_box, Label status ){
                                CORBA.String str_in, tmp_str;
              java.lang.String str;
              java.lang.String data = "";
              java.lang.String cmt = " data is wrong.";

              try {
                data = "In";
                str = in_box.get_text();
                if( str.length() <= 0 ) {
                        status.setText(data + cmt);
                        return;
                 }
                 str_in = new CORBA.String(str);
                CORBA.String str_ret = new CORBA.String();
                 long startTime = System.currentTimeMillis(); //startTime
                        str_ret = target.op1(str_in);
                 long startTime2 = System.currentTimeMillis(); //startTime
                 try{
                  PrintWriter dos = new PrintWriter(new BufferedWriter(new
```

```
FileWriter("file.txt")));
                        dos.write(str_ret.value);
                       dos.close();
                       }catch(Exception e){System.out.println(e);}
                       long stopTime = System.currentTimeMillis();

                    System.out.println("Tusinnro Time = " + ((TUSINstopTime - TUSINstartTime)) + "
msecs");
                    System.out.println("Transfer Time = " + ((startTime2 - startTime)) + " msecs");
                     System.out.println("Client Time = " + ((stopTime - startTime2)) + " msecs");
                     long TUSINROtime = TUSINstopTime - TUSINstartTime;
                     long Transfetime = startTime2 - startTime;
                     long ClientTime = stopTime - startTime2;


                try{
                     FileOutputStream fos = new FileOutputStream("data.txt",true);
                     DataOutputStream dos = new DataOutputStream(fos);
                     PrintStream pos = new PrintStream(dos);

                       pos.print("データ量= ");
                       pos.print(str);
                       pos.print(" ¥t");
                       pos.print(TUSINROtime);
                       pos.print(" ¥t");
                       pos.print(Transfetime);
                       pos.print("      ¥t");
                       pos.print(ClientTime);
                       pos.print("¥n");


                      dos.close();
                      fos.close();
                      System.out.println("Next");
                      }catch(Exception e){System.out.println(e);}
                }
                            catch(NumberFormatException e){
                        status.setText(data + cmt);
                        return;
                }
                catch( CORBA.Exception err){
                            status.setText("op1 error");
                            return;
                }
                status.setText("");
        }
        static ODLib odlib;
        static ODsample.stringtest target;
}
```

サーバプログラムソース

```
#include <stdlib.h>
#include <windows.h>
#include <string.h>
#include <fstream.h>
#include "orb_cplus.h"
#include <time.h>       // 時間計測用
#include <sys¥timeb.h>// 時間計測用
#include <sys¥types.h>
#include "simple.H"
#include "benchtime.h"
 #include <iostream>
/* ---------------------------------------------------------------------- */

static      CORBA::ORB_ptr                                      orb;
static      CORBA::BOA_ptr                                      boa;
static      CORBA::Environment_ptr                      env;
static      FJ::ImplementationRep_ptr           impl_rep;
static      CORBA::ImplementationDef_ptr        impl;
static      CORBA::Object_ptr                           o;

/* ---------------------------------------------------------------------- */
double startTime,stopTime;
struct _timeb timebuff;
char fn[2200000];
FILE *fname;
HighResTimer timer;
CORBA::Char*
ODsample_stringtest_impl::op1(
           const      CORBA::Char*      str1,
           CORBA::Environment & )
           throw( CORBA::Exception )
{
timer.startCounter();
          CORBA::Char*     ret;
                ret = CORBA::string_alloc(2200000);

          if((fname=fopen(str1,"r"))==NULL){
              printf("  ファイルのオープンに失敗 ¥n");
              strcpy(ret,"open err");
              return(ret);
              }
          if(fgets(fn,2200000,fname)==NULL){
              printf("ファイルの取り込みの失敗¥n");
          }
          fclose(fname);

          strcpy(ret,fn);

int tm = timer.getCounter();
cout << "New Server time is " << tm/1000 << "ms" << endl;
          return(ret);
}
/* ---------------------------------------------------------------------- */

int
```

```
main( int argc, char *argv[] )
{
        int         current_argc = argc;
        char        buf[128];

        env = new CORBA::Environment;

        try {
                orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, *env );

                boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, *env );

                o = orb->resolve_initial_references(
                                CORBA_ORB_ObjectId_ImplementationRepository,
                                *env );
                impl_rep = FJ::ImplementationRep::_narrow( o );

                o = impl_rep->lookup_id( _IMPL_ODsample_stringtest, *env );
                impl = CORBA::ImplementationDef::_narrow( o );

                boa->impl_is_ready( impl, *env );
        }
        catch ( CORBA::SystemException se ) {
                printf("SystemException raised!¥n");
                printf("exception-id = %s¥n", se.id());
                ExitProcess(0);
        }
        catch ( CORBA::Exception e ) {
                printf("Exception raised!¥n");
                printf("exception-id = %s¥n", e.id());
                ExitProcess(0);
        }

        CORBA::release( env );
        CORBA::release( boa );
        CORBA::release( orb );

        ExitThread(0);
        return(0);
}
```

(4) Sequence<char>型のソースプログラム(Java)

IDL プログラムソース

```
module ODsample{
        interface  seqtest{
                typedef sequence<char>        sampleseq;
                sampleseq op1(in string str1);
        };

};
```

クライアントプログラムソース

```
import java.lang.*;
import java.awt.*;
import java.io.*;

public class RemoteCall {
         long TUSINstartTime = System.currentTimeMillis();
        public void initialize() throws CORBA.Exception{
                try {
                        odlib = new ODLib();
                        odlib.initialize();
                        long NameStartTime = System.currentTimeMillis();
                        CORBA.Object obj = odlib.getobj("ODsample::seqtest");
                        long NameStopTime = System.currentTimeMillis();
                        target = ODsample.seqtest_var.narrow(obj);
                        System.out.println("NamingService time" + ((NameStopTime -
NameStartTime)) + " msecs");
                }
                catch( CORBA.Exception e ){
                        throw(e);
                }
        }
        long TUSINstopTime = System.currentTimeMillis();
        void call( Box in_box, Box out_box, Box inout_box,
                                Box ret_box, Label status )
        {
                CORBA.String str_in, tmp_str;
              java.lang.String str;
              java.lang.String data = "";
              java.lang.String cmt = " data is wrong.";

              try {
                data = "In";
                str = in_box.get_text();
                if( str.length() <= 0 ) {
                        status.setText(data + cmt);
                        return;
                 }
                 str_in = new CORBA.String(str);


                 ODsample.seqtestPackage.sampleseq ret_seq = new
ODsample.seqtestPackage.sampleseq(1000000);
                 long startTime = System.currentTimeMillis(); //startTime
```

```
                    ret_seq = target.op1(str_in);
                long startTime2 = System.currentTimeMillis(); //startTime
                try{
                 FileOutputStream fos = new FileOutputStream("file.txt");
                 DataOutputStream dos = new DataOutputStream(fos);
                 for(int i=0;i<ret_seq.value.length;i++){
                 dos.write(ret_seq.value[i].value);
                 }
                  dos.close();
                  fos.close();
                  }catch(Exception e){System.out.println(e);}
                  long stopTime = System.currentTimeMillis();

                System.out.println("Tusinnro Time = " + ((TUSINstopTime - TUSINstartTime)) + "
msecs");
                System.out.println("Transfer Time = " + ((startTime2 - startTime)) + " msecs");
                 System.out.println("Client Time = " + ((stopTime - startTime2)) + " msecs");
                 long TUSINROtime = TUSINstopTime - TUSINstartTime;
                 long Transfetime = startTime2 - startTime;
                 long ClientTime = stopTime - startTime2;
                 String turo = new String("通信路時間= ");
                 String   str_b = new String();
            try{
                 FileOutputStream fos = new FileOutputStream("data.txt",true);
                 DataOutputStream dos = new DataOutputStream(fos);
                 PrintStream pos = new PrintStream(dos);

                   pos.print("データ量= ");
                   pos.print(str);
                   pos.print("        ¥t");
                   pos.print(TUSINROtime);
                   pos.print("¥t");
                   pos.print(Transfetime);
                   pos.print("        ¥t");
                   pos.print(ClientTime);
                   pos.print("¥n");
                   str_b = str;

                  dos.close();
                  fos.close();
                  System.out.println("Next");
                  }catch(Exception e){System.out.println(e);}
            }
                        catch(NumberFormatException e){
                    status.setText(data + cmt);
                    return;
            }
            catch( CORBA.Exception err){
                    status.setText("op1 error");
                    return;
            }
            status.setText("");
        }
        static ODLib odlib;
        static ODsample.seqtest target;
}
```

サーバプログラムソース

```
#include <stdlib.h>
#include <windows.h>
#include <string.h>
#include <fstream.h>
#include "orb_cplus.h"
#include <time.h>       // 時間計測用
#include <sys¥timeb.h>// 時間計測用
#include "simple.H"
#include "benchtime.h"
#include <iostream>

/* ---------------------------------------------------------------------- */

static      CORBA::ORB_ptr                                      orb;
static      CORBA::BOA_ptr                                      boa;
static      CORBA::Environment_ptr                      env;
static      FJ::ImplementationRep_ptr           impl_rep;
static      CORBA::ImplementationDef_ptr        impl;
static      CORBA::Object_ptr                           o;

/* ---------------------------------------------------------------------- */
char fn[200000];
FILE *fname;
HighResTimer timer;
ODsample::seqtest::sampleseq*
ODsample_seqtest_impl::op1(
     const CORBA::Char*       str1,
           CORBA::Environment & )


           throw( CORBA::Exception )
{
timer.startCounter();

          if((fname=fopen(str1,"r"))==NULL){
              printf(" ファイルのオープンに失敗 ¥n");
              }
          if(fgets(fn,200000,fname)==NULL){
              printf("ファイルの取り込みの失敗¥n");
          }
          fclose(fname);
          CORBA::Char *r = ODsample::seqtest::sampleseq::allocbuf(1);
                        r = fn;
                        int fnsize=sizeof(fn)/sizeof(fn[0]);
                        int rsize=sizeof(r)/sizeof(r[0]);
                        int rpsize=sizeof(*r);
                        printf("fnsize= %d",fnsize);
                        printf("rsize= %d",rsize);
                        printf("rpsize= %d",rpsize);
            ODsample::seqtest::sampleseq_ptr ret;
            ret = new ODsample::seqtest::sampleseq(200000,200000,r,CORBA_TRUE);
            int tm = timer.getCounter();
          cout << "New Server time is " << tm/1000 << "ms" << endl;
          return(ret);
}
```

```
/* ------------------------------------------------------------------------ */

int
main( int argc, char *argv[] )
{
        int         current_argc = argc;
        char        buf[128];

        env = new CORBA::Environment;

        try {
                orb = CORBA::ORB_init( current_argc, argv, FJ_OM_ORBid, *env );

                boa = orb->BOA_init( current_argc, argv, CORBA_BOA_OAid, *env );

                o = orb->resolve_initial_references(
                                CORBA_ORB_ObjectId_ImplementationRepository,
                                *env );
                impl_rep = FJ::ImplementationRep::_narrow( o );

                o = impl_rep->lookup_id( _IMPL_ODsample_seqtest, *env );
                impl = CORBA::ImplementationDef::_narrow( o );

                boa->impl_is_ready( impl, *env );
        }
        catch ( CORBA::SystemException se ) {
                printf("SystemException raised!¥n");
                printf("exception-id = %s¥n", se.id());
                ExitProcess(0);
        }
        catch ( CORBA::Exception e ) {
                printf("Exception raised!¥n");
                printf("exception-id = %s¥n", e.id());
                ExitProcess(0);
        }

        CORBA::release( env );
        CORBA::release( boa );
        CORBA::release( orb );

        ExitThread(0);
        return(0);
}
```