愛知工業大学研究報告
第33号B　平成10年

Javaを用いた各種代理サーバの設計と実装支援

Making and support a proxy design in the Java.

深津 正芳[†]，　羽賀 隆洋[‡]

Masayoshi FUKATSU,　Takahiro HAGA

Absuract

We can connect to the Internet simply by many providers. It will give us only one connection. The one connection realizes that one computer is grown to the network. But, we have many computers. I hope that all compter that we have connect to network and be used. Though, I construct a making and support a proxy design in the Java.

1　はじめに

1995年秋、突然学内のLANと学外の全てのネットワーク的接続が断絶した。これは大学側、すなわち計算センターが、セキュリティ確保のため無条件な学内と学外の接続を制限したのである。しかも、この接続制限、いわゆるファイアーウォールの設置に関して、全てのパケットを通さないという事態にもかかわらず、前もってネットワークを利用している人達には十分な説明はなされていなかった。

当時、他大学のメンバーと協力し、ネットワークに関する設定などを調べていた我々は、突然全てのサービスが使用できなくなった自体に非常に困惑することになる。

事態を把握できない我々は、まずネットワークの解析から手を付け始め、最終的に当時まだ資料の少なかったファイアーウォールの技術を調べ、何とかしてこの状況を変化させるために、計算センターに対して交渉をし、同時に、いかにファイアーウォールを無効にするか、その影響を最小限にする技術の習得、アプリケーションの開発に力を注ぐことになる。

この文章に書かれている内容は、こうした背景のなかで作成検討され、ファイアーウォールが存在する状況下において、新しいサービスをどう提供または享受するかという問題を、実装のレベルにおいて解決する。

これら技術を使用することによってはじめて我々は、広大なネットワークに対し対等に近い形になれたのである。

[†]愛知工業大学大学院 学生 (豊田市)
[‡]愛知工業大学 情報通信工学科 (豊田市)

ファイアーウォールという言葉が言われる前から、ネットワークのセキュリティを確保する技術は存在している。しかし、それらはネットワークを遮断し、多くのサービスを犠牲にするのを前提としていた。ファイアーウォールがこれらの技術と根本的に違うことは、ネットワークの利便性を確保しつつセキュリティを向上させる技術である。

我々の前に立ちふさがる壁は、果たして我々にネットワークの利便と、セキュリティの確保を与えてくれたのであろうか。

2　設計

アプリケーションゲートウェイとは、その文字が示す様に、アプリケーションレベルでの中継を行うアプリケーションである。

通常、ネットワークで何らかのサービスを受ける時、クライアントとサーバは直接接続されデータが転送される。アプリケーションゲートウェイは、直接接続できない場合に、データを中継するものであり、クライアントはアプリケーションゲートウェイに、あたかもサーバに接続されているかの様に接続する。

アプリケーションゲートウェイは、クライアントの支持に従い、必要に応じてサーバと接続しデータを取り寄せ、サーバが行なっているサービスと見かけ上同じ動作をする。このことから、アプリケーションゲートウェイは、代理サーバとも呼ばれる。

このようなことからアプリケーションゲートウェイは、各サービスによって異なった動作を要求されるため、各サービスに応じてアプリケーションゲートウェ

イを用意する必要がある。

今回設計するシステムは、PPP 接続されたマシンを主たる起動されるべきマシンとし、それらのマシンは常に利用者の管理下にあり、システムが異常を起こしたり停止した場合に、速やかに再起動や調整が行われる状況にあるものとする。これはシステムが、安定性とシステムのセキュリティを確保する方法を提供していないことに起因する。

設計されるシステムは、各サービスにおける専用のアプリケーションゲートウェイを作成するものであるが、その利用は一時的なものとし、早急により高度に洗練されたアプリケーションゲートウェイを必要とする。特に、負荷が集中するような基幹システムに導入するのは、現在のコンピュータの能力からもかなり難しいと思われる。

設計されるべきアプリケーションゲートウェイは、あくまで未知のサービスに対するアプリケーションゲートウェイであり。その設計及び構築を JavaBeans の立場から支援するものである。

SOCKS(SOCKS Protocol: rfc1928) などに代表される比較的汎用的なアプリケーションゲートウェイは、それ独自のプロトコルを必要とし、専用のクライアントを要求するが、今回は、これら専用のクライアントを必要としない、アプリケーションゲートウェイの設計を主な目的としている。無論、SOCKS と同じアプリケーションゲートウェイの設計は可能である。

従って、作成されるプログラムは、各サービスごとに作成され、サーバと同じ動作をするものとなる。

これは、一時的に未知のサービスにいち早く適応するのを目的としているためである。

3 機能分類

今回は JavaBeans で実装を試みるため、アプリケーションゲートウェイのどの部分を Bean として設計するかが、実装する段階で大きな問題となる。

そこで、アプリケーションゲートウェイに内蔵されている各機能を、次の様に分類する。

3.1 基本機能

アプリケーションゲートウェイを作成するのに、最低限必要な機能として次の機能がある。

- サーバ

  ポートに常駐し接続を待機する。

  通常クライアントとサーバの形を取る場合、必ずサーバ側はクライアントの接続を待機するために、常にサーバ側のポートに常駐している。

- 接続

  本来のサーバに接続する。

  アプリケーションゲートウェイを実現するため、本来のサーバと接続しデータの転送に備える。

- データ転送

  実際のデータ転送を行う。

  送られて来たデータを転送先にそのまま転送する。

3.2 補助機能

アプリケーションゲートウェイには直接は必要無いが、セキュリティ等を確保するために次の機能が必要となる。

- アクセス制限

  アプリケーションゲートウェイにアクセスして来たマシンが、アクセスを許可されているかどうかを調べる。

  アクセスが許可されていない信用できないマシンに対する、一つの防衛処置である。

- ログ記録

  アプリケーションゲートウェイの動作を記録する。

  不正使用やアプリケーションの不正動作を発見するのに不可欠である。

3.3 プロトコル依存機能

アプリケーションゲートウェイは各プロトコルを模倣するために、そのプロトコルに依存した機能が必要である。

- 臨時サーバ

  一時的に接続を待機するサーバ。

  通常のサーバとは違い、1 度の接続を受け付けた後消滅する。

- プロトコル解析部

  実際のプロトコルの解析を行う。

  各々のプロトコルによってさまざまな処理を行う必要があるため、この部分はより低レベルな実装が必要になる。

4 実装上の問題

以上に述べられた機能を Bean として実装するにあたり、起きる問題点について考察する。

全ての Java で作成された class は Bean になることができる。だが、多くの Bean はその動作のインターフェースとして、Event と Event を受信する Listener を持っている (図 1)。

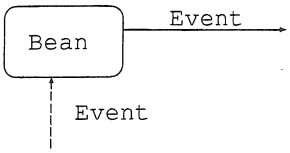

図 1: Bean のインターフェースとしての Event

4.1 Event のループ

各機能を純粋に Event で結んだ場合、Event は Event を発生させた Thread で実行されるため、図 2の様な問題が発生する。

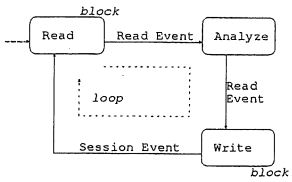

図 2: Event のループと Block

読み出し部はデータが読み込まれた後、解析部に Event を送る。解析部はそれを受けて処理を行い、結果を出力部に送る。出力部はデータを出力した後、読み出し部に Event を送るというループがおきる。これらの Thread は最初に行われた読み出し部の Thread で行われるのである。

つまり、最初に発生した Event は、全てのデータが転送され回線が切られるまで Event としての処理が終了しない。このため、複数の Event を通知することが出来ないだけでなく、読み出し部と出力部は処理が終了するまで Thread を停止させるため、その間の処理を全く行うことができない。

この問題を解決するために、Thread が停止する可能性がある場合は、自分自身の Clone を作成し、新しい Thread で動作させる方法をとる (図 3)。

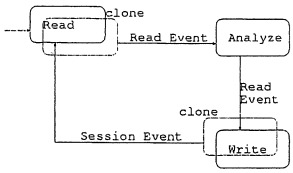

図 3: Clone による Thread の分割

4.2 同じ Event の発生

接続機能からは、中継の入出力における 2 つのデータの流れである、内側から外側、外側から内側の 2 つの Event が発生する。ところが、この 2 つの Event は同じ Event であるため、このままでは区別することができない (図 4)。

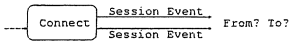

図 4: 同じ Event の発生

そこで 2 種類の Event を発生させ、間に Event を変換する AdapterBean を配置し問題を解決する (図 5)。

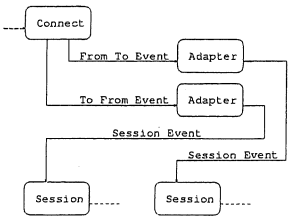

図 5: Adapter による Event の識別

5 Bean

これまでの考察により、それぞれの機能を実装した Bean を実装する。

基本的な Bean と Event について、Event 処理に関わる public なメソッドを以下に説明する。

5.1 スーパー・クラス

- メッセージのリスナ登録

  ログ出力に備えて、メッセージを出力する Bean に共通なメソッドを提供する abstract class。

  MkProxyBMessageListener を管理するメソッドが用意されている。

```
public abstract class
  MkProxyBMessageBase
extends Object
implements java.io.Serializable {
  public synchronized void
    addMkProxyBMessageListener
      (MkProxyBMessageListener l);
  public synchronized void
    removeMkProxyBMessageListener
```

```
        (MkProxyBMessageListener l);
}
```

- データ読みだし

  データ読みだし Bean に共通なメソッドを提供する abstract class。

  MkProxyBSessionCloseListener を管理するメソッドが用意されており、MkProxyBSessionListener のインターフェースを持っている。

```
public abstract class
  MkProxyBSessionReadBase
extends Object
implements MkProxyBSessionListener,
java.io.Serializable,
Runnable, Cloneable {
  public synchronized void
    addMkProxyBSessionCloseListener
      (MkProxyBSessionCloseListener l);
  public synchronized void
    reMovemkproxybsessioncloselistener
      (MkProxyBSessionCloseListener l);
  public abstract void
    mkProxyBSession
      (MkProxyBSessionEvent evt);
}
```

5.2 基本 Bean

- サーバ

  サーバ機能を提供する Bean。

  MkProxyBServerEvent を受けて、サーバ機能を提供し、MkProxyBConnectEvent を通じて接続 Bean に Event を伝える。

```
public class MkProxyBServer
extends MkProxyBMessageBase
implements MkProxyBServerListener,
  Runnable,Cloneable {
  public synchronized void
    addMkProxyBConnectListener
      (MkProxyBConnectListener l);
  public synchronized void
    removeMkProxyBConnectListener
      (MkProxyBConnectListener l);
  public void mkProxyBServer
    (MkProxyBServerEvent evt);
}
```

- 接続

  接続機能を提供する Bean。

  MkProxyBConnectEvent を受けて、接続先に接続し回線を確保した後、MkProxyBSessionEvent を通じてデータ転送 Bean に Event を伝える。

```
public class MkProxyBConnect
extends MkProxyBMessageBase
implements MkProxyBConnectListener,
    MkProxyBSessionCloseListener,
Runnable, Cloneable {
  public synchronized void
    addMkProxyBConnectFromToListener
      (MkProxyBConnectFromToListener l);
  public synchronized void
    removeMkProxyBConnectFromToListener
      (MkProxyBConnectFromToListener l);
  public synchronized void
    addMkProxyBConnectToFromListener
      (MkProxyBConnectToFromListener l);
  public synchronized void
    removeMkProxyBConnectToFromListener
      (MkProxyBConnectToFromListener l);
  public synchronized void
    mkProxyBConnect
      (MkProxyBConnectEvent evt);
  public synchronized void
    mkProxyBSessionClose
      (MkProxyBSessionCloseEvent evt);
}
```

- データ転送

  データ転送機能を提供する Bean。

  MkProxyBSessionEvent を受けて、実際のデータを転送し続ける。

  異常発生時には、MkProxyBSessionReadBase にある MkProxyBSessionCloseEvent を発生させる。

```
public class MkProxyBSessionFirst
extends MkProxyBSessionReadBase {
  public synchronized void
    mkProxyBSession
      (MkProxyBSessionEvent evt);
}
```

5.3 基本 Event

- サーバ

  アプリケーションゲートウェイのサーバソケットのポートと新たな接続先の IP アドレスとポートが格納されている。

```
public class MkProxyBServerEvent
extends java.util.EventObject {
  public MkProxyBServerEvent
    (Object source, int wport,
      InetAddress taddr, int tport);
  public int getWaitPort();
  public InetAddress getConnectAddr();
  public int getConnectPort();
}
```

- 接続

  確立されたソケットと接続先のIPアドレスとポートが格納されている。

```
public class MkProxyBConnectEvent
extends java.util.EventObject {
  public MkProxyBConnectEvent
    (Object source, Socket socket,
      InetAddress addr, int port);
  public Socket getFromSocket();
  public InetAddress getConnectAddr();
  public int getConnectPort();
}
```

- データ転送

  データ転送に必要な MkProxyBSessionInfo が格納されている。

```
public class MkProxyBSessionEvent
extends java.util.EventObject {
  public MkProxyBSessionEvent
    (Object source,
      MkProxyBSessionInfo info);
  public MkProxyBSessionInfo
    getSessionInfo();
}
```

- 接続終了

  接続終了時のステータスとメッセージが格納されている。

```
public class MkProxyBSessionCloseEvent
extends MkProxyBSessionEvent {
  public MkProxyBSessionCloseEvent
    (Object source,
      MkProxyBSessionInfo info,
      boolean stat, String mess);
  public boolean getStat();
  public String getMessage();
}
```

5.4 アダプタ Bean

- 接続方向変換

  接続 Bean によって確立されたデータの入出力で、クライアントからサーバへの流れと、サーバからクライアントへの流れを吸収する Bean。

```
public class MkProxyBConnectAdapter
extends Object
implements
MkProxyBConnectFromToListener,
MkProxyBConnectToFromListener,
  java.io.Serializable {
  public synchronized void
    addMkProxyBSessionListener
      (MkProxyBSessionListener l);
  public synchronized void
    removeMkProxyBSessionListener
      (MkProxyBSessionListener l);
  public void mkProxyBConnectFromTo
    (MkProxyBSessionEvent evt);
  public void mkProxyBConnectToFrom
    (MkProxyBSessionEvent evt);
}
```

5.5 その他の Bean と Event

その他の Bean と Event に関しては、ソースリストを参照されたい。

6 実装

これらの Bean を組み合わせることにより、アプリケーションゲートウェイを実装する。

6.1 作成手順

基本 Bean を BeanBox 等を使用し図 6の様に組み立てると、接続してきたデータをそのままサーバに転送するもっとも単純なアプリケーションゲートウェイを作成することができる。

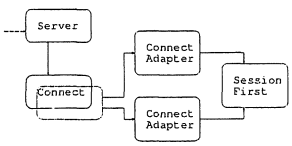

図 6: もっとも単純なアプリケーションゲートウェイ

このアプリケーションゲートウェイは、telnet などの中継を行うことができる。

- 複数の接続をもつプロトコル

  FTP(File Transfer Protocol: rfc737, 743, 775, 783, 959) などのプロトコルは、データの転送を制御する接続と、実際にデータを転送する接続の 2 つの接続を持っている。

  こうしたプロトコルの場合、制御する接続に流れるデータを逐次解析し、データを転送する接続をアプリケーションゲートウェイが用意しなければならない。

  通常こうしたプロトコルは、1 行毎に制御データが分かれているため、MkProxyBSessionFirst を、一行を読み込む Bean である MkProxyBSession-LineRead とそれを送信する Bean、MkProxyB-SessionSend に分け、その間にプロトコル解析部を新たに設置する (図 7)。なお、その他のプロトコルに対応するため、MkProxyBSessionByteRead(1 バイト読み込み)、MkProxyBSessionBufRead(バッファにたまっているデータの読み込み) も用意されている。MkProxyBSessionSend はこれら 3 つの Bean の Event-Listener になることができる。

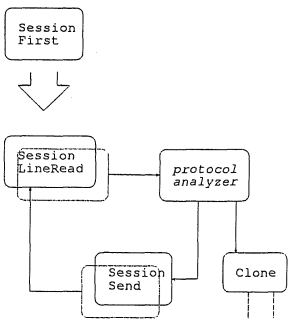

図 7: プロトコル解析部の設置

プロトコル解析部は、制御する接続に流れるデータを解析し、データ転送用の接続の要請があった場合、MkProxyBServerClone(臨時サーバ) を呼び出し接続の中継を行い、同時に流れるデータの IP アドレスとポートを、臨時サーバの IP アドレスとポートに書き換える。

このプロトコル解析部は、プロトコルに依存するため各々プロトコルに応じて作り直さなければならない。さらにその内容には高度な字句解析等が必要になり、このプロトコル解析部をさらに細かい Bean に分解し、Bean の合成によって作成することも可能であるが、処理が煩雑になるため、今回は実装する者が直接 java のコードを作成するという形式をとる。

現在既に、java コードそのものをかなり低いレベルでグラフィカルに合成するツールが開発されているのを見ると、将来 java のソースコードを実装者が直接打つことなく設計出来るようになるであろう。しかし、現状の JavaBeans でソースに近いコードを作成するのは、かなりの困難をともなう。

- ヘッダ部分に動作が記述されている

  多くのアプリケーションゲートウェイは、前述した 2 つのアプローチによって作成することができる。

  ここに記述されているアプリケーションゲートウェイは、ヘッダ部に接続先が示されているような、http の代理サーバや SOCKS の動きを実現する。

  このアプリケーションゲートウェイのプロトコル解析部は、接続機能を継承する。(図 8)。

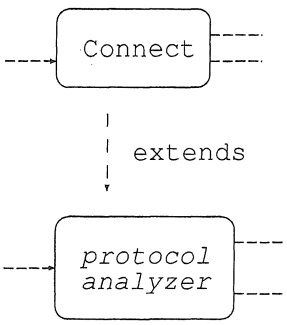

図 8: ヘッダ解析とその接続

プロトコル解析部は、アプリケーションゲートウェイに接続されたクライアントから、制御データを取り出しそれに従い接続先に接続する。

6.2 補助機能

アプリケーションゲートウェイの機能には直接関係はないが、次の機能を組み込んでおく必要が多くの場合存在する。

- アクセス制限

  MkProxyBConnect を継承し接続制限を持つ Bean を作成する (図 9)。

  この Bean はあらかじめ信用すべきマシンを知っており、それ以外からのアクセスがあった場合そこで Event をブロックし、次の Bean の発動を阻止する。

- ログ出力

  多くのデータを制御する Bean はログ出力に備えて状況の変化に応じて MkProxyBMessageEvent を発火させるべきである。

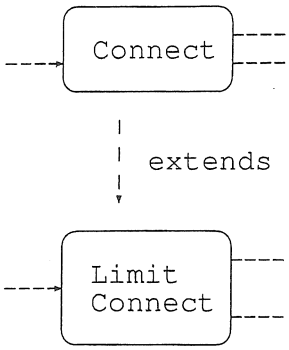

図 9: アクセス制限

実装時にはこれら Bean からの MkProxyBMessageEvent を捕捉し、ログ記録 Bean に伝えるなどしてログの記録を行う (図 10)。

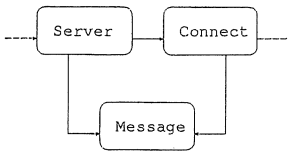

図 10: ログ出力

7 実際の設計と実装

実際に IRC(Internet Relay Chat Protocol: rfc1459) のアプリケーションゲートウェイの設計を試みる。

rfc1459 に記述されている IRC のプロトコルは、1 対 1 の純粋な通信であり、すでに作成されている単純なアプリケーションゲートウェイで、実現することが可能である。

だが、現実には DCC(Direct Client Connection) と呼ばれる、クライアント同士の接続が行われており、この問題を既にあるアプリケーションゲートウェイは解決してくれない。

7.1 作成しなければならない Bean

IRC の基本的なデータの流れは CR と LF により区切られた行単位で行われるため、MkProxyBLineRead を用いデータを行に分解した後、プロトコル解析部が解析を行う。

プロトコル解析部には、壁の内側と外側の IP アドレスを知る必要があるため、それを設定するコントロール部、分解するアダプタ部が必要である。

7.2 作成されたアプリケーションゲートウェイ

実際に作成されたアプリケーションゲートウェイは図 11となる。

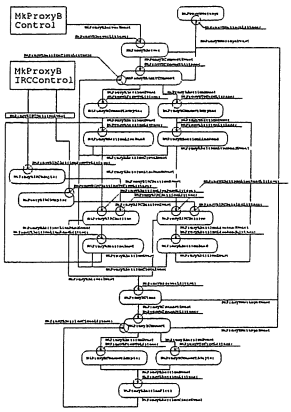

図 11: 作成された IRC アプリケーションゲートウェイ

8 応用

こうして設計されたアプリケーションゲートウェイは、全てモジュール化されており、また、プロトコル解析部の変更なしにデータ転送部の変更が非常に容易である。

さらに、特別なクライアントを必要とせず、サーバの接続を模倣することから、多くの応用が考えられる。

8.1 暗号通信

データ転送部分を暗号通信に対応させる、すなわちデータストリームに暗号化の Bean を連結させれば、そのデータ転送の全ては暗号化して転送される。

利用する多くのプロトコルについて、この暗号化を施したアプリケーションゲートウェイを作成すれば、利便性を失うこと無く比較的安全なデータの転送が可能となる。

8.2 TCP/IP 以外の通信

暗号化通信と同様に、データストリームにはデータの転送が可能であればどの様な方式でも良く、TCP/IP ではない方法でデータを転送することも可能である。

RS-232C などのシリアル転送や、CGI(Common Gateway Interface) を用いたブロック転送などにも応用することができ、TCP/IP で接続されていなくても、ネットワークの恩恵を得ることがある程度可能である。

9 おわりに

本稿では、Java 実行可能な条件下での、いかなるプロトコルにも容易に対応可能なアプリケーションゲートウェイの実装支援について述べた。Java の現在の発展状況と独特のマルチプラットホームの思想から、このアプリケーションは多くの環境で実行可能になるであろう。さらに、このアプリケーションゲートウェイは、TCP/IP 同士の中継に留まらず、データを比較的即時に転送出来る環境下であれば、多くのプロトコルを転送することを可能にする。これは、今までネットワークの恩恵を得られなかった人々に、新しい選択肢を与えるものである。

ところが、現状の Java の実行速度は早いとは言えず、実用的に利用しようとするとかなり強力なコンピュータが必要となる。また、ネットワークに直接接続されているため、セキュリティの問題は常に考慮しなければならない。アプリケーションの作成そのものは簡単であるが、それを安全に作成するのは非常に困難であり、たとえ安全に作成されたとしても、運用する時に細心の注意が必要となる。

Java の実行速度については、いろいろな技術が考案されており、コンピュータの実行速度の向上と共に問題は解決していくであろう。セキュリティに関しては、いまだに論議されているものも多く、アプリケーションゲートウェイがネットワークの接続を転送するものである以上、使用すれば少なからず問題が発生するのはある程度避けられない。

セキュリティを完全に守るためには、ネットワークとの接続を絶つのがもっとも確実である。しかし、それは同時にネットワークの利便性をも失うことを意味する。ネットワークの今後の発展を考えると、もはやネットワークと接続を行わないのはナンセンスと言わざるを得ない。我々は既に移動端末から容易にネットワークにアクセス出来るのである。もし、現状のネットワークが利便性を提供していないのであれば、ネットワークセキュリティの知識のないユーザが、移動端末を通じて外部のネットワークに無防備な内部のネットワークを接続するかもしれない。

こうした意味からも、全てのネットワークはある程度セキュリティを考慮すべきである。そして、全てのユーザに対し行動を明確に制限出来ないのであれば、そのネットワークはある程度の利便性をセキュリティを守るために提供すべきである。

本稿はかなり強力な利便性を提供しないネットワーク下で作成された。このアプリケーションゲートウェイは、多くのユーザに利便性を提供すると同時に、多くの強固なファイアーウォールを駆逐する能力を秘めている。

参考文献

1. Kevin Washburn, Jim Evans (油井 尊 訳). TCP/IP バイブル (*TCP/IP:Running a Successful Network*). ソフトバンク株式会社 出版事業部. 1994.

2. William R. Cheswick, Steven M. Bellovin (川副 博 監訳). ファイアーウォール (*Firewalls and Internet Security*). ソフトバンク株式会社 出版事業部. 1995.

3. Robert B. Murray (岩谷 宏 訳). 現実的な C++プログラミング (*C++ Strategies and Tactics*). ソフトバンク株式会社 出版事業部. 1994.

4. 大谷卓史, 武藤健志. はじめての Java. 株式会社技術評論社. 1996.

5. 坂本 衛. JavaBeans プログラミング入門. 株式会社オーム社. 1997.

6. Robert Englander (鷲見 豊 訳). JAVA Beans 基礎から開発まで (*Developing JAVA Beans*). 株式会社オライリー・ジャパン. 1997.

（受理　平成10年 3 月20日）