Latest Progress and Trends in Java/EJB Technologies

● Sumio Tasaka ● Jun Ginbayashi

(Manuscript received February 2, 2004)

Java is evolving into a full-scale system development technology by the addition of functions such as Applets, object-oriented programming, cross-platform support, internationalization, JavaBeans, Remote Method Invocation (RMI), Java Database Connectivity (JDBC), Enterprise JavaBeans (EJB), Java Plug-in, HotSpot Client Virtual Machine (VM), Java Naming and Directory Interface (JNDI), RMI/Internet Inter-ORB Protocol (IIOP), eXtensible Mark-up Language (XML) support, and Java Secure Socket Extension (JSSE). These functions have been added in three specifications: Java 2 Platform, Enterprise Edition (J2EE); Java 2 Platform, Standard Edition (J2SE); and Java 2 Platform, Micro Edition (J2ME). Also, the manner in which these specifications are decided has shifted from decisions made by Sun Microsystems to decisions made by an open community through a process called the Java Community Process (JCP). The main key-phrase of the latest trend in Java/EJB is "Ease of Development," and J2SE 1.5 (code name "Tiger") will play a central role in this trend. Prospective enhancements in the Java programming language specifications such as Generics, Enhanced for Loop, Autoboxing/Unboxing, Typesafe Enums, Varargs, Static Import, and Metadata are expected to simplify system development, learning, and mastering. With these new technologies, Java will be used by a wider range of developers and EJB-based development of enterprise systems will become more popular.

1. Introduction

Since its appearance in 1995, Java technology (including Enterprise JavaBeans [EJB]) has been evolving with various functions towards a technology for full-scale system development. Frequent enhancements with these functions have led to a wider Java application area. However, these enhancements have raised application developers' feelings of insecurity about the stability of Java and forced them to spend more time learning and mastering Java.

This paper aims to eliminate unnecessary anxiety among developers by looking back on the history of Java/EJB technology and clearly showing its future trend.

First, in Section 2, the history so far of Java/EJB technology is summarized. Section 3 shows the direction of future enhancements in Java/EJB, mainly based on the themes that are currently being discussed in the Java Community Process (JCP). In Section 4, the details of the latest version (Java 2 Platform, Standard Edition [J2SE] 1.5), whose outline has just become clear, are introduced. Finally, Section 5 concludes with a brief look at the future of Java/EJB.

2. History and current status of Java

Java was introduced by Sun Microsystems in 1995 together with a browser, HotJava. Then, Java gained public attention in the form of Applets, which are programs running on a browser, and was later expanded to include, for example, server-side applications and mobile appliances.

The major version upgrades of Java are shown in **Table 1**.

The major enhancement points of each version are given below.

1) Version 1.0

Java Development Kit (JDK) 1.0 was announced as the first release of Java with the following features:

- Object orientation
- Cross-platform support
- Network compatibility

Because it was a C++-like language, and also because of its simple and pure language specifications, Java was quickly and warmly accepted by developers, which is unusual for a new language. However, it did not have sufficient functions for system development. Also, it was difficult to use Java in a Japanese environment, because internationalization was not supported.

2) Version 1.1

Approximately one year later, JDK 1.1 was announced. The following functions were added in JDK 1.1:

- Internationalization
- JavaBeans
- Remote Method Invocation (RMI)
- A new event model
- Java Database Connectivity (JDBC)

Java did not have enough functions to support system development until functions such as JavaBeans and JDBC were provided in this version.

3) Version 1.2

From JDK 1.2, Java was given the new brand name "Java 2." In JDK 1.2, the following functions were added:

- Java Foundation Classes (JFC)
- Input Method Framework (IMF)
- Common Object Request Broker Architecture (CORBA) support
- EJB
- Java Plug-ins

After the release of Java 2, EJB made its appearance and more enterprise systems were developed using the server-side version of Java. It would appear that, at first, Java was expected to be effective for solving the shortage of client-side applications on platforms other than Windows (as implied by the catch phrase, "write once, run anywhere"). However, server-side Java became popular in those days because of the expansion of the Internet.

4) Version 1.3

From JDK 1.3, Java and its related functions were reorganized into the following three editions based on the target system environment: Java 2 Platform, Enterprise Edition (J2EE);[1] Java 2 Platform, Standard Edition (J2SE);[2] and Java 2 Platform, Micro Edition (J2ME).

J2SE 1.3 had the following additional functions:

- HotSpot Client Virtual Machine (VM)
- Java Naming and Directory Interface (JNDI)
- RMI/Internet Inter-ORB Protocol (IIOP)
- Security enhancements

After the release of version 1.3, the way in which Java will be applied to enterprise systems became clear with the editions based on target

Table 1
Major version upgrades of Java.

Major Java upgrades Release date	Version	Name
May 1995	Java (announced)	
January 1996	1.0	JDK 1.0
February 1997	1.1	JDK 1.1
December 1998	1.2	Java 2 (JDK 1.2)
July 2000	1.3	Java 2 (J2SE 1.3, J2EE 1.2, J2ME)
February 2002	1.4	Java 2 (J2SE 1.4, J2EE 1.3, J2ME)

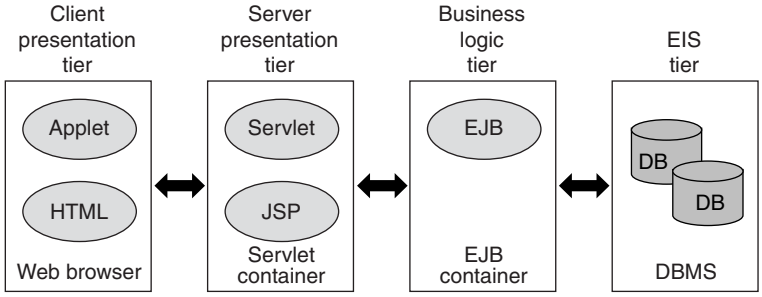

Figure 1
J2EE application model and components.

platforms. In particular, Java application to enterprise systems became more popular after EJB 2.0 and JavaServer Pages (JSP) were announced in J2EE. **Figure 1** shows the application model and components supported by J2EE. To make it easier to build enterprise systems through component-based development, several detailed application models based on EJB have been advocated by companies and consortiums such as the Component Consortium for EJB,[3] which has been established in Japan. **Figure 2** shows the EJB-based application model of the Component Consortium for EJB. At the same time, because of J2ME, the application area of Java expanded to include mobile appliances, which led to the current widespread use of Java in mobile-phone applications.

Among the enhancements in version 1.3, the following functions added to EJB in EJB 2.0 are important because they can be used to design and implement coarse-grained entities and make database access more efficient, which are both necessary tasks when developing enterprise systems:

- A new architecture for container-managed persistence (CMP 2.0)
- Support for the management of relationships among entity beans
- Query syntax for select methods for entity beans (EJB-QL)
- Message-driven beans

5) Version 1.4

Until version 1.3, Sun Microsystems decided the specifications of Java and its related

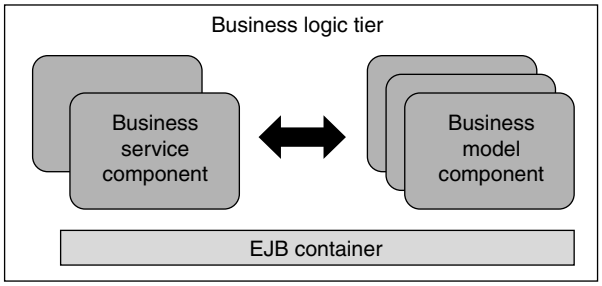

Figure 2
EJB-based application model of Component Consortium for EJB.

functions. Since version 1.4, however, the specifications have been decided in a more open manner. Now, new specifications and improvements of existing specifications are made based on Java Specification Requests (JSRs) submitted by the Java community to a process called the Java Community Process (JCP).[4]

When a JSR is submitted to the JCP, the member companies of the JCP discuss whether it should be implemented. If they decide it should be implemented, an expert group is established to decide the final specifications for the JSR. Only member companies of the JCP can submit a JSR, but submitted JSRs are made public on the JCP's Web site so the JCP can receive a wider range of opinion.

J2SE 1.4 was the first version whose specifications were discussed and determined by the JCP. The following functions were added:

- XML support
- Java Web Start
- A new I/O

- A logging API
- Assertion
- Java Secure Socket Extension (JSSE)

In this way, the specifications of Java are now discussed and evolved within an open community.

3. Future enhancements of Java

A roadmap for future enhancements of Java has been released.[5] The roadmap is shown in **Table 2**.

As described in Section 2, the application area of Java has become wider and Java technology is becoming more popular. J2ME has been adopted in mobile appliances such as mobile phones. J2EE has already reached the level of de facto standard in the development of enterprise mission-critical systems along with the rich enhancement of EJB functions.

However, Java is still difficult for beginners to learn and there are relatively few high-level developers. For this reason, Java is not so successful in the area of section-wise application, where Basic and C# are used. On the other hand, Java is successful in the area of mobile applianc-es and enterprise systems.

As a result, for the next version of Java, ease of development rather than function enhancement will be the priority issue. This could be the largest change of policy since Java was born.

The next version of Java will be J2SE 1.5 (code name "Tiger"). For J2SE 1.5, the following five themes will be considered:

1) Quality: compatibility will be extremely important.
2) Performance and scalability
3) Ease of development
4) Monitoring and manageability
5) Desktop clients

"JSR-176: J2SE 1.5 (Tiger) Release Contents" is being developed in the JCP according to these themes. Furthermore, several JSRs have been submitted with respect to the specifications of more detailed functions (**Table 3**).

The final version of J2EE 1.4 was released in November 2003. EJB 2.1, in particular among

Table 2
J2SE Technology Update and Roadmap.

Date	Version (Code name)
Summer 2004	1.5 (Tiger)
Early 2005	1.5.1 (Dragonfly)
After 2005	1.6 (Mustang)

Table 3
JSR-176: J2SE 1.5 (Tiger) Release Contents.

JSR-003	Java Management Extensions (JMX) Specification
JSR-013	Decimal Arithmetic Enhancement
JSR-014	Add Generic Types to the Java Programming Language
JSR-028	Java SASL Specification
JSR-114	JDBC Rowset Implementations
JSR-133	Java Memory Model and Thread Specification Revision
JSR-163	Java Platform Profiling Architecture
JSR-166	Concurrency Utilities
JSR-174	Monitoring and Management Specification for the Java Virtual Machine
JSR-175	A Metadata Facility for the Java Programming Language
JSR-199	Java Compiler API
JSR-200	Network Transfer Format for Java Archives
JSR-201	Extending the Java Programming Language with Enumerations, Autoboxing, Enhanced for Loops and Static Import
JSR-202	Java Class File Specification Update
JSR-204	Unicode Supplementary Character Support
JSR-206	Java API for XML Processing (JAXP) 1.3

several constituent specifications in J2EE 1.4, has the following additional functions:

1) Implementation of Web services
2) A container-managed timer service
3) An enhancement of EJB QL (ORDER BY, aggregate operators)
4) An extension for message-driven beans

Ease of development is also the major remaining problem in EJB, and the use of metadata, for example, will be pursued in the standardization activity for EJB3.0 (JSR-220). The following features are now under discussion in JSR-220:

1) A simpler CMP programming model
2) Development and testing without an EJB container
3) More standardized deployment

4. Extension of Java programming language specifications in J2SE 1.5

The outline of the latest version, J2SE 1.5, has just become clear. This chapter explains which types of functions, especially which types of extensions to the Java programming language specifications, will be added in J2SE 1.5.[6),7)]

The following extensions in J2SE 1.5 have been announced:

1) Generics
2) Enhanced for Loop (foreach)
3) Autoboxing/Unboxing
4) Typesafe Enums
5) Varargs
6) Static Import
7) Metadata

These functions are explained in detail below.

4.1 Generics

For the current collection-type classes such as java.util.list, and java.util.Vector, extra casts are needed because all the objects in the collection are treated as the base class Object. Also, even when an object of a different type is added to the collection, an error only occurs at runtime.

Generics make it possible to check the type of elements in collection-type classes at compile time by specifying the type of objects stored in collection-type classes. This function is similar to Template in C++. This function will make it possible to write more type-safe descriptions in Java.

For example, when String objects are stored into and retrieved from ArrayList, it is currently necessary to cast them as follows:

```
List wordlist = new ArrayList();
wordlist.add("title");
String title =
   ((String) wordlist.get(0)).toUppercase();
```

However, when Generics is used, the cast is not needed:

```
List<String> wordlist =
   new ArrayList<String>();
wordlist.add("title");
String title =
   wordlist.get(0).toUppercase();
```

Also, because any type of object can be stored in a List class, even when only String objects should be stored in a wordlist, Integer objects can be added to the wordlist as follows without causing errors at compile time:

```
List wordlist = new ArrayList();
wordlist.add(new Integer(1));
Integer i = (Integer) wordlist.get(1);
```

If we describe as follows using Generics:

```
List<String> wordlist =
   new ArrayList<String>();
wordlist.add(new Integer(1));
Integer i = wordlist.get(1);
```

an error occurs at compile time because the type is specified.

This function makes the code simple without annoying casts and increases the level of safety

by type checking at compile time. Also, the number of errors at runtime is reduced.

Generics has been considered for quite some time and might have been included in J2SE 1.4. It will finally be added in J2SE 1.5.

4.2 Enhanced for Loop (foreach)

Enhanced for Loop provides the same function as the foreach syntax in C# and other languages.

For example, to look at the elements of an array or collection one by one, we currently use Iterator as follows:

```
void cancelAll(Collection c) {
   for (Iterator i = c.iterator();
      i.hasNext();){
      TimerTask tt = (TimerTask) i.next();
         tt.cancel();
   }
}
```

By describing this with Enhanced for Loop, the code becomes simpler:

```
void cancelAll(Collection c){
   for (Object o : c)
      ((TimerTask)o).cancel();
}
```

Also, it is possible to describe this example with Generics:

```
void cancelAll(Collection<TimerTask> c){
   for (TimerTask task : c)
      task.cancel();
}
```

Enhanced for Loop makes it unnecessary to use loop variables and Iterator in code. It is expected to decrease the number of mistakes in coding, because it reduces the volume of description, especially when describing a nested loop.

4.3 Autoboxing/Unboxing

In the current Java, when a variable of a primitive type, for example, the int type, is treated as an object, it must be converted to the corresponding wrapper class, for example, Integer. In J2SE 1.5, conversion from a primitive type to a wrapper class and vice versa is automatically performed by Autoboxing/Unboxing. The function of Autoboxing/Unboxing is also adopted in C#.

For example, the following description in the current Java:

```
// from int to Integer
int i = 42;
Integer x = new Interger(i);

// from Integer to int
Integer y = 42;
int j = y.intValue();
```

can be simply described as follows by using Autoboxing/Unboxing:

```
// from int to Integer
int i = 42;
Integer x = i;

// from Integer to int
Integer y = 42;
int j = y;
```

Autoboxing/Unboxing makes it unnecessary to use a wrapper class. Also, when combined with Generics, this function is very useful when, for example, an Integer needs to be stored in a collection.

4.4 Typesafe Enums

Typesafe Enums is a famous function in C, C++, Pascal, and other languages, and will be added in J2SE 1.5. It can be used as a label in switch statements. Typesafe Enums can be defined like a class rather than a list of integers. Also, because Typesafe Enums automatically generates VALUES, family(), valueOf(), and other methods, it is

especially useful in the Enhanced for Loop syntax.

The following shows an example of using Typesafe Enums:

```
public enum Coin{
   penny(1), nickel(5), dime(10), quarter(25);
   Coin(int value) { this.value = value; }
   private final int value;
   public int value() { return value; }
}

public class CoinTest{
   public static void main(String[] args){
      for (Coin c : Coin.VALUES)
         System.out.println
            (c + ": ¥t"+ c.value() +"¢ ¥t" + color(c));
   }

   private enum CoinColor
   { copper, nickel, silver }

   private static CoinColor color(Coin c) {
      switch(c) {
         case penny: return CoinColor.copper;
         case nickel: return CoinColor.nickel;
         case dime: return CoinColor.silver;
         case quarter: return CoinColor.silver;
         default: throw new
            AssertionError("Unknown coin: " + c);
      }
   }
}
```

In this example, class CoinTest uses an enum definition Coin. The new Enhanced for Loop syntax in the main method prints out the contents of the enum definition Coin. For example, "penny: 1¢ copper" is printed out for the first content, penny. The example shows that VALUES, which is automatically generated, is used in the Enhanced for Loop syntax.

Typesafe Enums does not exist in the current Java. It simplifies descriptions and therefore reduces the burden on developers.

4.5 Varargs

Varargs is a function that makes it possible to define a method having variable-length arguments like printf and scanf in C/C++.

Currently, a function similar to variable-length arguments can be described using an array, for example, as follows:

```
Object[] args = {
   new Integer(9999),
   "tom"
};

String result =
   MessageFormat
   .format("ID {0}: Name:{1}, args);
```

By using Varargs, this example can be described more simply without an array as follows:

```
String result =
   MessageFormat
   .format("ID {0}: Name:{1}, 9999, "tom");
```

4.6 Static Import

Static Import is a function that makes it possible to describe static fields and methods like the other classes by using import declarations. Because Java does not have the macro functions found in C/C++, it is usual to define an interface in which only constants are defined. However, this is an improper usage of an interface.

By using Static Import, constants in another class can be imported by describing "import static" as follows:

```
import static java.awt.Color.CYAN;

button.setForeground(CYAN);
```

In Static Import, not only constants but also static methods can be imported.

4.7 Metadata

Metadata is exactly the same as Attribute in C#. This function makes it possible to define a new attribute for classes, methods, and fields. For example, in the same way a developer can describe the accessibility to a class or method by using a declaration (e.g., public or private), a developer can explicitly specify the accessibility to a class, method, or field by defining a Metadata attribute.

Metadata has two major purposes. One is to enable developers to define an attribute clearly by describing a Metadata attribute within code. The other is to utilize such information by using tools.

For example, when Metadata is used to describe attribute @Remote in order to specify that a method is remote as follows:

```
import javax.xml.rpc.*;

public class CoffeeOrder{
   @Remote public Coffee [] getPriceList(){
      ...
   }
   @Remote public String
      orderCoffee(String name, int quantity){
      ...
   }
}
```

a tool can generate the following source code, which increases development productivity:

```
public interface CoffeeOrderIF
   extends java.rmi.Remote {
   public Coffee [] getPriceList()
      throws java.rmi.RemoteException;
   public String
      orderCoffee(String name, int quantity)
      throws java.rmi.RemoteException;
}

public class CoffeeOrderImpl
   implements CoffeeOrderIF{
   public Coffee [] getPriceList(){
      ...
   }
   public String
      orderCoffee(String name, int quantity){
      ...
   }
}
```

As the example shows, Metadata is not only an important function from a tool vendors' point of view, but also a big help for developers.

5. Conclusion

Since its appearance in 1995, Java has been evolving into a full-scale system development technology through the addition of various functions in new specifications (e.g., internationalization, JavaBeans, JDBC, EJB). At the same time, the application area of Java has expanded to include, for example, mobile appliances and enterprise mission-critical systems. Also, the manner in which the specifications are decided has shifted from decisions made by Sun Microsystems to decisions made by an open community through a process called the Java Community Process (JCP).

Currently, a significant change of policy is occurring in Java and the key phrase is "Ease of Development." J2SE 1.5, which will appear in the summer of 2004 (code name "Tiger") is at the center of this change. The extension of the Java programming language specifications now proposed in J2SE 1.5 has been carefully designed to realize a high degree of Ease of Development and is also designed to improve Ease of Learning and Ease of Mastering. With these new technologies, Java will be used by a wider range of developers and EJB-based development of enterprise systems will become more popular.

References

1) Java 2 Platform, Enterprise Edition (J2EE) Web page.
http://java.sun.com/j2ee/
2) Java 2 Platform, Standard Edition (J2SE) Web

page. (in Japanese).
http://java.sun.com/j2se/
3) Component Consortium for EJB Web page. (in Japanese).
http://www.ejbcons.gr.jp/indexi.html
4) Java Community Process Web page.
http://jcp.org/en/home/index
5) Java 2 Platform, Standard Edition (J2SE) Update and Roadmap.
http://servlet.java.sun.com/javaone/sf2003/conf/sessions/display-1540.en.jsp
6) New Language Features for Ease of Development in the Java 2 Platform, Standard Edition 1.5:A Conversation with Joshua Bloch.
http://java.sun.com/features/2003/05/bloch_qa.html
7) Forthcoming Java Programming Language Features.
http://servlet.java.sun.com/javaone/sf2003/conf/sessions/display-3072.en.jsp

Sumio Tasaka received the B.S. degree in Physics from Tokyo Metropolitan University, Tokyo, Japan in 1989. He joined Fujitsu Ltd., Yokohama, Japan in 1989, where he has been engaged in research and development of development environments.

Jun Ginbayashi received the B.S. and M.S. degrees in Mathematics from The University of Tokyo, Tokyo, Japan in 1981 and 1984, respectively. He also received the D.Phil. degree in Computing from The University of Oxford, Oxford, United Kingdom in 1996 after three years' study there. He joined Fujitsu Ltd., Tokyo, Japan in 1986, where he has been engaged in research, development, and promotion of development environments for business application systems.