



Padova 7 Maggio 2004

Tiger

nuove caratteristiche per
JAVA 1.5

Java™
Italian
Portal

by Matteo Baccan
<http://www.baccan.it>



www.JavaPortal.it



Tiger: nuove caratteristiche per Java 1.5

- Relatore: Matteo Baccan matteo@baccan.it
- Data: Milano 03/06/2004 13:00
- Area: Programming & development
- Target: Tecnici
- Difficoltà: Media

50 minuti per svelare la tigre sotto i suoi aspetti.

La nuova versione di java implementa caratteristiche che da tempo erano attese nello standard del linguaggio: Metadata, Generic Types, AutoBoxing e Unboxing dei tipi primitivi, Static import, gestione dinamica dei Loop e delle Enumeration. Nuovi strumenti per l'Analisi Diagnostica e il Monitoring e Manageability. Vediamo di cosa si tratta e di come poter utilizzare queste nuove feature all'interno dei nostri programmi

Intervento allineato alla **BETA 2**



Obiettivi

Dopo aver completato questo seminario sarai in grado di:

- Descrivere le nuove caratteristiche di Java 1.5
- Sfruttare i nuovi costrutti che sono stati introdotti con Tiger
- Creare codice più compatto ed efficiente
- Migliorare lo stile della tua programmazione



Storia

- 1995 Java 1.0
Prima release
- 1997 Java 1.1
Gestione degli eventi, ereditarietà, miglioramento della JVM
- 1999 Java 1.2
Swing, list, set, hashmap
- 2000 Java 1.3
Performance
- 2001 Java 1.4
Nuovo IO, XML, ASSERT
- 2004 Java 1.5
La svolta!!



Tiger

- Metadata (Annotations)
- Generics
- AutoBoxing e Auto-Unboxing
- Static import
- Foreach
- Enumeration
- Varargs
- Input/Output formattati
- Network
- Diagnostica e monitoring
- Pack200

Riferimenti



- Java Community Process (<http://www.jcp.org>)
 - JSR-014 Generics
 - JSR-175 Metadata
 - JSR-201 Foreach, Boxing/Unboxing, enumeration etc

Q: Cos'è JCP?

A: Fin dalla sua introduzione, nel 1998, è un processo nato per sviluppare e revisionare le specifiche tecniche di Java e le sue implementazioni. JCP contribuisce all'evoluzione della piattaforma Java cooperando con la comunità internazionale di sviluppatori.



JAVAC

Nuove opzioni di compilazione di Javac:

- source 1.5: Abilita l'uso delle nuove caratteristiche di Java 1.5 all'interno dei sorgenti (-target 1.5 è implicito)
- target 1.5: Abilita javac a usare le nuove feature di Java 1.5 all'interno di librerie e virtual machine
- Xlint: Abilita javac a produrre dei warning su costrutti legali ma logicamente sospetti come, ad esempio, la dichiarazione di una classe Serializable senza la definizione di un serialVersionUID.

```
c:\tools\biew.exe
File : impo.class           Size : 440 bytes           100%
00000000: CA FE BA BE 00 00 00 30 06 21 0A 00 06 00 0F 09  0 ! 0 0 * 0
00000010: 00 10 00 11 0A 00 12 00 13 0A 00 14 00 15 07 00  0 0 0 0 0 0 0 0
00000020: 16 07 00 17 01 00 06 3C 69 6E 69 74 3E 01 00 03  0 0 0 0 0 0 0 0
00000030: 28 29 56 01 00 04 43 6F 64 65 01 00 0F 4C 69 6E  (>U@ 0 Code@ *Lin
00000040: 65 4E 75 6D 62 65 72 54 61 62 6C 65 01 00 04 6D  eNumberTable@ 0 m
00000050: 61 69 6E 01 00 16 28 5B 4C 6A 61 76 61 2F 6C 61  ain@ 0 (Ljjava/la
00000060: 6E 67 2F 53 74 72 69 6E 67 3B 29 56 01 00 0A 53  ng/String;)U@ 0 S
00000070: 6F 75 72 63 65 46 69 6C 65 01 00 09 69 6D 70 6F  ourceFile@ 0 impo
00000080: 2E 6A 61 76 61 0C 00 07 00 08 07 00 18 0C 00 19  .java? 0 0 0 0 0 0 0 0
```



Metadata/Annotations

Un sistema per poter arricchire il codice con nuove informazioni

Vantaggi

- Permette a tool che agiscono su codice Java di non ricorrere a file di proprietà esterni
- Permette di semplificare la manipolazione del codice
- Arricchisce i programmi di nuove informazioni
- Non influisce sulle performance

Sintassi

```
@annotation( [param=valore] )
```

Ed ora un esempio ...



Metadata/Annotations

Enum RetentionPolicy:

Indica la validità dei metadata:

CLASS:

Le annotazioni sono registrate nel file .class dal compilatore ma non sono mantenute dalla VM a runtime

RUNTIME:

Le annotazioni sono registrate nel file .class dal compilatore e mantenute dalla VM a runtime, così possono essere lette tramite reflection

SOURCE:

Le annotazioni sono scartate dal compilatore



Generics

- Permette la scrittura di classi generiche (C++ Template)
- Elimina il problema del runtime-casting
- Sposta gli errori nella fase di compilazione

Javac non è il primo compilatore java a supportare i Generics. Il primo è stato: Pizza. <http://pizzacompiler.sourceforge.net/>



Generics – 1.4 senza

```
public static void main(String[] args) {  
    List myIntList = new LinkedList(); // Object container  
    myIntList.add(new Integer(2));  
    myIntList.add("String");  
    Integer iValue1 = (Integer)myIntList.get(0);  
    Integer iValue2 = (Integer)myIntList.get(1);  
    System.out.println( "*" + iValue1.intValue() + "*" );  
    System.out.println( "*" + iValue2.intValue() + "*" );  
}
```

Exception in thread "main" java.lang.ClassCastException:
java.lang.String
at gen14.main(gen14.java:10)



Generics – 1.5 con

```
public static void main(String[] args) {  
    List<Integer> myIntList = new LinkedList<Integer>(); //container  
    myIntList.add(new Integer(2));  
    myIntList.add("String");  
    Integer iValue1 = myIntList.get(0);  
    Integer iValue2 = myIntList.get(1);  
    System.out.println( "*" + iValue1.intValue() + "*" );  
    System.out.println( "*" + iValue2.intValue() + "*" );  
}
```

```
gen1.java:8: cannot find symbol  
symbol   : method add(java.lang.String)  
location: interface java.util.List<java.lang.Integer>  
    myIntList.add("String");
```



Generics – una classe

```
public class gen2<GEN> {  
    GEN uno;  
    GEN due;  
    public static void main(String[] args) {  
        gen2<Integer> myInt2 = new gen2<Integer>();  
        myInt2.setUno( new Integer(10) );  
        myInt2.setDue( new Integer(7) );  
        System.out.println( myInt2.getUno()+myInt2.getDue() );  
    }  
    public void setUno( GEN arg ) {    uno = arg; }  
    public void setDue( GEN arg ) {    due = arg; }  
    public GEN getUno() {    return uno; }  
    public GEN getDue() {    return due; }  
}
```



Auto boxing/unboxing

Capacità di convertire tipi primitivi in oggetti e viceversa

Presente anche in altri linguaggi, ad esempio: C#

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vclrfBoxingUnboxingPG.asp>



Auto boxing/unboxing – prima

```
private static final Integer ONE = new Integer(1);
public static void main(String[] args) {
    // Frequenza di stringhe
    Map m = new TreeMap();
    for (int i=0; i<args.length; i++) {
        Integer freq = (Integer) m.get(args[i]);
        m.put(args[i], freq==null ? ONE :
            new Integer(freq.intValue() + 1));
    }
    System.out.println(m);
}
```



Auto boxing/unboxing – dopo

```
public static void main(String[] args) {  
    // Frequenza di stringhe  
    Map<String,Integer> m = new TreeMap<String,Integer>();  
    for (String word : args) {  
        Integer freq = m.get(word);  
        m.put(word, freq==null ? 1 : freq+1 );  
    }  
    System.out.println(m);  
}
```



Static import

- Permette di importare dei metodi static, risolti a livello di compilazione
- Si possono importare singoli metodi, l'intera classe o l'intero package
- Semplice da usare

NB: All'uso dell'import static cambia la versione del .class: da 0x30 (1.4 compatibile) a 0x31 (1.4 non compatibile)



Static import

```
import static java.lang.Math.max; // Singolo metodo  
import static java.lang.Math.*; // Tutti gli static
```

```
public class impo2 {  
    public static void main(String[] args) {  
        int uno = 10;  
        int due = 20;  
        System.out.println( max( uno, due ) );  
    }  
}
```



Foreach

Nuovo modo di scrivere i cicli, il for viene potenziato

- Chiarezza: Molto più chiaro da scrivere e leggere, meno codice
- Errori: Minor uso di variabili temporanee, non c'è il rischio di fare cicli annidati incrociati
- Compilatore: Pensa a tutto lui

EnhancedForStatement:

```
for ( FormalParameter : Expression )  
    Statement
```



Foreach - l'evoluzione

// Old Style

```
Vector c = new Vector();  
for( Enumeration e = c.elements(); e.hasMoreElements(); )  
    String x = (String) e.nextElement();
```

// New Style

```
Vector<String> c = new Vector<String>();  
for( String s : c )  
    String x = s;
```



Enumerations

Prima

```
public static final int matteo = 0;  
public static final int carlo = 1;  
public static final int raffaele = 2;
```

Problemi

- Non typesafe. Sono solo numeri non li posso "raggruppare" in un tipo
- Le costanti sono "compilate" nel codice
- Se cambio l'ordine o tolgo una costante ho un errore runtime



Enumerations

Soluzione

Introduzione delle enum

```
public enum nomi { matteo, carlo, raffaele };
```

Gestione del tipo

Non compilazione nel codice

Gestione degli errori

Vantaggi

- Sono classi e sono quindi estendibili e possono implementare un'interfaccia



Enumerations

```
public enum nomieta { matteo(33), carlo(29), raffaele(55);  
    private final int eta;  
    nomieta( int v ){  
        eta = v;  
    }  
    public int eta(){  
        return eta;  
    }  
};
```

Uno sguardo al decompilato ...



Varargs

Possibilità di definire un numero di parametri variabile

```
public class varTest {  
    public static void main(String[] args) {  
        varTest x = new varTest();  
        x.testArg( "ciao", "io", "sono", "Matteo", 10, 20 );  
    }  
    void testArg(Object ... args) {  
        for( Object s : args ) {  
            System.out.println( s );  
        }  
    }  
}
```



Input/Output formattati

```
// Grazie a varargs
// Output formattato
System.out.printf("[%s] [%5d] [%5x]\n", "Matteo", 33, 2459);
```

```
// Input formattato
// BETA 1
//Scanner s=Scanner.create(System.in);
// BETA 2
Scanner s= new Scanner(System.in);
System.out.println( "Ciao come ti chiami? " );
String str = s.next();
System.out.println( "Quanti anni hai? " );
int val  = s.nextInt();
s.close();
```



Network

Introduzione dei timeout a livello di classe: URLConnection.java

```
public void setConnectTimeout(int timeout) {  
    if (timeout < 0) throw new IllegalArgumentException("...");  
    connectTimeout = timeout;  
}  
  
public void setReadTimeout(int timeout) {  
    if (timeout < 0) throw new IllegalArgumentException("...");  
    readTimeout = timeout;  
}
```

Fino a JDK 1.4: settaggio di sistema comune a tutte le istanze

- Dsun.net.client.defaultConnectTimeout=...
- Dsun.net.client.defaultReadTimeout=...



Network

- Introduzione del “PING” nella classe InetAddress

```
public boolean isReachable(int timeout) throws IOException {  
    ...  
}
```

- Miglioramento del supporto dei Cookie
- Miglioramento della gestione dei Proxy server: si possono gestire le casistiche di non connessione col server
- Fino a JDK 1.4: settaggio di sistema comune a tutte le istanze
 - Dhttp.proxyHost=...
 - Dhttp.proxyPort=...



Stack

Nuovi metodi per poter gestire lo stackTrace dei programmi

source

```
StackTraceElement e[]=Thread.currentThread().getStackTrace();  
for (int i=0; i <e.length; i++) {  
    System.out.println(e[i]);  
}
```

```
System.out.println("\n"+Thread.getAllStackTraces());
```

output

```
java.lang.Thread.dumpThreads(Native Method)  
java.lang.Thread.getStackTrace(Unknown Source)  
stackTest.main(stackTest.java:7)
```

```
{Thread[Reference Handler,10,system]=[Ljava.lang.StackTraceElement;@130c19b,  
Thread[main,5,main]=[Ljava.lang.StackTraceElement;@1f6a7b9, Thread[Signal  
Dispatcher,10,system]=[Ljava.lang.StackTraceElement;@7d772e, Thread  
[Finalizer,8,system]=[Ljava.lang.StackTraceElement;@11b86e7}
```



Memoria

Classi per la gestione della memoria di Hotspot JVM

// Cambiato nella BETA 2

```
List pools = ManagementFactory.getMemoryPoolMXBeans();
for(Object o : pools.toArray() ){
    MemoryPoolMXBean p = (MemoryPoolMXBean)o;
    System.out.println("Memory type="+p.getType()+" Memory
usage="+p.getUsage());
}
```

```
Memory type=Non-heap memory Memory usage=init = 196608
(192K) used = 438336(428K) committed = 458752 (448K)max =
33554432(32768K)
```

```
Memory type=Heap memory Memory usage=init = 524288(512K)
used = 164072(160K) committed = 524288 (512K)max = -1(-1K)
```



Pack200

Pack200 nuova modalità di compressione dei JAR
Ottimizza le strutture dei .class file

```
zip pack200 pack200+zip pack20+gzip
```

```
-----  
plugin.jar 1068615 324282 313435 132233 132352  
tools.jar 6016154 1936403 1711907 714670 717355
```

pack200 e unpack200 sono disponibili con la Beta 2



Pack200

Utile nelle applicazioni client/server

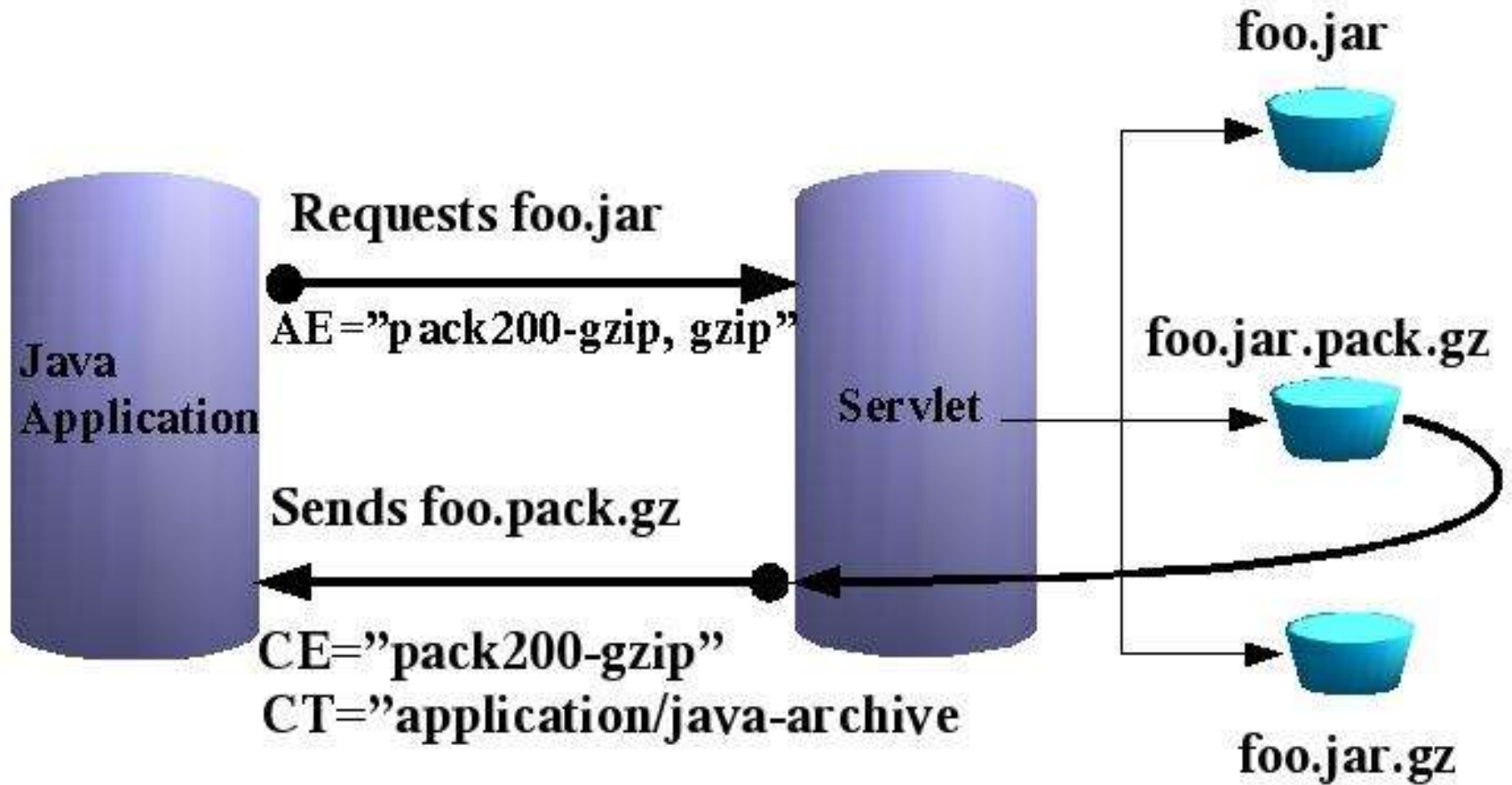
Dalla versione 1.5, HTTP compression è implementata da Java Web Start e dal Java Plug-in, come discusso da RFC 2616. Le modalità di compressione possibili sono gzip e pack200-gzip.

L'applicazione invia una richiesta HTTP al server, indicando che “Accept-Encoding” (AE) è pack200-gzip o gzip.

Il server cerca il JAR richiesto, risponde con un “Content-Encoding” (CE) indicando pack200-gzip , gzip, o NULL, in base alla richiesta e opzionalmente imposta un “Content-Type” (CT) a “application/Java-archive”. In questo modo JWS o JP conoscono il metodo di compressione dei JAR ed utilizzano la corretta decompressione



Pack200





Non abbiamo parlato di

Sicuramente da approfondire

Concurrency Utilities

Sono una serie di costrutti di alto livello che includono: executors, che sono un framework per la creazione di thread, code thread safe, Timers, locks e altre primitive di sincronizzazione



Riferimenti

- Java 1.5 download
<http://java.sun.com/j2se/1.5.0/download.jsp>
- Breve storia delle versioni di Java
<http://www.particle.kth.se/~lindsey/JavaCourse/Book/Java/Chapter01/versions.html>
- Java Community Process
<http://www.jcp.org>
- Mokabyte – j2se 1.5 beta1
http://www.mokabyte.it/2004/03/jdk1_5.htm
- Esempi di codice
http://zamples.com/JspExplorer/samples/samplesJDK1_5.html
- Dove poter scaricare questo materiale
<http://www.baccan.it>
<http://www.javaportal.it>



and now ...

Q&A



GRAZIE

da

Matteo Baccan

www.JavaPortal.it



Ringraziamenti...

- Le aziende grazie alle quali posso essere qui oggi



<http://www.grupposisge.it/>



www.JavaPortal.it