Vorlesungsreihe
Entwicklung webbasierter Anwendungen

Entwicklung webbasierter Anwendungen mit JAVA - Teil 1 -

Prof. Dr.-Ing. Thomas Wiedemann
email: wiedem@informatik.htw-dresden.de

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN (FH)
Fachbereich Informatik/Mathematik

Gliederung

Notwendigkeit von Programmierwerkzeugen

- Motivation der Entscheidung für Java

Java

- Historie , Java im Überblick
- Grundlagen der Java-Programmierung
  - Datentypen
  - Operatoren
  - Anweisungen

Quelle(n) :

- **Go To Java 2, von Guido Krüger, 2. Auflage, Handbuch der Java-Programmierung (Online verfügbar !!)**

Notwendigkeit von Programmierwerkzeugen

Im Rahmen der bisher betrachteten Technologien :

- wurde auf Basisprotokolle und verfügbare Standardwerkzeuge orientiert
- damit können bereits beträchtliche Teile von webbasierten Anwendungen abgedeckt werden (Email, Web-Interface, Kommunikation ...)

Eine Programmieroption ist jedoch generell notwendig oder sinnvoll zur :

- Verknüpfung der verschiedenen Werkzeuge und Protokolle
- Definition spezieller Geschäftslogiken
- Validierung, Konvertierung und Speicherung von Daten
- Filterung und Bereitstellung von Daten aus Dateien und Datenbanken

Spezielle Anforderungen von webbasierten Systemen :

- bei Client-Server-Anwendungen sind Programmiermöglichkeiten sowohl auf der Server- wie auch auf der Clientseite notwendig
- die Entscheidung sollte frei nach den optimalen Einsatzbedingungen und Anforderungen erfolgen und möglichst alle Browsergenerationen unterstützen

Motivation der Entscheidung für Java

- zur Lösung der genannten Aufgaben kommen eine Reihe von Programmiertechnologien in Betracht

Im Vergleich schneidet JAVA gegenwärtig am günstigsten ab :

- JAVA ist sowohl auf Server- wie auf Clientseite (Javascript) einsetzbar (andere Technologien wie PHP und Perl sind nur auf serverseitige Programmierung ausgerichtet -> siehe Seminarvorträge)
- obwohl durch SUN entwickelt und weiterhin gepflegt, ist es weitgehend unabhängig von speziellen Betriebsystemen und Hardwarevoraussetzungen
- nach einigen Optimierungen ist die Performance ist nun auch für viele praktische Aufgaben ausreichend
- die sehr umfangreichen Funktionsbibliotheken erleichtern insbesondere die Arbeit mit Internetprotokollen und –servern
- die Entwicklung mit Java verläuft relativ zügig und liefert stabile Anwendungen
- Benutzerschnittstellen (GUI´s) sind mit Java realisierbar
- Komplexe Projekte werden ebenfalls schont gut unterstützt (Enterprise-Beans)

Historie von Java

- Anfang der 90er Jahre startete SUN das "Green-Projekt" zur zukünftigen Programierung intelligenter Haushaltsgeräte (Toaster, Videorecorder, Fernseher,etc.)
- Bestandteile des Projekts war das Green-OS und ein portabler Interpreter (Oak)
- nach einem kommerziellen Mißerfolg des Greenprojektes wurde Oak als Interpretersprache innerhalb eines SUN-Browsers "Webrunner" verwendet
- die erfolgreiche Anwendung von Oak führte zur Umbenennung in HotJava
- die stabile Arbeitsweise von Hotjava führte zur Lizenzierung durch die Fa. Netscape im Dezember 1995 und wurde in Netscape 2.0 eingesetzt
- Anfang 1996 Freigabe des Java Development Kit - JDK 1.0
- sehr rascher Anstieg der Anwendungen
- Ende 1998 wurde JDK 1.2 freigegeben und als Java 2 Platform umbenannt
- Mitte 1999 erste öffentliche Betaversion des JDK 1.3 mit deutlich verbesserter Geschwindigkeit und Stabilität für verschiedene Plattformen (Windows, SOLARIS und Linux)

Java im Überblick - grundlegende Sprachmerkmale

- Java wurde vollständig neu entworfen
- wesentliche Elemente wurden von C und C++ übernommen

Zielstellung der Entwickler :

- »Java soll eine einfache, objektorientierte, verteilte, interpretierte, robuste, sichere, architekturneutrale, portable, performante, nebenläufige, dynamische Programmiersprache sein.«
- Ein Großteil dieser Ziele wurde gut umgesetzt !

Einige Vorteile von ergeben sich aus der Vermeidung von C/C++ - Problemen :

- keine expliziten Pointer, keine separaten Header-Dateien
- bessere Laufzeitüberwachung bei Matrizen und anderen dynamischen Objekten
- Automatische Speicherverwaltung bei Strings und Objekten (Garbage Collector)
- keine Mehrfachvererbung und keine Templates in Java
- sehr gutes Fehlermanagement (wesentlich stabiler als C/C++)
- sehr gutes Sicherheitsmanagement (Vermeidung unsicherer Programmierpraktiken)
- portabel auf verschiedene Rechnerplattformen

Java im Überblick - technische Realisierung

Realisierung als Interpretersprache

- Java-Quellcode wird übersetzt durch Java-Compiler (javac) in Bytecode
- Bytecode ist eine Art portabler Objektcode
- wird von einer Virtuellen Maschine (VM) interpretiert
- bei Bedarf kann Bytecode auch in nativen Code konvertiert werden (Just in Time Compiler)
- zukünftig sollen auch Prozessoren zur direkten Ausführung des Bytecodes verfügbar sein

Sicherheitsmaßnahmen :

- Sprachentwurf vermeidet gefährliche Techniken (keine Pointer)
- innerhalb der VM wird eine in sich geschlossene Umgebung erzeugt (Sandbox) , in welcher nur eine beschränkte Anzahl von Funktionen verfügbar ist
- es werden keine direkten Zugriffe (d.h. ohne entsprechende VM-Funktionen) auf das Dateisystem des Rechners oder das Betriebssystem zugelassen

Java im Überblick – Versionen und abgeleitete Sprachen

- Hauptentwicklung JAVA zur Anwendungsentwicklung
- **Javascript** als Netscape-eigene Technologie innerhalb der Browser zur Steuerung der HTML-Darstellung
  - im Gegensatz zur JAVA rein interpretativ
  - Kein kompletter Befehlsvorrat von JAVA verfügbar
  - Keine größeren Anwendungen damit realisierbar
- **VisualJ++ von Microsoft als Konkurrenztechnologie**
  - wird langfristig wahrscheinlich nicht erfolgreich sein
  - teilweise spezielle Erweiterungen durch MS
  - Starke Ausrichtung auf Windowsrechner (nicht 100% portabel)

Java : Allgemeine Syntaxregeln

- zulässiger Zeichensatz: Unicode-Zeichen, auch für Bezeichner !
- Kommentare: - einzeilig beginnend mit //
  - mehrzeilig mit /* ... */ und Dokumentation mit /** ... */
  (mit dem Tool javadoc können diese Dokum. Extrahiert werden)
- Java-Bezeichner können beliebig lang sein, alle Stellen sind signifikant !
- Groß- und Kleinschreibung ist signifikant !
- Bezeichner müssen mit einem Unicode-Buchstaben beginnen (das sind die Zeichen 'A' bis 'Z', 'a' bis 'z', '_' und '$') und dürfen dann weitere Buchstaben oder Ziffern enthalten.
- Unterstrich und Dollarzeichen sollen nur aus historischen Gründen bzw. bei maschinell generiertem Java-Code verwendet werden.

Wesentliche Unterschiede zu C/C++

- kein Präprozessor und damit keine #define-, #include- und #ifdef-Anweisungen
- Backslash \ verkettet keine aufeinanderfolgenden Zeilen
- mit + verkettete konstante Strings werden zu einem String zusammengefasst

Java-Datentypen : Primitive Datentypen

- Java kennt acht elementare Datentypen, die gemäß Sprachspezifikation als primitive Datentypen bezeichnet werden. Alle Angaben sind verbindlich !

Typname	Länge	Wertebereich	Standardwert
boolean	1	true, false	false
char	2	Alle Unicode-Zeichen	\u0000
byte	1	-128...127	0
short	2	-2^{15} ...$2^{15}-1$	0
int	4	-2^{31}...$2^{31}-1$	0
long	8	-2^{63}...$2^{63}-1$	0
float	4	+/-3.40... * 10^{38}	0.0
double	8	+/-1.79... * 10^{308}	0.0

Besonderheiten im Vergleich zu C/C++

- boolean kann i.d.R. nicht mit int ersetzt werden (bei if ...)
- ES FEHLEN : explizite Zeiger, Typdefinitionen (typedef) , Aufzählungen, (enum) , Recordtypen (struct und union) , Bitfelder - Ersatz durch Javamittel möglich

Java - Variablen

Java kennt 3 Typen von Variablen:

- Instanzvariablen, die im Rahmen einer Klassendefinition definiert und zusammen mit dem Objekt angelegt werden.
- Klassenvariablen, die ebenfalls im Rahmen einer Klassendefinition definiert werden, aber unabhängig von einem konkreten Objekt existieren.
- Lokale Variablen, die innerhalb einer Methode oder eines Blocks definiert werden und nur dort existieren.
- Eine Variable in Java ist immer typisiert. Sie ist entweder von einem primitiven Typen oder von einem Referenztypen abgeleitet. Mit Ausnahme eines Spezialfalls werden alle Typüberprüfungen zur Compile-Zeit vorgenommen.
- Java ist damit im klassischen Sinne eine **typsichere Sprache**

Deklaration von Variablen an beliebiger Stelle im Code mit :

Typname Variablenname; Typname Variablenname = Initialwert ;

Bsp.: int a; char b = 'x';

Java - Arrays

- Arrays in Java sind Objekte und werden zur Laufzeit erzeugt !
- Damit sind Array-Variablen Referenzen und besitzen Arrays Methoden und Instanz-Variablen. besitzen
- Arrays sind semidynamisch, d.h. ihre Größe kann zur Laufzeit festgelegt, später aber nicht mehr verändert werden.

Deklaration und Initialisierung von Arrays

1. Deklaration einer Array-Variablen mit [] - Klammern (zum Halten der Referenz)
   int[] a; double[] b; boolean[] c; int[][] aa; // analog mehrdimensional
2. Initialisierung mit new oder Wertzuweisung (auch bereits bei Deklaration)
   a = new int[5]; b = new double[10]; int[] a = new int[5];
   int[] x = {1,2,3,4,5}; boolean[] y = {true, true};
3. Zugriff - Elemente werden von 0 bis n-1 durchnumeriert
   a[2]= 3; a[4]= a[0]+a[3];
   Der Array-Index muß vom Typ int sein und muß größer 0 und kleiner als die Länge des Arrays (abrufbar über Instanzvariable length) sein.

Java – Referenztypen

- zu den Referenztypen gehören Objekte, Strings und Arrays
- die Konstante null definiert eine leere Referenz

Wichtige Eigenschaften von Referenztypen

- müssen mit Hilfe des new-Operators angelegt werden
- in Java erfolgt der Zugriff auf Referenztypen in der gleichen Weise wie der auf primitive Typen (ein Dereferenzierungsoperator & existiert nicht)
- die Zuweisung einer Referenz kopiert lediglich den Verweis auf das betreffende Objekt, das Objekt selbst dagegen bleibt unkopiert
- nach einer Zuweisung zweier Referenztypen zeigen diese also auf dasselbe Objekt
- echtes Kopieren erfordert Aufruf der Methode clone
- Gleichheitstest zweier Referenzen testet, ob beide Verweise gleich sind
- Test auf inhaltliche (Un)Gleichheit erfolgt mit equals-Methode
- Nicht mehr verwendete **Referenztypen werden in** ein automatisches Speichermanagement freigegeben (im Hintergrund arbeitender Garbage Collector sucht periodisch nach Objekten, die nicht mehr referenziert werden)

Java – Operatoren

- Mathematische Operatoren

  \+ - * / %-Rest

  ++ -- als Post- und Prä-inkrement/decrement
- Vergleichsoperatoren

  == >= <= > < != (einfaches = führt in if ( ) zu Warnung)
- Logische Operatoren – nur zur Verarbeitung von Boolesche Werten !

  ! -Not ^ -Xor & -Und | -oder && || mit Abbruch bei Ergebnis
- Bitweise Verknüpfung

  ! - Inversion ^ -Xor & -Und | -oder

  Schieben: >>> Rechtsschieben ohne Vz. >> mit Vz. <<
- Zuweisungsoperatoren

  += -= *= >>>= ...

Java – Anweisungen für Verzweigungen

- Leere Anweisung mit ; (Vorsicht bei while( ) ;
- Zusammenfassung zu Blockanweisungen mit { anw1; anw2; ...; } - kann anstelle einer einfachen Anweisung stehen

Verzweigung mit if

- Syntax : if (ausdruck) anweisung; // oder
  if (ausdruck) anweisung1; else anweisung2;
- C++ -Abweichung: Der Wert ausdruck muß den Typ boolean haben !
- Ersatz für bedingtes Kompilieren : if (false) anweisung; // NICHT SICHER !! da Compilerabhängig

Verzweigung mit switch

- Syntax : switch (ausdruck)
  { case constant: anweisung; break;
  case ... : ... break;
  default: // falls nicht in der case-Auflistung
  }

Java – Schleifenanweisungen

while / do while –Schleifen :

- Syntax : while (ausdruck) anweisung; // kopfgesteuert
  do anweisung; while (ausdruck); Syntax :

for-Schleifen

- Syntax for (init; test; update) anweisung;
  - init-Ausdruck dient zur einmaligen Initialisierung (darf auch Var.deklaration enthalten und diese ist dann nur innerhalb des Blocks gültig)
  - test-Ausdruck prüft auf Laufzeitbedingung, falls true wird Schleife fortgesetzt
  - update-Ausdruck dient zur Veränderung des Schleifenzählers

Allgemein für alle Schleifen

- break bricht die gesamte Schleifenausführung ab
- continue bricht die aktuelle Schleife ab und setzt mit nächster Schleife fort