Java User Group Karlsruhe

jug-karlsruhe@googlegroups.com

http://groups.google.com/group/jug-karlsruhe

Neues in Java SE 6

Dr. Stefan Schneider
CTO ISV Engineering APAC-EMEA
Sun Microsystems GmbH

Revision 1.0
4.7.2007

The Global Power of Sun

Fortune
187
Company

Annual Revenues
$13+ Billion

Patents
11,000+

Java Devices
5.5+ Billion

Annual R&D
~$2 Billion

Solaris 10
Licenses
7.6+ Million

Java Developers
6+ Million

Quarterly Revenue
Up 3.3% Y/Y

Annual Storage
Petabytes Shipped
410

Worldwide Employees
34,500

Cash
$5.486 Billion

Business Presence
100 Countries

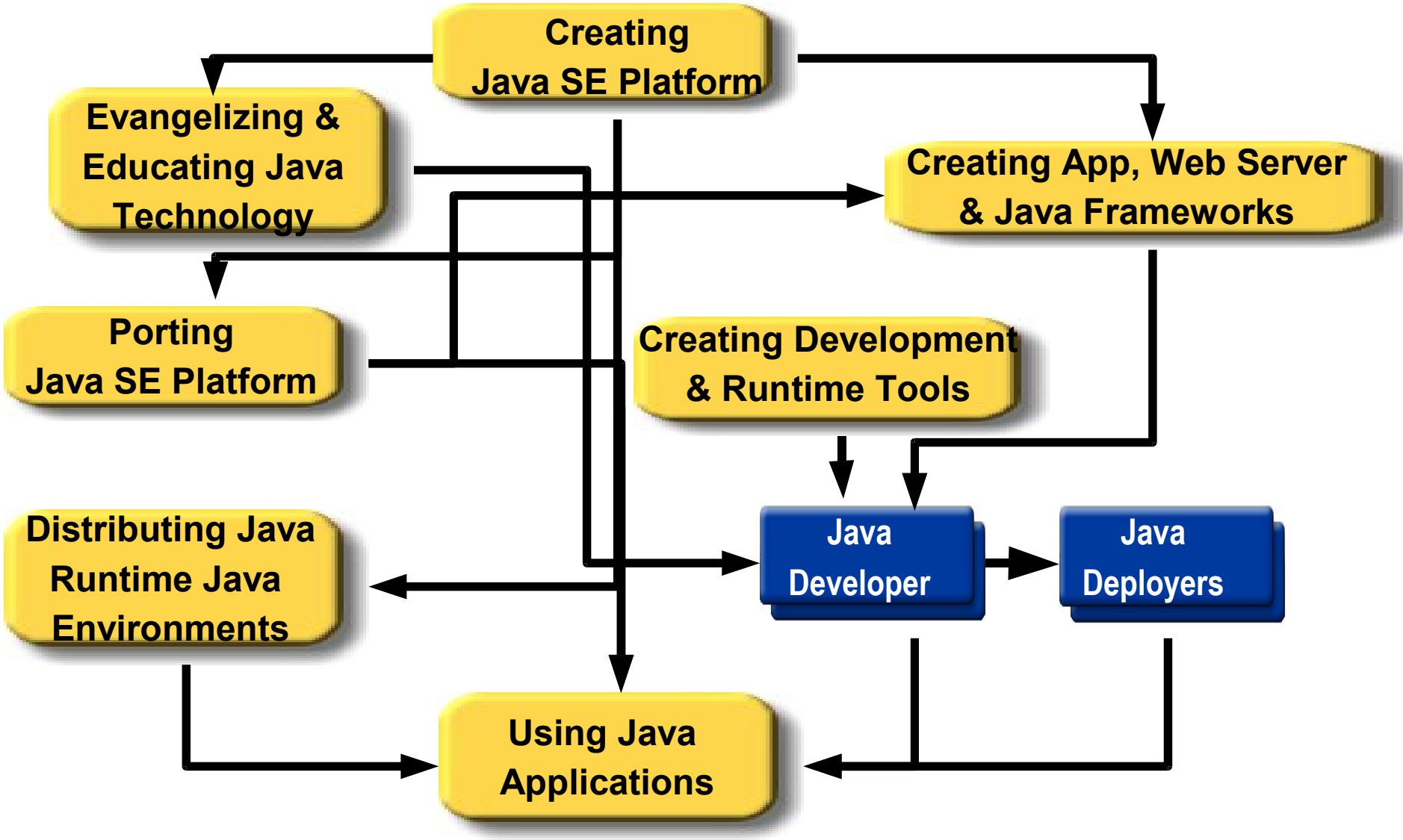
Business Update

- Profitability – 2 consecutive quarters of growth and profitability – software up 12% and services up 7% Y/Y in Q3FY07
- 5 consecutive quarters of server revenue market share growth – outpacing the overall market in Y/Y growth
  (Source: IDC Worldwide Quarterly Server Tracker, May, 2007)
- Intel endorses Solaris as ***the*** mission critical OS for Xeon servers; Sun to deliver Xeon-based servers
- Formed Microelectronics Group; announced first relationship with Marvell Technology Group
- Stanford University Linear Accelerator Center adds Project Blackbox to rapidly expand its physics and astrophysics research

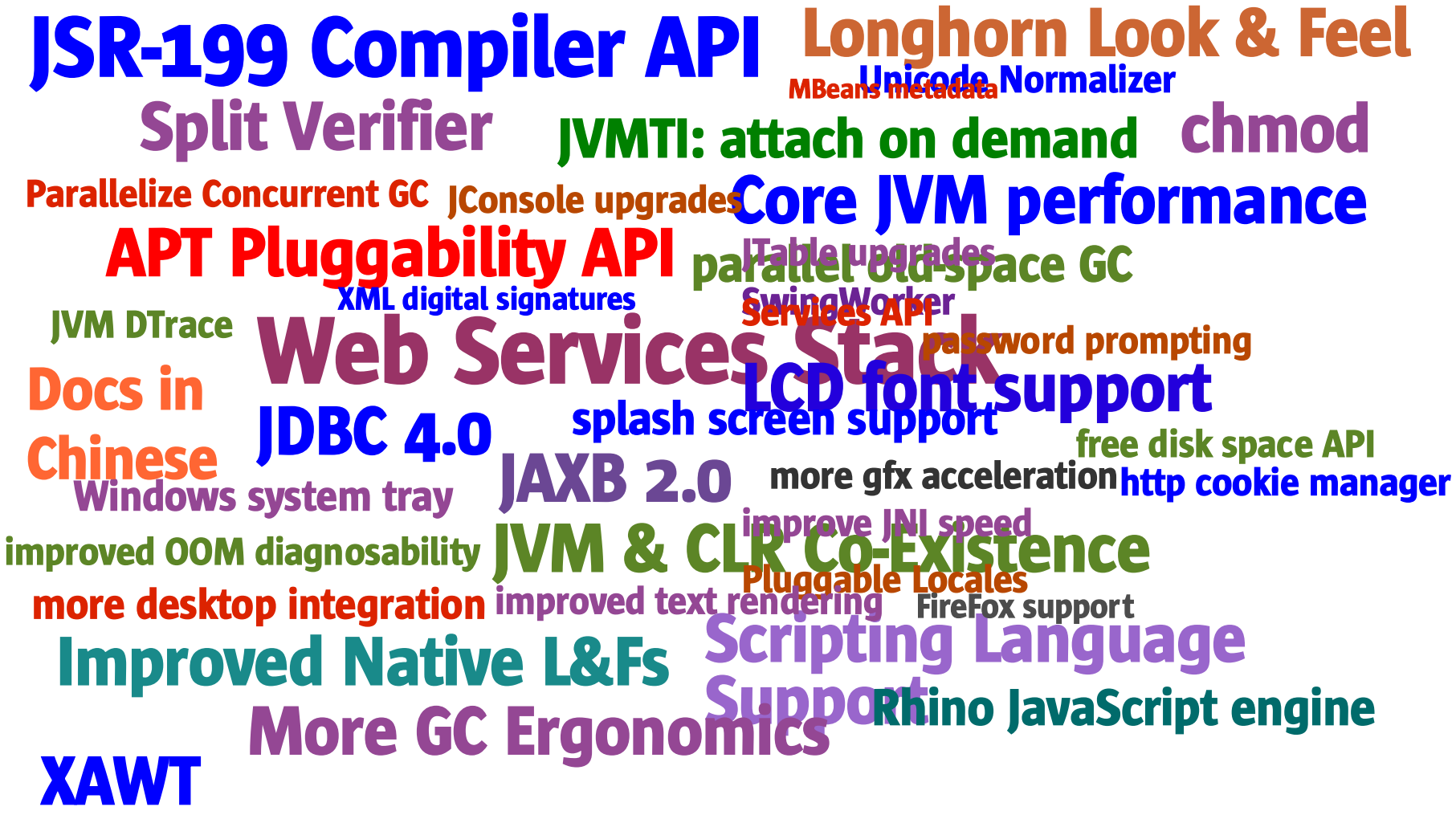
Agenda

- Generelles zu Java 6
- Ausgewählte Themen
  - Desktop Java
  - Scripting
  - Monitoring und Management
  - JVM Internas
  - XML/Web Services
  - Datenbanken (JDBC 4.0, Java DB)
- Ausblick auf Java 7

Anforderungen an eine SE Plattform



Einige ausgewählte Eigenschaften...



Java SE 6

- Java SE 6 FCS Freigabe: 11. Dezember 2006
  - Java SE 6 enthält keine neuen Spracherweiterungen
  - Inhaltliche Schwerpunkte
    - Desktop
    - XML/Web Services
    - Ease of Development
    - Leistung
  - Erste Version mit komplett öffentlicher Entwicklung
    - Wöchentliche Source & Binary Snapshots
    - Zahlreiche Bugfixes aus der Community!

Java SE 6 Themen
mit eigenständigen JSR Spezifikationen

JDK6 Umbrella Spec – JSR 270

XML & Web Services

JAX-B 2.0 (JSR-222)
JAX-WS 2.0 (JSR-224)
WS Metadata (JSR-181)
StAX API (JSR-173)
XML Signature (JSR-105)

Ease of Development

Common Annotations (JSR-250)
Scripting API (JSR-223)
JDBC 4 (JSR-221)
Plug. Annotations (JSR-269)

Miscellaneous

Java Compiler API (JSR-199), Class File Update (JSR-202)

Java SE 6 - Java für den Desktop

- Java SE 6 und Dolphin (Java 7) verstärken den Fokus auf Desktop Java
  - Integration von Komponenten aus den SwingLabs Projekten (swinglabs.org)
  - Erweiterung der Deployment Eigenschaften
  - Stabilität, “Look & Feel”
  - Mehr Eigenschaften sind für das Dolphin (SE 7) geplant
    - Beans Binding Framework (JSR-295)
    - Swing Application Framework (JSR-296)
    - Deployment Modules: Java Module System (JSR-277)
    - Development Modules: “Super Packages” (JSR-294)

Desktop Java - Highlights

Neue Schnittstellen: Tray Icons, Splash Screens, Desktop Helper API, Group Layout Manager, Swing Worker, Jtable Sortieren, Management dokumentmodaler Dialoge

Leistung:

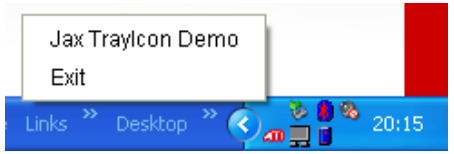
- Echtes Double Buffering
- Mehr HW-beschleunigung
- Single Threaded Rendering für OpenGL

Look & Feel:

- Windows Vista Unterstützung
- Native Rendering für GTK/Win
- LCD Text (Subpixel Rendering)

Desktop Java – „Tray Icons“

- “TrayIcons” sind Minianwendungen die in einem speziellen Bereich durch den Window Manager verwaltet werden
  - haben kein eigenes Fenster
  - bestehen aus Ikone und Kontextmenü
  - Beispiel: Startleiste in Windows
- Neue Klassen java.awt.TrayIcon/SystemTray
  - TrayIcon Anwendung registrieren ein Bild und ein Menü beim SystemTray
  - SystemTray interagiert mit dem jeweiligen WindowManager



Desktop Java – Tray Icons

```
TrayIcon trayIcon = null;
if (SystemTray.isSupported()) {
    SystemTray tray = SystemTray.getSystemTray();
    Image image = Toolkit.getDefaultToolkit().getImage("...");
    PopupMenu popup = new PopupMenu();
    MenuItem defaultItem = new MenuItem("Jax TrayIcon Demo");
    ActionListener listener = new ActionListener() {
        public void actionPerformed(ActionEvent e) {.....}
    };
    defaultItem.addActionListener(listener);
    popup.add(defaultItem);
    trayIcon = new TrayIcon(image, "JFS Tray Demo", popup);
    trayIcon.addActionListener(listener);
    try {
        tray.add(trayIcon);
       } catch (AWTException e) { ...}
} //....
```

Desktop Java – Splash Screens

- JDK6 bietet Unterstützung für Splash Screens die beim Start der Anwendung eingeblendet werden
  - Steigerung der “wahrgenommenen” Leistung
  - Unterstützung für GIF,JPEG,PNG, Animated Gif
  - JVM Kommandozeile: java -splash:myimage.gif ...
  - Manifest Eintrag

    ```
    SplashScreen-Image: filename
    ```

  - Splash Bild wird noch vor JVM Bootstrap Code geladen
- **`java.awt.SplashScreen`** für Kontrolle des SplashScreens durch die Anwendung
  - Erlaubt Fortschrittsanzeige während sich Anwendung initialisiert

Desktop Java – Splash Screens

```
/* Start der Applikation mit java -splash:file.gif
 * oder über META-INF/MANIFEST.MF Eintrag
 */

public static void main(String[] args) {
    //SplashScreen anfordern
        SplashScreen screen = SplashScreen.getSplashScreen();
    //G2D Objekt zur Manipulation
        Graphics2D g = screen.createGraphics();
        Font defaultFont = g.getFont();
        g.setFont(new Font(defaultFont.getName(), Font.BOLD, 20));
        g.setColor(Color.RED);
        g.drawString("Application ready!", 30, 30);
        screen.update();
    screen.close()
}
```

Desktop Java – Desktop APIs

- Über die **`java.awt.Desktop`** Klasse können Java Anwendungen assoziierte native Anwendungen für Dateitypen starten
  - Start des Default Browser
  - Start des E-Mail Clients
  - Start der Standardanwendung um Dateien anzuzeigen, zu drucken oder zu editieren
- Achtung: Betriebsystemunterstützung notwendig!
  - Methode: **`Desktop.isDesktopSupported()`**

Desktop Java – Desktop API

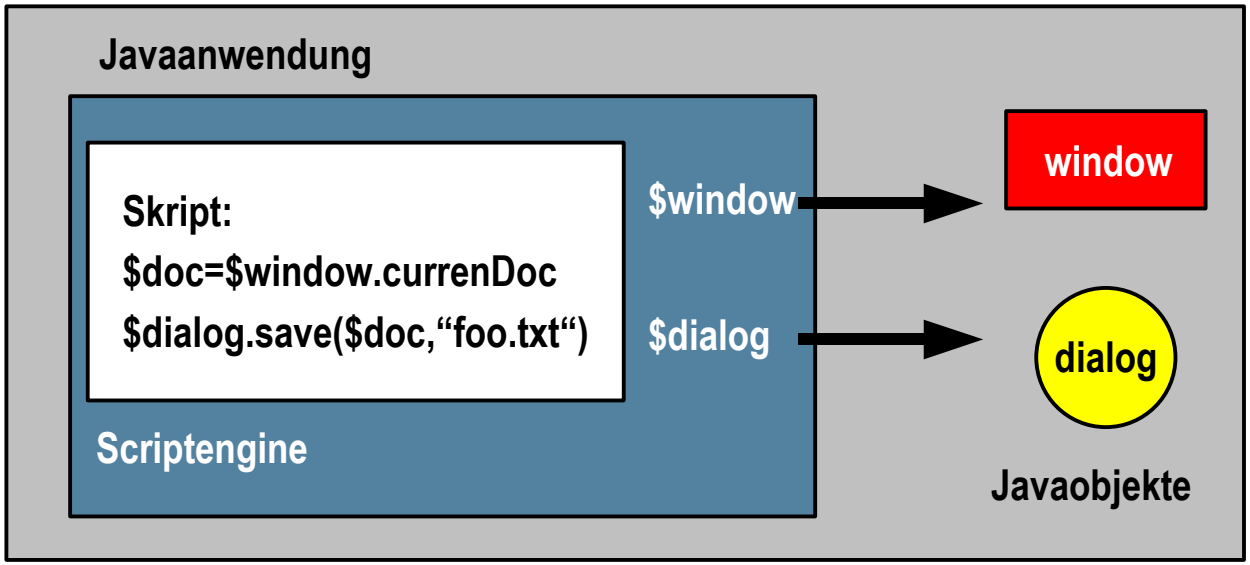
```
/* Desktop API ist nicht auf allen OSen bzw.
 * Window-Managern unterstützt
 */

if (Desktop.isDesktopSupported()){
   Desktop dt = Desktop.getDesktop();
   try {
       dt.open(new File ("C:\\duke.gif"));
       dt.edit(new File ("C:\\test.txt"));
       dt.print(new File ("C:\\doc1.pdf"));
       if (dt.isSupported(Desktop.Action.BROWSE))
           dt.browse(new URI("file:///C:/index.html"));

        } catch (Exception ex){
       ex.printStackTrace();
      }
```

Scripting in Java: Motivation

- Offenheit für zukünftige Konzepte (Web 2.0)
- Dynamischere Anwendungen
- Leichtere Anpassung von Anwendungen beim Anwender
- Leichtere Zugang für wenig geübte Programmierer
- Kombination der Flexibilität von Skriptumgebungen mit der reichen Java SE und EE Infrastruktur



Scripting in Java

- Skriptsprachen und Java rücken stärker zusammen!
  - Scripting Engine Integration über Brücke in Java SE 6
  - Neuer Bytecode “invokedynamic” in Java SE 7 für dynamisch typisierte Sprachen (JSR 292)
  - Erweiterungen in Java EE geplant bzgl. Co-Packaging
- Implementierung des `javax.script` API (JSR 223)
  - Unterstützung für Fremdsprachen die JSR 223 implementieren
- Das Sun JDK 6 enthält bereitsdie Mozilla Rhino JavaScript Engine
- Weitere Skriptsprachen/engines integrierbar

Scripting in Java

- Scripting Integration ist bidirektional
  - Javaobjekte rufen externe Skripte auf
  - Skripte rufen Javaobjekte auf
- Neues Kommandozeilen Werkzeug `jrunscript` als einfache Scripting Shell
- Mögliche Anwendungsfälle von Skripten
  - Anwendungstests
  - Extern beinflussbare Parameter (Formeln, Kontrolllogik/Workflow usw.)

Scripting in Java - jrunscript

- jrunscript erlaubt interaktive Benutzung von Skripten mit Java
  - Default: JavaScript, aber auch Groovy,BeanShell, JPython möglich

```
js> importPackage(Packages.javax.swing)
js> var frame = new JFrame(„JFS2006-Demo“)
js> var bt1 = new JButton(„ClickMe“)
js> frame.add(bt1,“North“)
js> frame.pack()
js> frame.show()
```

Skripte in Javacode

- Einfacher Aufruf eines Skripts aus Java

```
ScriptEngineManager sem = new ScriptEngineManager();
ScriptEngine engine = sem.getEngineByName("js");
try {
        engine.eval(new FileReader("hello.js"));
       } catch (Exception ex) { ...}
```

- Aufruf eines Skripts mit “Bindings”

```
ScriptEngineManager sem = new ScriptEngineManager();
ScriptEngine engine = sem.getEngineByName("js");
SimpleBindings bind = new SimpleBindings();
//Binding der Variable datum zur Verwendung im Script
bind.put("datum", new Date());
engine.setBindings(bind, ScriptContext.ENGINE_SCOPE);
engine.eval(new FileReader("hello.js"));
```

Skripte in Javacode

- Aufruf einer Skriptfunktion in Java

```
ScriptEngineManager sem = new ScriptEngineManager();
ScriptEngine engine = sem.getEngineByName("js");
engine.eval(new FileReader("mycalulator.js"));
//Nicht von allen Engines supported
Invocable inv = (Invocable)engine;
//Aufruf liefert immer Typ Object
Object result = inv.invoke("calc",10,10);
//Typ Mapping abhängig von Engine
Double d = (Double)result;
```

```
//JS
function calc(x,y){
    return x+y
}
```

Das Projekt `scripting.dev.java.net`

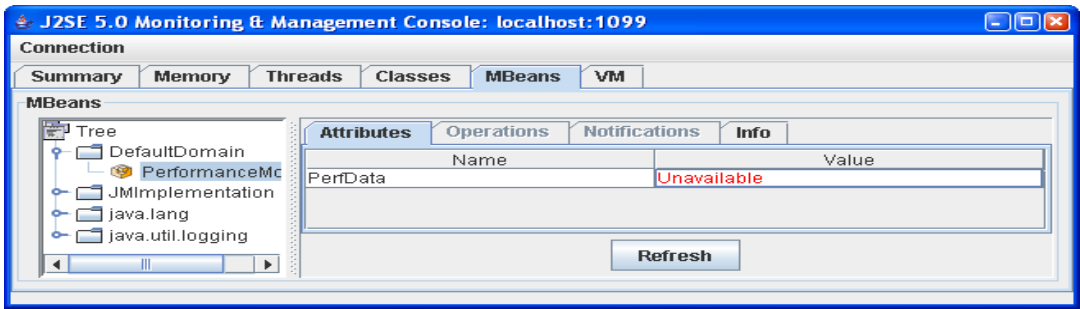
- BSD Lizenz für “script engine” Quellen
- Stellt Implementierung von javax.script.ScriptEngine
- Enthält Sammlung von Referenzen auf “script engines”
- Enhält Beispiele für die Laufzeitumgebung
- Entwickler sind eingeladen mitzumachen !
- z.Zt. verfügbar:
  - AWK
  - BeanShell
  - Ejs
  - FreeMarker
  - Groovy
  - Jaskell
  - Java
  - JavaScript
  - jst
  - JudoScript
  - JUEL
  - OGNL
  - Pnuts
  - Python
  - Ruby
  - Scheme
  - Sleep
  - Tcl
  - Velocity
  - XPath
  - XSLT
  - Jelly
  - JEP
  - Jexl

Management und Wartbarkeit

- Management und bessere Wartbarkeit waren bereits Fokus für Java SE 5.0 gewesen
  - JMX Integration und JVM Instrumentierung, Werkzeuge (jps, jstat, jmap, JConsole)
- Java SE 6 erweitert Managementeigenschaften
  - Neuer MBean Typ: MXBeans
  - Deskriptorunterstützung über Annotations
  - Neue Metaannotation **`@DescriptorKey`** um Bean-attribute mit Metainformationen zu versehen
  - “Generifikation” von einigen Methoden z.B. **`MBeanServer.queryNames()`**
  - Helferklasse JMX mit statischen Methoden um Proxies für MXBeans zu erzeugen

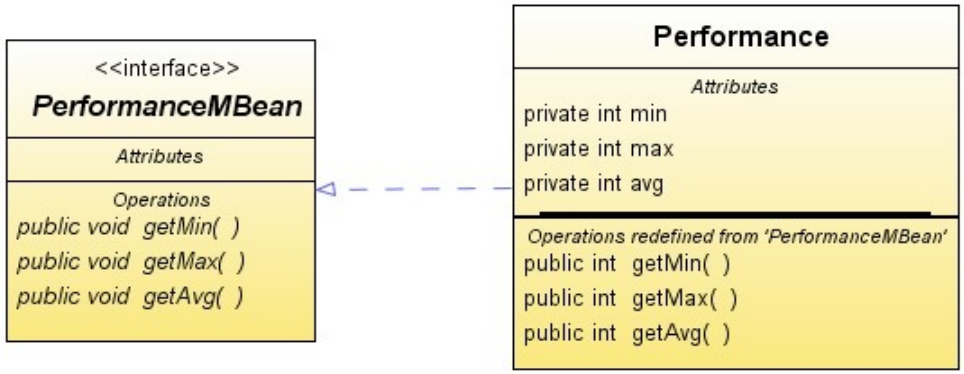
Management: MXBeans

- MXBeans sind ein neuer Typ von JMX MBeans
  - Aufbau: Schnittstelle mit Prefix <Name>MXBean, Klasse <Name>
  - JMX Infrastruktur bildet Parameter/Return Types auf JMX OpenTypes wie CompositeDataSupport ab
  - Type Mapping beschrieben in MXBean Spec
- Weshalb MXBeans?
  - MBeans liefern oft zusammenhängende Werte
    - z.B. min/max/avg Werte für Requeststatistiken oder Pools
    - Werte sind meist primitive Typen
  - MBeans Client muss das Backend Modell kennen

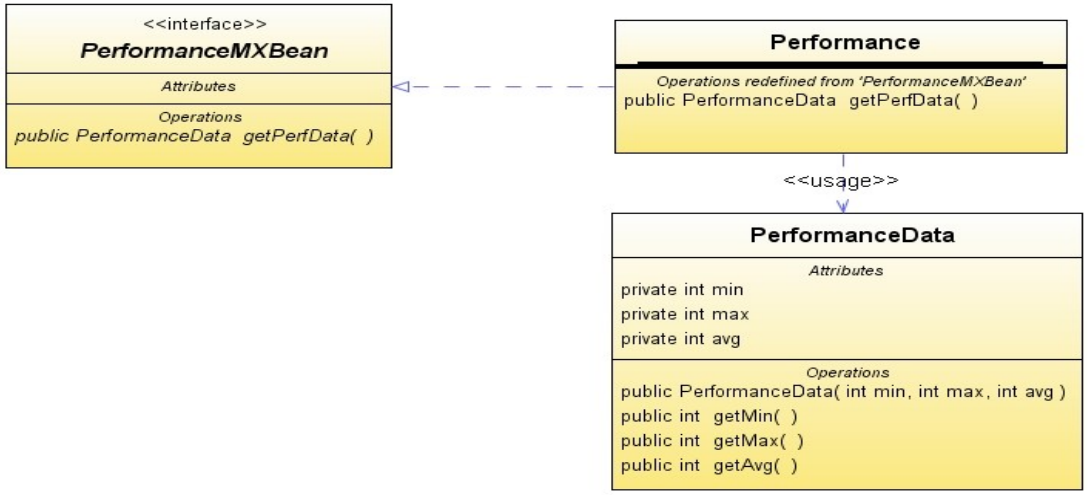


Management - MXBeans

- Beispiel (Standard MBean)
  - > MBean hat einzelne Attribute für Leistungsdaten

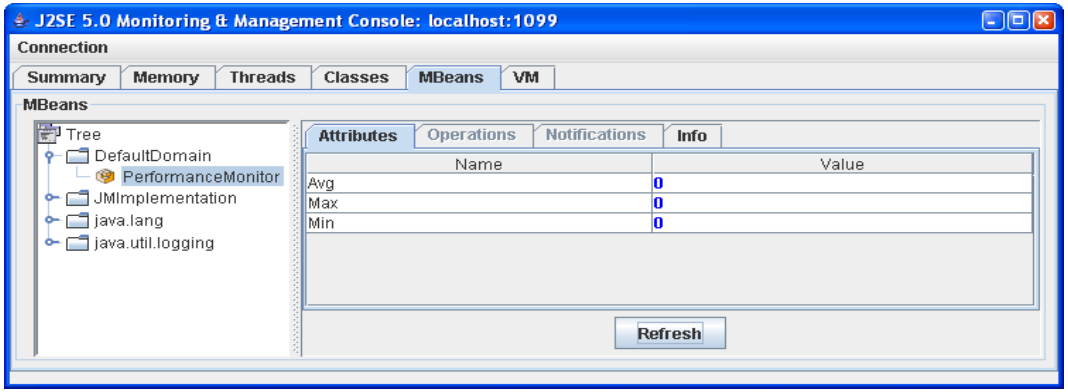


- Beispiel mit MXBean
  - > Zusammenhängende Daten werden in Hilfsobjekt aggregiert

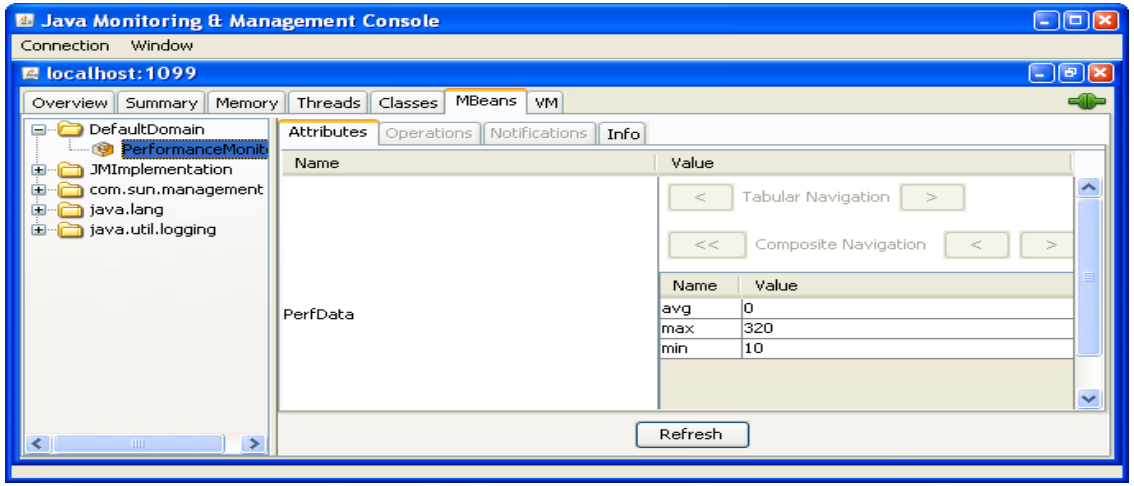


Management - MXBeans

- Beispiel (Standard MBean)
  - > JConsole für Standard MBean



- Beispiel mit MXBean
  - > JConsole mit MXBean und Containerobjekt



Java SE 6 - JVM Interna

- Classpath Wildcards für Kommandozeilenstart
  - Matched jar Files – nicht rekursiv
  - **`java -cp lib/* <Main>`**
- Neuer, schneller Verifier: Split-Verifyer
  - Details: JSR202, https://jdk.dev.java.net/verifier.html
- Standardübersetzer API – JSR 199
  - Schafft Java API zum Steuern des Compilers
- “JVM Attach on Demand” für JConsole
  - Ermöglicht dynamischers Monitoring jeder Applikation ohne `-Dcom.sun.management.jmxremote` Flag

Java SE 6 - JVM Interna

- Signal SIGQUIT Verbesserungen (CTRL-BREAK)
  - Liefert nun Thread Status und Heapinformation
  - Sperrinformationen für java.util.concurrent
- Verbesserte “OutOfMemory” Fehlerbehandlung
  - `Exception in thread "main" java.lang.OutOfMemoryError: Java heap space`
  - `Exception in thread "main" java.lang.OutOfMemoryError: Perm space`
  - `Exception in thread "main" java.lang.OutOfMemoryError: Out of swap space?`
- Neue JVM Parameter
  - `-XX:OnOutOfMemoryError=<script>`
  - `-XX:+HeapDumpOnOutOfMemoryError`
- Nur Solaris:: DTrace Probes in der JVM
  - Ermöglicht Analyse von GC, Threads, Monitoren...

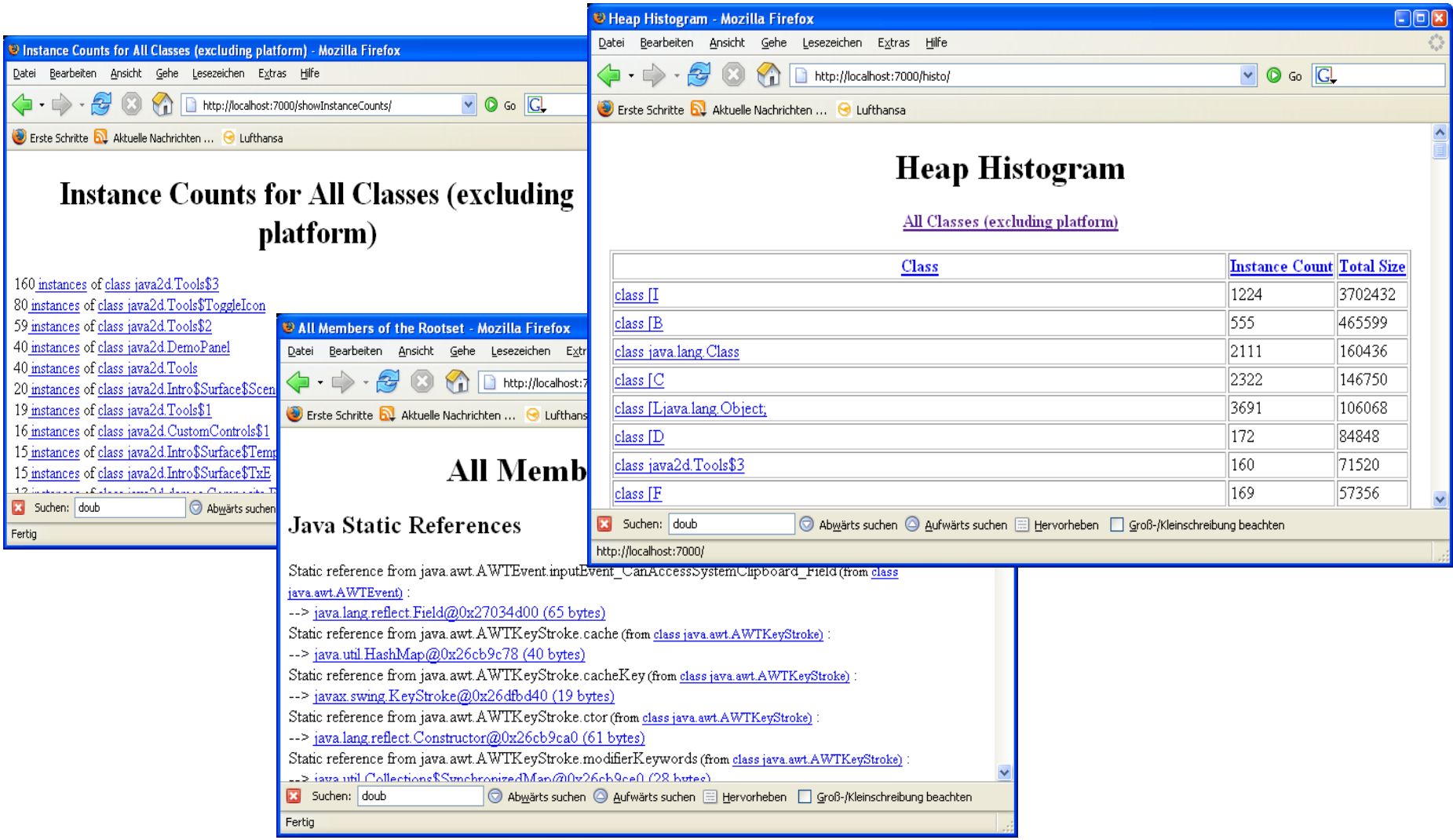
Java SE 6 - JVM Interna

- Heap Dump Analyse (nun auch für Windows)
  - jmap -histo <pid> liefert ein Heap Histogramm
  - num #instances #bytes class name

```
----------------------------------------
 1:     1298     5571944        [I
 2:      555      469824        [B
12:      160       72960        java2d.Tools$3
15:     2298       55152        java.lang.String
```

  - jmap -dump:format=b,file=heap.dmp <pid> liefert Heap Dump einer laufenden JVM
  - jhat <dumpfile> startet Heap Analyse Tool
    - Mini Web Server auf localhost:7000

Java SE 6 - JVM Interna



XML & Web Services in JDK6

- JAX-WS 2.0 API
  - SOAP basierte Web Services
- JAX-B 2.0 API
  - XML Databinding
- XML API für digitale Signaturen (JSR 105)
- Streaming API für XML (StAX- JSR 173)

JAX-WS 2.0

- Neue JAX-RPC 2.0 Spezifikation umbenannt in JAX-WS 2.0 (Java API for XML Web Services, JSR 224)
  - Implementiert eine komplett neue WS Infrastruktur
  - Enge Integration mit JAX-B 2.0 für Mapping
  - Verwendung von Streaming API (StAX JSR 73) und JAXP 1.3
  - Entworfen für zukünftige Erweiterungen (WS-*)
  - Durchgängige Verwendung von Annotations für höhere Dienste
    - Portabilität & bessere Leistung
  - “Unterstützung von Fast Infoset”

JAX-WS 2.0 API in JDK6

- JAX-WS ist die Weiterentwicklung von JAX-RPC
  - Verwendung von Codeannotierungen für einfachere Implementierung
  - Java SE 6 bietet Client + Server Unterstützung für JAX-WS
  - Code Annotationierungen sind durch JSR-181 („Web Service Metadata“) definiert
  - „Configuration by Exception“ - API arbeitet mit vielen default Settings
    - @WebService() -> Name des Service = Klassenname
- Ausgangsbasis für Web-Service-Erstellung kann sein:
  - bestehendes WSDL Dokument
    - Java Skeleton Generierung über **`wsimport`** Tool
  - Java Implementierung
    - Generierung WSDL aus Java Code per **`wsgen`** Tool

JAX-WS Web Services in JEE 5

- JAX-WS unterstützt POJOs und EJBs als Serviceendpunkt-Implementierungen
- Javaklasse wird **`@WebService`** annotiert
  - Wird zum **`wsdl:port-type`** Element im WSDL
  - Standard: Jede public Methode wird zur WS Operation
- Methoden können selektiv mit **`@WebMethod`** annotiert werden
  - Werden zu **`wsdl:operation`** Elementen im WSDL

JAX-WS 2.0 Beispiel

- Trivialer Echo Service

```
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
@WebService
@SOAPBinding(style=SOAPBinding.Style.RPC)
public class MyService {
    public String echo(String param){
        return "Hello " +param;
    }
}
```

- Endpoint Aktivierung
  - Benutzt eingebauten HTTP Server von Java SE 6

```
public static void main(String args[]){
  Endpoint.publish("http://host:8080/echo",
                   new MyService());
  }
```

JAX-WS 2.0 Beispiel

- Erzeugung der Client Artefakte

```
<jdk6>/bin/wsimport -d <dir> http://localhost:8080/echo?wsdl
```

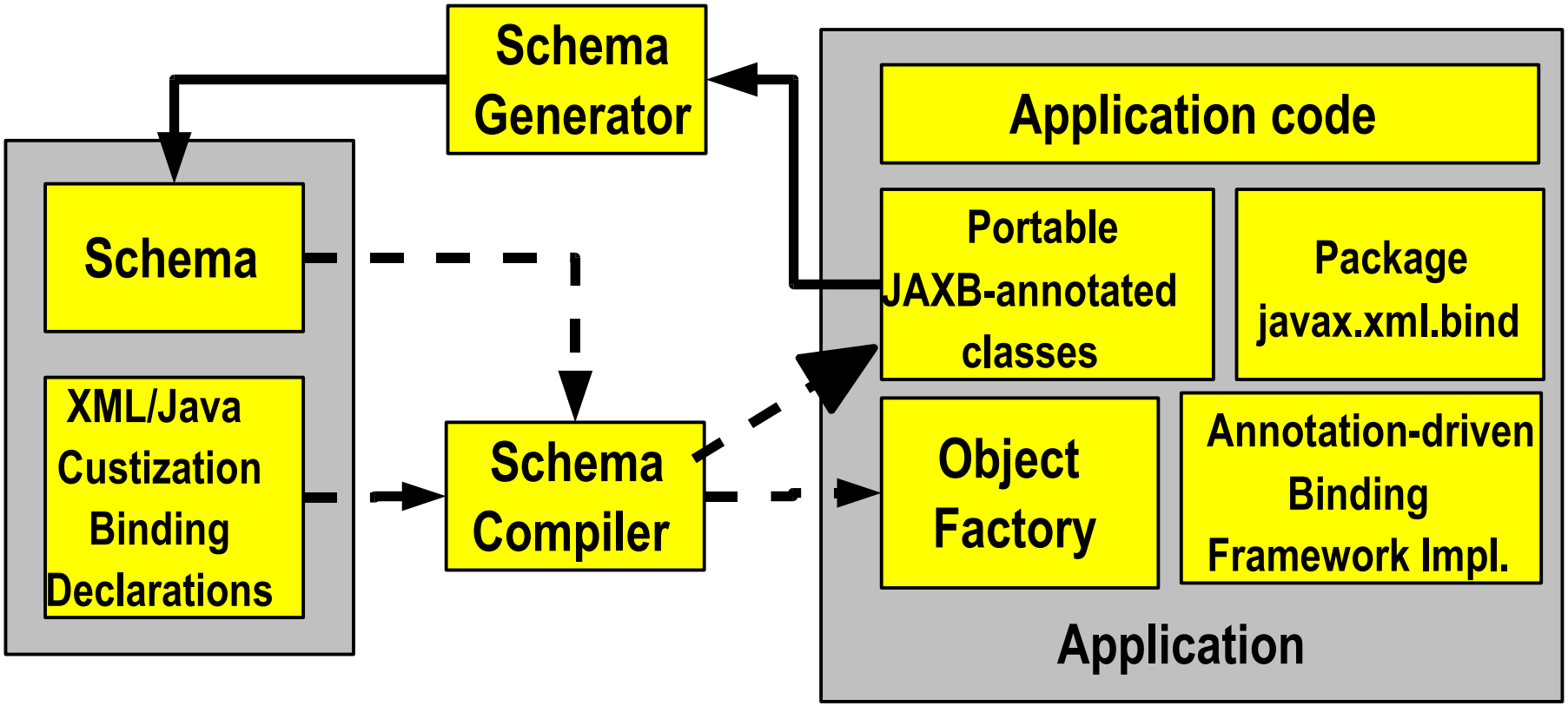
- Client Code

```
public static void main(String[] args) {
        MyService svc = new MyServiceService()
                    .getMyServicePort();
        String ret = service.echo("JFS2006");
        //...
 }
```

XML Databinding mit JAX-B 2.0

- JAX-B 2.0 ist eine komplette Überarbeitung
  - JAX-B 2.0 bietet nun 100% Unterstützung für XML Schemas
  - Drastische Reduktion des generierten Codes über Annotations und höhere Leistung
  - JAX-B 1.0 : Schema ➞Java only
    - JAXB Schema Compiler, Generierte Code nicht änderbar
  - JAX-B 2.0 : Auch Java ➞ XML + schema compiler
    - JAXB 2.0 = XML Persistenz für POJOs
- Java <-> XML Mapping über Annotationen
  - Schemaübersetzer ist eher ein Skeleton Generator
  - Generierter Code kann nach Belieben modifiziert werden
    - Anpassung über Inline Annotations im Schema
    - Externes Binding File

JAX-B Architektur



JAX-B 2.0 vs 1.0

- XML Dokument:
  `<point><x>1</x><y>2</y></point>`
  - > JAX-B 1.0: 308 LoC, 38 Files, ~220kb Code
  - > JAX-B 2.0: 62 LoC, 2 Files, 3kb Code

```
@XmlAccessorType(FIELD)
@XmlType(name = "", propOrder = {"x","y"})
@XmlRootElement(name = "point")
public class Point {
    protected float x;
    protected float y;
    public float getX() {
        return x;
    }
    public void setX(float value) {
        this.x = value;
    }
    public float getY() {
        return y;
    }
   public void setY(float value) {
        this.y = value;
    }
}
```

JAX-WS und JAXB Integration

- Zur Entwicklungszeit (wsgen/wsimport Werkzeuge)
  - JAXB generiert Java Typen von WSDL Schemas
  - JAXB generiert WSDL Schemas von Java Typen
- Zur Laufzeit
  - JAX-WS un/marshalled die Nachricht(soap:envelope)
  - JAXB un/marshalls der Nutzlast (soap:body child, soap:header, soapfault Elements)
  - StAX Parser

Streaming API for XML (StAX)

- Nachteile traditioneller XML Parsing Ansätze:
  - DOM: Speicherintensiv, langsam
  - SAX: Callback-Ansatz gibt Anwendung wenig Kontrolle über das Parsen
- Streaming API ist ein Pull-Parser-Ansatz
  - Client hat volle Kontrolle des Parsingvorgangs
  - Event Filter, Namensraumunterstützung
  - Geringer Speicherverbrauch, hohe Leistung
- StAX sieht zwei Modelle vor: Cursor vs. Iterator
  - Cursor API: Entwickler navigiert ein Cursorobjekt über den XML InputStream
    - Metainformationen müssen über XMLStreamReader-Instanz erfragt werden
  - Iterator API: XML Stream wird als Menge diskreter Parsingereignisse zugestellt
    - Metainfos als nicht veränderbare Objekte im Event gekapselt

Leistung in Java SE 6

- Java SE 6 enthält sehr viele Leistungsoptimierungen
  - ~ 10% Leistungsverbesserung für viele Anwendungen realistisch
- Biased Locking (SunLabs, Backport nach 1.5.0_06)
  - Biasing eines Objektes zu einem Thread. Neusperren durch eben diesen Thread wahrscheinlich. Fastpath is dann wesentlich schneller.
  - Beinflussbar per -XX:+UseBiasedLocking (Default: On)
- Lock Coarsening
  - Vermeidung unnötiger Sperren über Escape Analyse
- java.math.BigDecimal Optimierung
- Parallel Old Generation Garbage Collector
  - -XX:+UseParallelOldGC
- Unterstützung großer Seiten nun auch für Windows/Linux
  - -XX:+UseLargePages - Standard bei Solaris

Hauptspeicheroptimierung

- Optimierung des arraycopy (5.0_02) in Assembler
  - Viele Anwendungen kopieren sehr vieleDaten
- Prefetch für Objektallokierung auf x86 und x64 (5.0_06)
  - Hauptspeicher ist langsam (300-600 Zyklen!)
  - Objekte werden sequentiell im Hauptspeicher allokiert
  - Prefetch (L1/L2 cache enthält bereits Speicher für neue Objekte)
  - Prefetch plattformabhängigen Anzahl von Cachezeilen jenseits eines neuen Objekts
  - Opteron: 32 Bit VM stellt 6 64-Byte Cachezeilen zur Verfügung; x64 VM hat 4

Garbage Kollektoren

- Parallel-old-generation Kollektor
  - -XX:+ParallelOldGC
  - Optimiert für große Prozesse. Hilft SPECjbb2005 mit vielen Lagern
  - 3-5x fache Verkürzung der Pausenzeiten
  - Wichtig: -XX:ParallelGCThreads=<n> nicht größer als Prozessoranzahl !

Java 6: OOB Leistung
Java 6 b58 vs Java 5 FCS

		32 bit					64 bit			
		Original options	No options				No options			
		x86	Sparc	x86			Sparc	AMD64		
		Windows	Solaris	Windows	Linux	Solaris	Solaris	Windows	Linux	Solaris
SERVER	SpecJBB2000	13 %	31 %	N/A	11 %	14 %	22 %	18 %	14 %	15 %
	SpecJBB2005	15 %	7 %	N/A	SNF	15 %	SNF	1.27 %	31 %	SNF
	JVM 98	15 %	27 %	N/A	17 %	18 %	19 %	15 %	13 %	17 %
	Scimark	12 %	4.05 %	N/A	11 %	13 %	4.75 %	4.35 %	4.38 %	1.83 %
	jetstream	72 %	109 %	N/A	70 %	70 %	93 %	38 %	32 %	45 %
	volano	0.1 %	-13 %	N/A	4.78 %	1.06 %	-13 %	-8 %	9 %	1.19 %
	volano25	0.48 %	-17 %	N/A	SNF	8 %	-20 %	-3.53 %	14 %	SNF
CLIENT	SwingMark		-8 %	11 %	-35 %					
	SwingMark_Native		1.52 %	66 %	-35 %					
	JVM 98		21 %	33 %	36 %					
	jetstream		95 %	170 %	152 %					
	startup3		0.41 %	3.04 %	13 %					
	footprint3_real		2.29 %	-4.02 %	12 %					
	footprint3_perceived		46 %	N/A	24 %					

- Most “red” cells are due to current regressions that have been analyzed and are in process of being addressed.

Datenbanken

- Derby DB Bestandteil des JDK!
- JDBC 4.0
  Vereinfachung der Treiber Registrierung:
  - Verwendung der Services per
    META-INF/services/java.sql.driver

```
String url =
  „jdbc:oracle:thin:@localhost:1521/db“;
DriverManager.getConnection(url,user,pwd);
```

- Diverse Anpassung an neuere ANSI SQL Standards
- Annotations für Result Manipulation (@Query, @Update)
- Support for SQL ROWIDs,SQL/XML Unterstützung

Kernbibliotheken

- java.io.File
  - getFreeSpace() - liefert unallokierten Speicher der Partition
  - getUsableSpace() - liefert unallokierten Speicher, prüft auf chreibrechte, Quotas etc.
  - getTotalSpace() - Gesamtplatz der Partition
  - chmod Features
    - setExecutable(), setReadable(), setWriteable()

Beispiel: Dateisystem

```
private void copyFile(File src, File dstdir)
  throws IOException
  {
     if (src.length() < dst.getUsableSpace()
        doCopy(src,int);

     else
         throw new
            IOException(„ File zu gross“);
}
```

Kernbibliotheken

- java.io.Console – Formatierter Input/Output auf der Konsole:
  - System.console().readPassword()

Fazit

- Java SE 6 ist die konsequente Weiterentwicklung von Java 5
  - Sukzessive Vereinfachung von Schnittstellen
  - Behutsame Integration von neuen Eigenschaften
  - Java SE 6 wandelt sich zur kompletten Entwicklungsplattform
    - HTTP Server, Java DB (Derby), Scripting, Diagnosetools
  - Performance & Stabilität als permanenter Fokus
- Participate: http://jdk.dev.java.net !

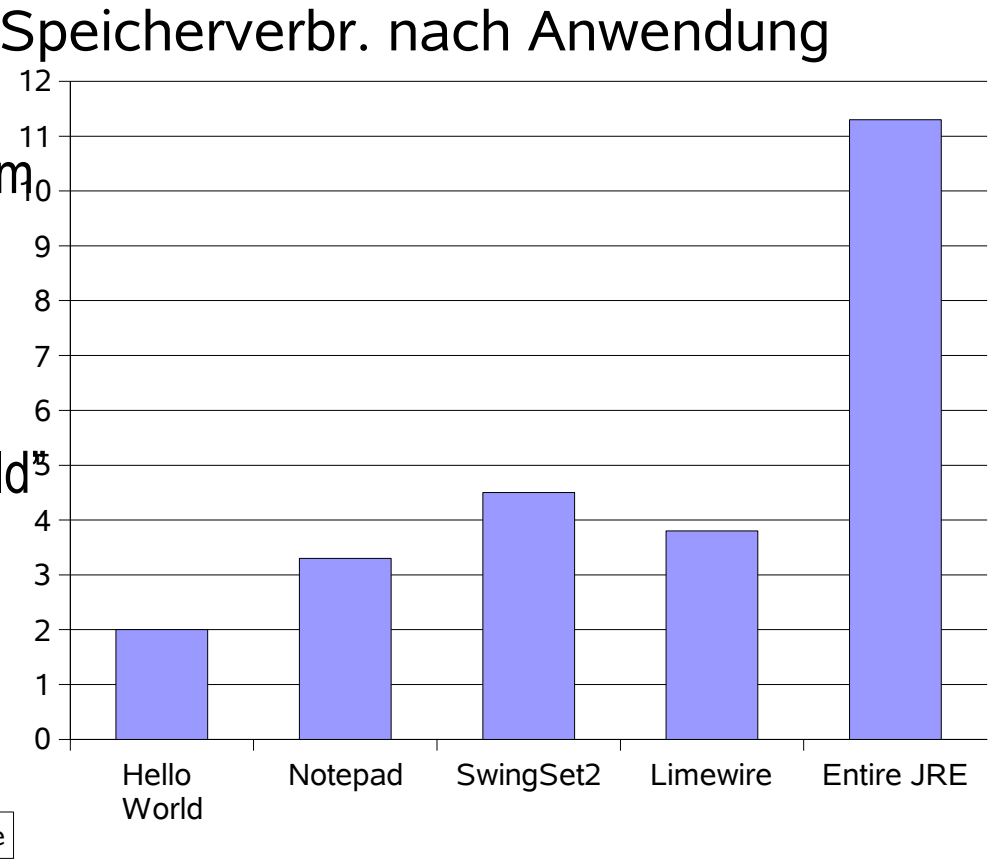
Evolution der Programmiersprache in Java 7: Kandidaten

- Superpackages
- Erweiterungen der Annotationssyntax (JSR 308)
- Abstraktionen von Kontrollkonstrukten
  - “Closures”
  - “Concise Instance Creation Expressions”
  - “First-class methods”
- Überladen von Operatoren
- Abrunden
  - Kürzere Variablendeklarationen, Zeichenketten im switch Befehl, Vergleichen von Aufzählungen

Konsumenten JRE

Ende 2007/Anfang 2008

- In Kürze
  - > Vorladen im Cache (vor Start!)
  - > Hinweis: Nicht das gleiche wir eine laufende VM
  - > Kooperation mit dem Betriebssystem
- Stark verbesserte Installation
- Ein modulares JRE
  - > Nur die Infrastruktur um “Hello World” laufen zu lassen
  - > Der Rest wird im Hintergrund nachgeladen

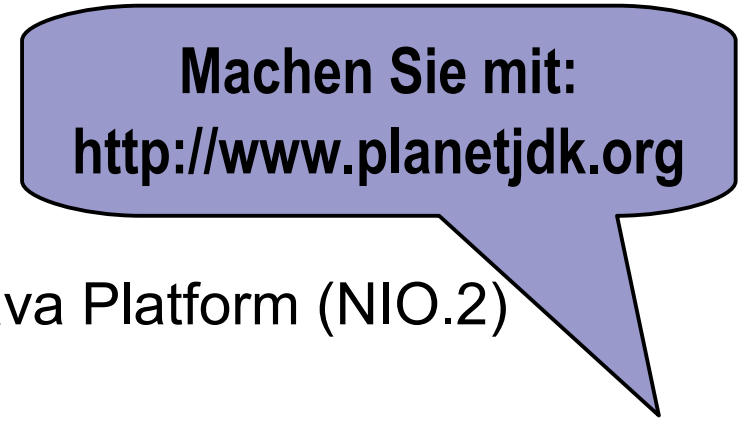


Weitere wichtige Upgrades

- JSR 255: Java Management Extensions (JMX™)
  - Specification, version 2.0
    - Namespaces, federated JMX technology servers
  - opendmk.dev.java.net/
- JSR 262: Web Services Connector for JMX Agents
  - Based on ws management standards
  - Early draft available
    - ws-jmx-connector.dev.java.net/

Vorgeschlagene JSRs für Java 7

- JSR 277: Java Module System
- JSR 294: Improved modularity support in the Java programming language
- JSR 295: Beans binding
- JSR 303: Bean validation
- JSR 296: Swing application framework
- JSR 203: More new I/O APIs for the Java Platform (NIO.2)
- JSR 220: Enterprise JavaBeans™ 3.0
- JSR 255: JMX specification, version 2.0
- JSR 262: Web services connector for JMX agents
- **JSR 260: Javadoc™ Tag Technology Update**
- *JSR(s) TBD Java Language changes*
- JSR 308: Annotations on Java Types
- **JSR 310: Date and Time API**



Java User Group Karlsruhe

jug-karlsruhe@googlegroups.com

http://groups.google.com/group/jug-karlsruhe

Ende.

Fragen und Antworten ?!

Dr. Stefan Schneider

stefan.schneider@sun.com

blog: http://blogs.sun.com/partnertech

Mitauthoren der Päsentation:

Daniel Adelhard, Sun Microsystems

Matthias Schmidt, Sun Microsystems

JUG Karlsruhe: Wie geht es weiter ?

- 15.8.2007 20:00 Informatikgebäude UKA
- Geplante Themen:
  - 30 min. Springüberblick (Oliver Schlicht; Firma Synyx)
  - 30 min. Java GC Überblick & Demo (Stefan Schneider, Sun)
- Kontakt
  - Nikola Veber
  - jug-karlsruhe@googlegroups.com
  - http://groups.google.com/group/jug-karlsruhe

StAX Cursor Beispiel

```
XMLInputFactory pf = XMLInputFactory.newInstance();
XMLStreamReader reader =
   pf.createXMLStreamReader(new FileInputStream(...));

   while (reader.hasNext()){
      int evt = reader.next();
      switch (evt){
         case XMLStreamConstants.START_ELEMENT:
         //z.B. Abfrage des Elements per
                  reader.getLocalName
            break;
         case XMLStreamConstants.CHARACTERS:
            System.out.println("Element:" + reader.getText());
            break;
      }
   }
```

Java 6 Leistung

1.6.0b89 performance relative to 1.5.0b64 - Mozilla Firefox

Datei Bearbeiten Ansicht Gehe Lesezeichen Extras Hilfe

Summary (statistically significant regressions highlighted)

1.6.0 b89 **performance relative to** 1.5.0 b64

		32 bit					64 bit			
		Original options	No options				No options			
		x86	Sparc	x86			Sparc	AMD64		
		Windows	Solaris	Windows	Linux	Solaris	Solaris	Windows	Linux	Solaris
SERVER	SpecJBB2000	16 %	33 %	N/A	11 %	12 %	26 %	22 %	12 %	16 %
	SpecJBB2005	31 %	78 %	N/A	26 %	29 %	B&S NF	33 %	45 %	42 %
	JVM 98	16 %	31 %	N/A	19 %	22 %	24 %	17 %	15 %	18 %
	Scimark	12 %	7 %	N/A	11 %	12 %	4.37 %	6 %	12 %	6 %
	jetstream	57 %	86 %	N/A	61 %	57 %	81 %	34 %	32 %	30 %
	volano	-1.29 %	2.1 %	N/A	4.73 %	-0.3 %	1.6 %	-0.53 %	11 %	8 %
	volano25	2.2 %	-3.3 %	N/A	7 %	8 %	-3.52 %	4.11 %	19 %	26 %
	maribor	8 %	25 %	N/A	20 %	19 %	24 %	24 %	23 %	17 %
CLIENT	SwingMark		-12 %	11 %	-34 %					
	SwingMark_Native		-1.4 %	67 %	-34 %					
	JVM 98		20 %	39 %	44 %					
	jetstream		65 %	172 %	155 %					
	startup3		-1.67 %	2.61 %	13 %					
	footprint3_real		13 %	-3.79 %	12 %					
	footprint3_perceived		50 %	N/A	24 %					
	start_application_swing		-20 %	-2.57 %	2.23 %					
	start_application_awt		-12 %	-4.92 %	10 %					
	start_applet_awt		-22 %	-31 %	14 %					
	start_jws		-70 %	-74 %	-35 %					
	start_jpi		-16 %	-46 %	-0.42 %					

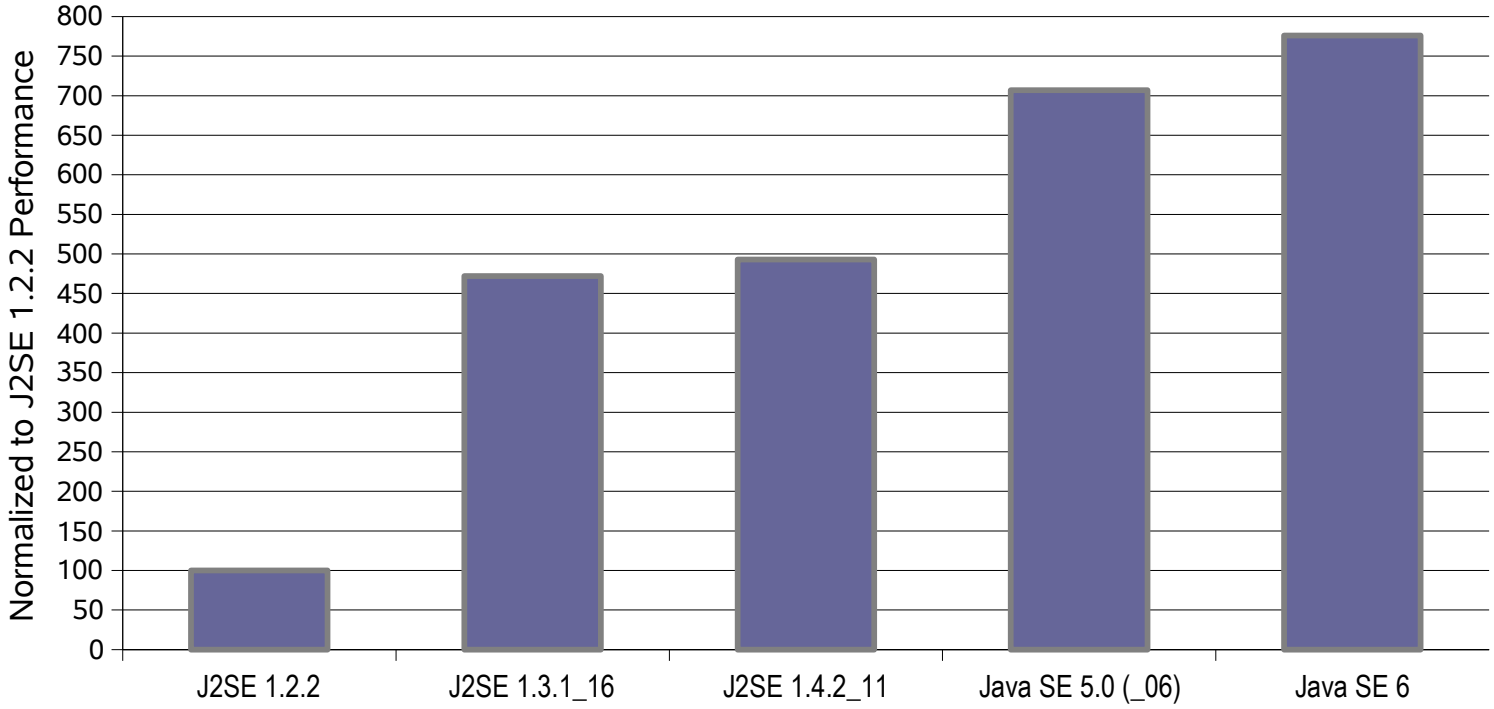
JAXB/JAX-WS Arbeitsteilung

JAX-WS Web Services in JEE 5

- Mit `@SOAPBinding` können Encoding Style und Formatting Style (RPC/Document bzw. Literal/Encoded) definiert werden
  - Default: Document/Literal
- `@OneWay` für void Operationen
  - Aufrufender Thread kann sofort zurückkehren
- `@WebParam` für Mapping auf WS Message Part
- Weitere für Handler, Result Mapping etc.

Java SE Performance History

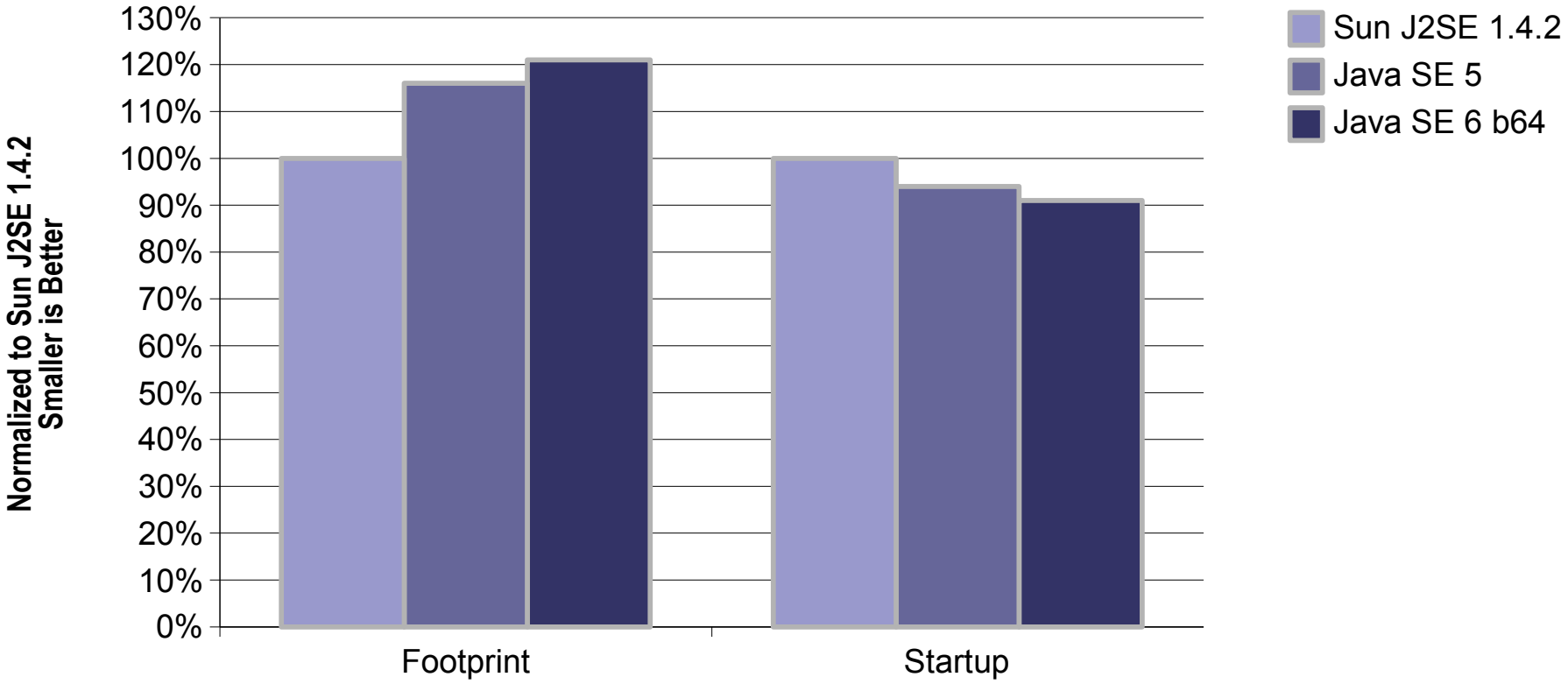
VolanoMark™



VolanoMark™ 2.5 is a benchmark from Volano LLC (http://www.volano.com)
using Sun's 32-bit JVM on Solaris 9, SunFire v490 4x1.2 GHz, 32 GB

Startup and Footprint

Windows, out-of-box (no command line arguments)



Client Benchmarks
Graphics and I/O (Windows)