

Virtual Machines unter der Haube



Alexander Lais <alexander.lais@gmx.de>
unfug.org
21.06.2007

Agenda

- Aufbau der Laufzeitumgebungen
- Binärformate und Befehlssatz
- Compilierung
- Garbage Collection
- Nativer Code

Agenda

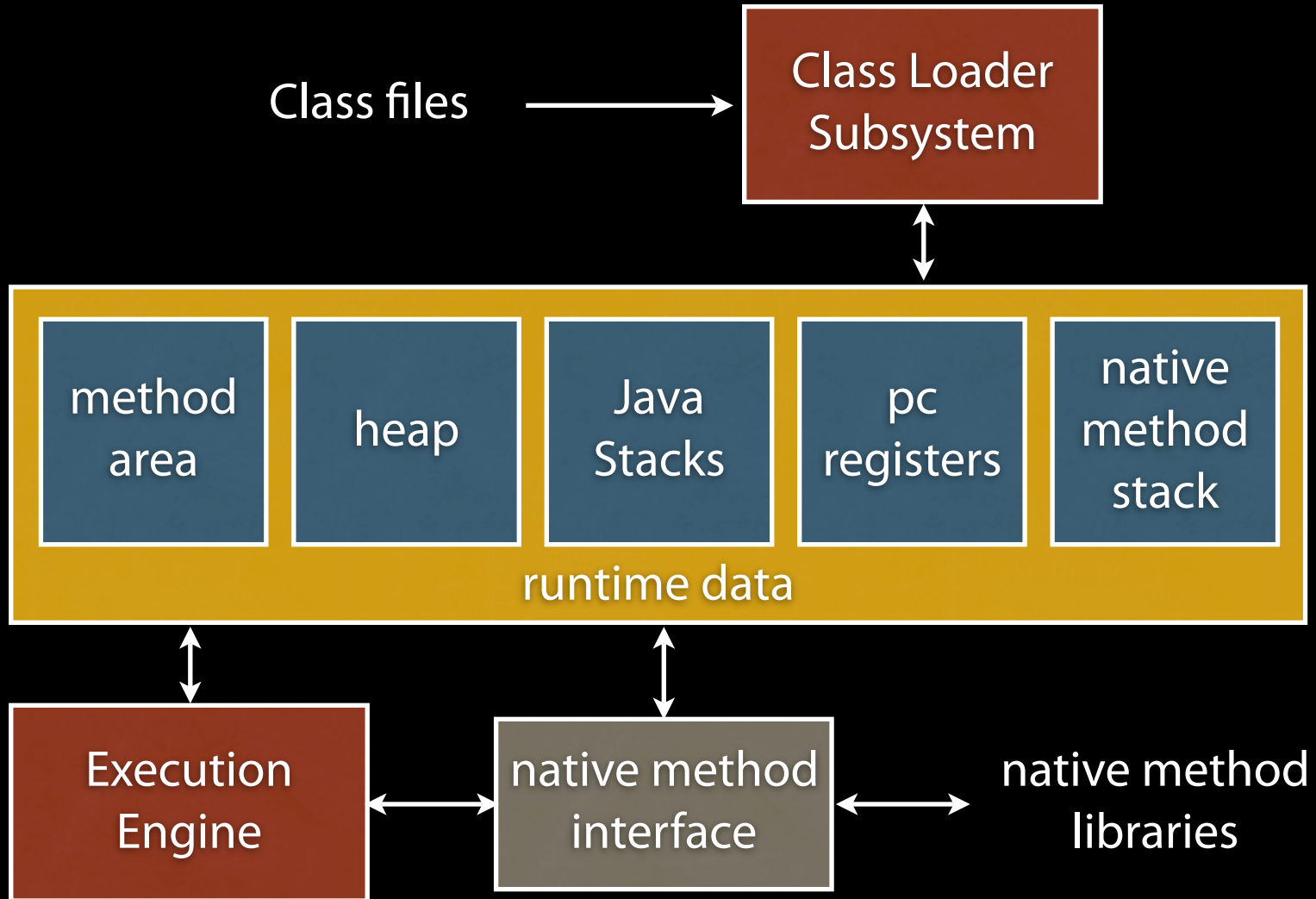
- ▶ **Aufbau der Laufzeitumgebungen**
 - Binärformate und Befehlssatz
 - Compilierung
 - Garbage Collection
 - Nativer Code

Aufbau der VMs

- Abstrahierte definierte Laufzeitumgebung
- Stack-Machines
- Plattformunabhängiger Bytecode
- Automatische Speicherverwaltung
- Anbindung an nativen Code möglich



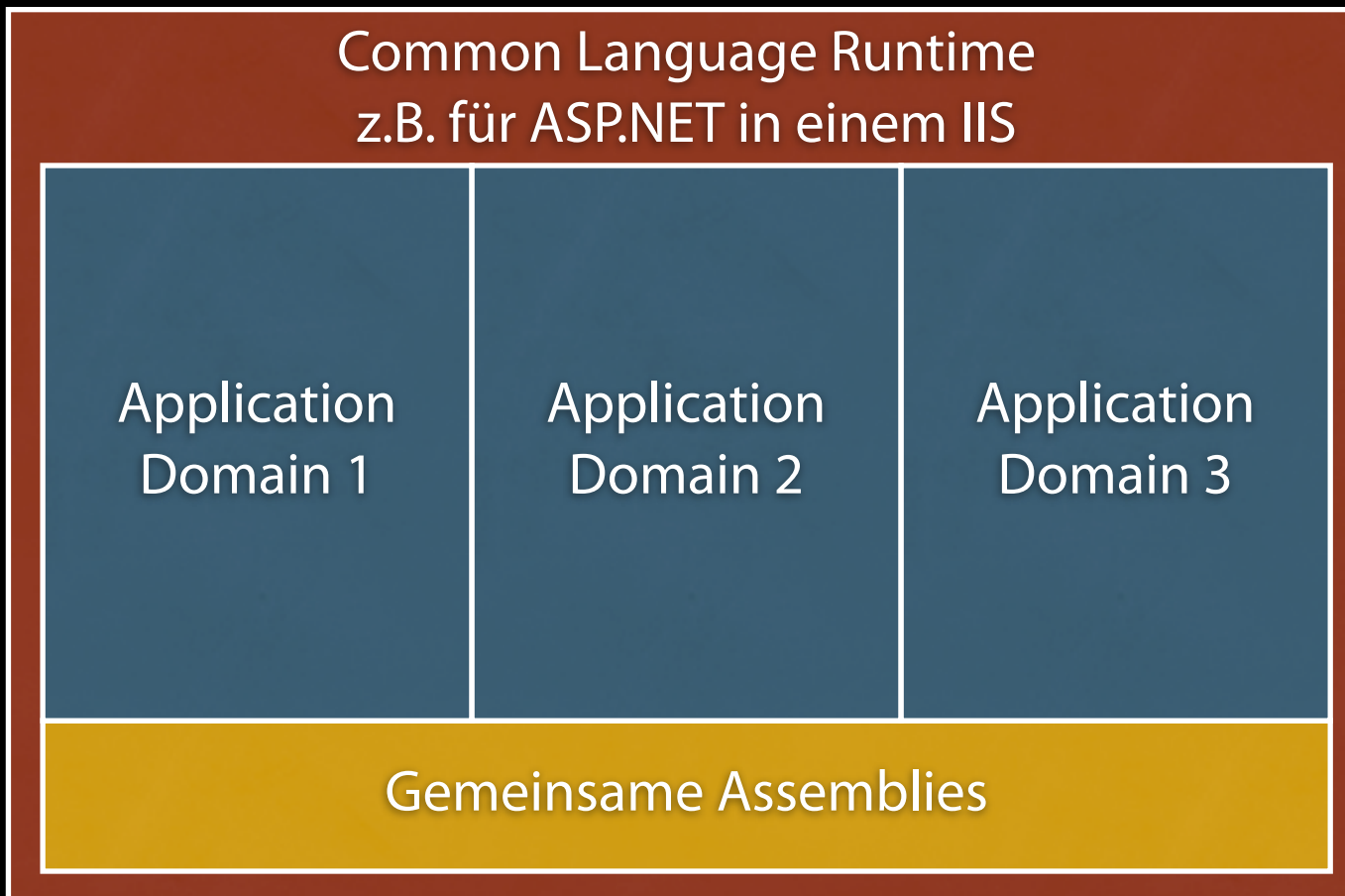
Java HotSpot™ VM





.NET CLR

Im Wesentlichen ähnlich wie Java
Besonderheit: Application Domains



Agenda

- Aufbau der Laufzeitumgebungen
- ▶ **Binärformate und Befehlssatz**
- Compilierung
- Garbage Collection
- Nativer Code



Befehlssatz

- Fein granular. Viel Arbeit für den Compiler

```
iconst_1 // Push int constant 1.
istore_1 // Pop into local variable 1, which is a: byte a = 1;
iconst_1 // Push int constant 1 again.
istore_2 // Pop into local variable 2, which is b: byte b = 1;
iload_1  // Push a (a is already stored as an int in local variable 1).
iload_2  // Push b (b is already stored as an int in local variable 2).
iadd     // Perform addition. Top of stack is now (a + b), an int.
int2byte // Convert int result to byte (result still occupies 32 bits).
istore_3 // Pop into local variable 3, which is byte c: byte c = (byte) (a + b);
iload_3  // Push the value of c so it can be returned.
ireturn  // Proudly return the result of the addition: return c;
```

- Verschiedene Befehle für Datentypen
- i = int, f = float, a = object reference, ...



Befehlssatz

- Arbeit liegt bei der Implementierung
- Bytecode nicht auf Datentypen fixiert

```
// *****  
// i = i + 1  
// *****  
ldloc.1           // load variable 1 onto stack  
ldc.i4.1          // load constant onto stack  
add               // add  
stloc.1           // store to variable 1
```

- Programmgesteuerte Erstellung von MSIL
System.Reflection.Emit Namespace



Binärformate

- Class-Files
 - Binärcode der Klassen, Methoden, etc.
 - Class beginnt mit Magic Number `0xCAFEBAFE`
- Diverse Classloader möglich
 - Dateien, Streams, JARs, URLs, ...
 - „Dynamic Class Loading“ möglich



Binärformate

- Portable Executable (PE)
 - Nennt sich „Assembly“
 - *.exe, *.dll, ...
- Bootstrapping-Code
 - bindet mscorcore.dll und damit die .NET CLR
 - Aufruf von `_CorExeMain()`
- MSIL-Code

Agenda

- Aufbau der Laufzeitumgebungen
- Binärformate und Befehlssatz
- ▶ **Compilierung**
- Garbage Collection
- Nativer Code

Compilierung

- HotSpot-Ansatz
 - erstmal interpretieren. Wenns „heiß“ wird, kompilieren
- CLR Just-in-Time
 - Wenn die Methode aufgerufen wird
- Ahead-of-Time
 - In aller Ruhe vorher



HotSpot Compiler

- „Client“-JIT
 - schnell kompilieren, wenig optimieren
 - Optimiert auf Interaktion
- „Server“-JIT
 - etwas langsamer, besser optimiert
 - Spart Zeit unter Last



CLR JIT

- Kompilieren, wenn eine Klasse/Methode benutzt wird
- 64-bit Version optimiert ausgiebiger
 - x86_64 bietet deutlich mehr Register
- Vorteil von JIT allgemein:
Bestmögliche Optimierung auf die Plattform



CLR Ahead of Time

- NGEN
 - Cachen vorcompilierter Assemblies
- Seit .NET 2.0:
 - .NET Runtime Optimization Service v2.0...
 - Automatisch vorcompilierte Basis-Libs



Java Ahead of Time

- Gerne für Real Time Java eingesetzt
- IBM WebSphere Real Time Compiler
 - AOT-Kompilat als jar-Datei
- GCJ - GNU Compiler for Java
 - jRate für Real Time Java
- Excelsior JET

Agenda

- Aufbau der Laufzeitumgebungen
- Binärformate und Befehlssatz
- Compilierung
- ▶ **Garbage Collection**
- Nativer Code

Garbage Collection

- Verschieden Ansätze:
„Stop the World“, Parallel, Concurrent
- Organisation:
Generationen. Die Meisten sterben jung...
- Einflußnahme:
Manuell starten, Weak References



Garbage Collectors

- Serial Collector
- Parallel Collector
- Compacting Parallel Collector
- Concurrent Mark-Sweep
- Manuelles Rufen über `System.gc()`;



Garbage Collection

- Teile des Heaps werden zusammengefasst
- Manuelles Rufen über
`System.GC.Collect([int Generation]);`
- Aufräumen nativer Ressourcen:
`IDisposable.Dispose()`

Agenda

- Aufbau der Laufzeitumgebungen
- Binärformate und Befehlssatz
- Compilierung
- Garbage Collection
- ▶ **Nativer Code**



Nativer Code

Java Native Interface (JNI)

- Keyword `native`
- `System.loadLibrary("libname");`
 - lädt `libname.so` oder `libname.dll`
- Alternativ: `JNI_CreateJavaVM()`
 - JVM in nativem Code spawnen



JNI Beispiel

Java

```
package pkg;
class Cls {
    native double f(int i, String s);
    ...
}
```

C++

```
extern "C" /* specify the C calling convention */
jdouble Java_pkg_Cls_f__ILjava_lang_String_2 (
    JNIEnv *env,          /* interface pointer */
    jobject obj,          /* "this" pointer */
    jint i,               /* argument #1 */
    jstring s)            /* argument #2 */
{
    const char *str = env->GetStringUTFChars(s, 0);
    ...
    env->ReleaseStringUTFChars(s, str);
    return ...
}
```



Nativer Code

.NET Interoperability

- P/Invoke, nativer Code in .NET

```
[DllImport("User32.dll")]  
static extern Boolean MessageBeep(UInt32 beepType);
```

- COM Interop, .NET als COM+ verkleidet

```
[System.Runtime.InteropServices.ComVisible(True)]  
public class ComSiehtMich {  
    ...  
}
```

- C++/CLI interop: „It just works“ (???)



Nativer Code

C++/CLI (Compiler-Flag: /clr)

- Aufgebohrte C++-Syntax
- Gemischte DLLs, native Einsprungspunkte

```
#include <string> (...)  
using (...)  
int main(void) {  
    std::string teststr("It just works!")  
    String^ netstring = gcnew String(teststr.c_str());  
    Console::WriteLine(netstring);  
}
```

- es funktioniert auch noch :-)

Ist was hängengeblieben?

Quellen

1. http://java.sun.com/docs/books/jvms/second_edition/html/VMSpecTOC.doc.html
2. http://java.sun.com/docs/books/jvms/first_edition/html/VMSpecTOC.doc.html
3. <http://java.sun.com/docs/books/jni/html/intro.html#994>
4. http://java.sun.com/docs/books/jvms/second_edition/html/ClassFile.doc.html#80959
5. <http://msdn.microsoft.com/msdnmag/issues/04/05/VisualC2005/default.aspx#S2>
6. <http://www.javaworld.com/javaworld/jw-09-1996/jw-09-bytecodes.html>
7. viele andere, die untergegangen sind...