

entwickler

Software, Systems & Development

magazin

CD-INHALT: Alle Infos auf Seite 5

www.entwickler-magazin.de

Mai/Juni

3.09



Entwickler-TV



Bonus für
Abonnenten
der Profi-CD:
Keynote der BASTA!
Spring 2009

Testversion

Advantage Database Server 9

Software

Eclipse Rich Ajax Platform,
Apache Tomcat 6.0.18,
MySQL 6.0.9-alpha,
PostgreSQL 8.3.7, Wireshark,
NetBeans for C/C++

Weitere Artikel

► SVG praktisch

Einsatz in der Geometrie

► Per JDBC auf Datenbanken zugreifen

► WebOnDisk

Webanwendungen auf dem
Desktop

► PDF-Reporting

Optimierte Reports mit der
Gnostice eDocEngine VCL



Datenträger enthält
Info- und
Lehrprogramme
gemäß § 14 JuSchG

Config im Griff

Softwarekonfiguration per Framework



Sicher tunneln:

OpenVPN mit Delphi

Lose Kopplung



So werden SOAs einfacher und flexibler entwickelt

Auswahlstrukturen in C# Grundlagen für Entwickler

SQLite für die Kleinen Integration in Embedded-Systeme

WebOnDisk: Webanwendungen auf dem Desktop

Write once, run everywhere

Webanwendungen sind modisch, dennoch möchten viele Anwender „webige“ Programme auch offline nutzen können. Wer beide Ansprüche mit einer Applikation erfüllen möchte, sitzt zwischen zwei Stühlen. Eine Umgebung muss her, die es erlaubt, eine Webanwendung auf dem Webserver sowie auf dem Desktop zu nutzen.

von George Herzog

Einst wurde die Java-Laufzeitumgebung mit dem Spruch „write once, run everywhere“ (sinngemäß: schreib die Anwendung einmal und starte sie auf jedem Betriebssystem) beworben. In Zeiten, in denen die meisten Rechner ständig mit dem Internet verbunden sind, muss die Bedeutung dieser Forderung erweitert werden: Die einmalig und möglichst kostengünstig erstellten Anwendungen müssen sowohl als Internetapplikation wie auch als lokale Anwendung laufen können. Wer heutzutage nun etwas auf sich hält, stellt Onlineanwendungen bereit, deren Gestaltungsmöglichkeiten denen von Desktopsoftware hoch überlegen sind. Das Aussehen solcher Webseiten kann mittels HTML und CSS-Styles leicht geändert, bzw. Dynamik mittels JavaScript eingebaut werden. Die Erweiterbarkeit der Software ist gewährleistet, da auf Standards aufgebaut wird, die jeder Browser versteht. Mit geringem Aufwand kann so eine dyna-

mische Webseite erstellt werden. Aber wie werden solche dynamischen Webseiten als Desktopanwendung bereitgestellt? Nichts ist naheliegender, als mit der Anwendung einen auf freien Standards basierenden Webserver mitzuliefern.

Eine der häufig gewählten Laufzeitumgebungen für Webanwendungen im professionellen Umfeld ist Java, so werden wir uns in diesem Artikel auf Java-basierende Servlet-Container und Applikationen konzentrieren. Unser Ansatz ermöglicht es, die Laufzeitumgebung (Java in der benötigten Version) und Server „out of the box“ mit der Webanwendung mitzuliefern und so von der Umgebung möglichst unabhängig autonom starten zu können – dieser Ansatz wird WebOnDisk-Technologie genannt.

Die Herausforderung ...

Der Benutzer kennt die Details und Anforderungen nicht, die unsere Webanwendung benötigt. Es ist nicht zumutbar und auch fehleranfällig, zu verlangen,

dass der Benutzer die Laufzeitumgebung und einen Servlet-Container einrichtet und konfiguriert, Ports einstellt usw. (besonders unter Windows sind Benutzer es nicht gewöhnt, Konfigurationsdateien zu editieren). Eine „webige“ Anwendung ist gefordert, die out of the box funktioniert und sich selbst konfiguriert. Natürlich muss die Anwendung auch ohne Administratorrechte und ohne Installation (bzw. von Read-Only-Medien gestartet) erwartungsgemäß arbeiten! Die Aufgabenstellung kristallisiert sich in folgenden Anforderungen heraus:

- Der Webanwendung soll die Java-Laufzeitumgebung in der benötigten Version zur Verfügung gestellt werden, unabhängig davon, ob Java auf dem Clientrechner installiert ist. Die installierte Java-Version soll verwendet werden, soweit verfügbar, ansonsten nutzt die Webapp eine mitgelieferte JRE.
- Ein Servlet-Container soll gestartet werden, der JSP-Servlets, Struts- bzw.

JSF-Anwendungen ausführen kann. Ein verfügbarer Port auf *localhost* soll ausgesucht werden, unter dem der Server die Webanwendung zur Verfügung stellt.

- Der Browser soll automatisch mit dem zur Laufzeit ermittelten URL und Port der eingerichteten dynamischen Webseite, die sofort nutzbar ist, gestartet werden.
- Nach Abschluss der Benutzersessions soll der Webserver heruntergefahren und die belegten Ressourcen sollen wieder freigegeben werden.
- Eine erfolgreiche Anwendung wird auf vielen Rechnern mit unterschiedlicher Konfiguration eingesetzt. Dies verlangt nach einer möglichst fehlertoleranten und self-contained Anwendung. Ein hohes Hotline-Aufkommen (über 0,5 Prozent der Benutzer) verursacht nämlich bei 50 000 Kunden schon 250 Reklamationen! Kein Unternehmen kann sich heute viele verärgerte Kunden leisten!

... Lösungsmöglichkeiten ...

Die Anforderung, Java-(Web-)Anwendungen auf dem Desktop möglichst unkompliziert zu starten, stellt sich nicht zum ersten Mal. Es existieren bereits Möglichkeiten, die in Betracht gezogen werden können:

- **RAP:** Die Rich Ajax Platform (ein Eclipse-Plug-in [1]) ist mit dem Anspruch angetreten, die Entwicklung von Webanwendungen auf dem Desktop zu unterstützen. Die von der Firma Inno-Pract entwickelte Architektur ist Open Source und steht somit jedem frei zur Verfügung, der Webanwendungen auf dem Desktop starten möchte. Die Idee des Frameworks, mit einer Codebasis Web- und Desktopanwendungen zu kompilieren, ist hervorragend. Leider gibt es noch ein Paar Hürden, die es in unserem Beispiel verhindern, die Technologie einzusetzen:
- Es gibt Funktionen, die nicht überall und transparent zur Verfügung stehen. So sind Mausereignisse in Webumgebungen nicht durchgängig verfügbar.
- Die Benutzerführung kann nicht immer konsistent umgesetzt werden. Beispiel:

einen online und offline gleich funktionierenden Datei-Öffnen-Dialog gibt es nicht.

- Das Session-Handling kann nicht offline und online gleich gehandhabt werden.
- Eine gemeinsame Codebasis für beide Anwendungstypen ist nicht in jedem Fall möglich, da bestimmte Funktionen auf unterschiedlichen Plattformen verschiedene Interfaces implementieren müssen.
- Eine einfache Konfiguration der Oberfläche ist nur mit großem Aufwand möglich, da die Anwendung nicht auf HTML-Seiten basiert.
- **Java Web Start:** Mithilfe der Java-Web-Start-Technologie ist es möglich, über das Internet kleine Anwendungen zur Verfügung zu stellen, die bei Bedarf heruntergeladen und gestartet werden. Die gestartete Anwendung benötigt keinen Browser. Die Technologie ist ein von Sun definierter Standard, der (prinzipiell) betriebssystemunabhängig zur Verfügung steht. Allerdings gibt es Ausschlusskriterien, die die Software nicht erfüllt, sodass auch diese Lösung nicht in Betracht gezogen werden kann:

- Die Technologie basiert nicht auf HTML/CSS/JSP/JSF, ist somit nicht flexibel anpassbar.
- Eine bereits installierte Java Runtime wird vorausgesetzt.
- Eignet sich nicht für größere Datenbestände, da ein Download von mehreren 100 MB Daten viel Zeit in Anspruch nimmt. Eine Auslieferung über CD-ROM ist nicht vorgesehen.

Alle bekannten Optionen, die plattformunabhängige Lösungen anbieten, weisen Nachteile auf, die die oben beschriebenen Anforderungen nicht erfüllen. Zum Glück gibt es im Java-Umfeld genug Open-Source-Projekte, die es erlauben, eine eigene Implementierung, basierend auf freier Software, selbst voranzutreiben. Es steht nichts im Wege, einen leichtgewichtigen Servlet-Container, wie Tomcat [2] oder Jetty [3], in unsere Anwendung einzubinden, die Java-Laufzeitumgebung mitzuliefern und mit entsprechender Konfiguration zu starten.

...und die Lösung

Die Anwendung – genannt WebOnDisk – schweißt die verschiedenen Technologien zusammen, die zum Starten der dynamischen Desktop-Webapp benötigt werden. Da das Programm die (Java-) Runtime Environment und die verfügbaren Webserver, bzw. Browser zur Laufzeit ermitteln

Listing 1

Beispiel zum Ermitteln zweier freier Ports

```
void CleanupSocket(int hSocket)
{
    if (hSocket)
        closesocket(hSocket);
    SAcleanup();
}

BOOL IsPortFree(USHORT ushPort, const char* sIP)
{
    WSADATA wsaData;
    WORD version = MAKEWORD(2, 2);
    WSASStartup(version, (LPWSADATA)&wsaData);
    SOCKETADDR* _pmySockAddress;
    sockaddr_in service;
    service.sin_family = AF_INET;
    service.sin_addr.s_addr = inet_addr(sIP);
    service.sin_port = htons(ushPort);
    _pmySockAddress = (SOCKETADDR*) &service;

    SOCKET hSocket = socket(AF_INET, SOCK_STREAM, 0);
    int_iReturn = bind(hSocket, _pmySockAddress,
        sizeof(service));
    if (_iReturn == SOCKET_ERROR)
    {
        if (WSAGetLastError() == WSAEINVAL)
        {
            CleanupSocket(hSocket);
            return FALSE; //port in use
        }
        CleanupSocket(hSocket);
        return FALSE; //other error
    }
    CleanupSocket(hSocket);
    return TRUE;
}

USHORT GetFreePort()
{
    USHORT ushPort = 49152;
    while (ushPort <= USHRT_MAX)
    {
        if (IsPortFree(ushPort, "127.0.0.1"))
            break;
        ushPort++;
    }
    return ushPort;
}
```

muss, muss sie in einer Form vorliegen, die von der Umgebung möglichst unabhängig ist. Die Wahl fällt in einem solchen Fall meist auf eine C++-Executable, die plattformspezifisch kompiliert eine sehr schnelle Low-level-Anbindung an das Betriebssystem ermöglicht.

In einem ersten Schritt wird die verfügbare Java-Umgebung ermittelt. Ist ein installiertes Java in der benötigten Version vorhanden, so wird diese verwendet, ansonsten kann eine mitgelieferte JRE benutzt werden. Ob Java installiert ist, ist aus der Registrierung zu ermitteln, indem der Ast `HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Runtime En-`

`vironment` überprüft wird. Mithilfe der Library `jvm.dll` aus dem Unterverzeichnis `bin/client/` kann die JRE geladen werden. Als Nächstes sollen zwei freie Ports ermittelt werden, die von unserem Webserver

Mit einem Workaround lässt sich die Session-Timeout überlisten.

belegt werden können (Connector Port und Shutdown Port des Servlet-Containers). In einer Schleife werden dazu freie Ports gesucht (das Beispiel aus Listing 1 nutzt das Windows-API; für UNIX-basierte Systeme nutzt man spezifische Aufrufe).

Jetzt kann ein Servlet-Container (in unserem Beispiel Jetty) mithilfe der `procrun`-Library [4] problemlos mit Angabe der ermittelten Ports gestartet werden (Listing 2). Die `procrun`-Library kapselt Zugriffe auf das Java-Framework, mit dessen Hilfe der Webserver hochgefahren werden kann (`libprocrun` ist eine Library der Apache Software Foundation mit der Bezeichnung „Apache Commons Daemon“ unter der Apache License Version 2).

In einem weiteren Schritt muss festgestellt werden, wann die Servlet Engine ansprechbar ist. Hierzu werden in einer Schleife in definierten Zeitabständen HTTP-Requests an den ermittelten freien Connector Port abgesetzt, bis eine Antwort ermittelt werden kann. Ist eine Antwort verfügbar, kann der Standardbrowser des Benutzers mit dem URL der

Webapp als Command-Line-Parameter gestartet werden, der zu der lokal eingetragenen Webseite navigiert.

Eine neue Fragestellung taucht auf, wenn der Benutzer die Arbeiten auf unserer lokalen Anwendung beendet. Sollte der Servlet-Container heruntergefahren werden? Dagegen spricht, dass der Start des Servers beim Neustart der Anwendung länger dauert als der Start des Browsers im Fall eines bereits laufenden Webservers. Andererseits belegen der Server und die JRE Ressourcen, die nach dem Ende der Benutzersessions freigegeben werden können. Sollen die Ressourcen des Clientrechners geschont werden, wird empfohlen, den Server zu beenden, sobald alle Sessions abgelaufen sind. Um die Anzahl der offenen Sessions zu ermitteln, kann die Serveranwendung im Fall von Tomcat die Manager-Applikation periodisch befragen (unter dem URL `manager/list`). Unter Jetty steht die Manager-Applikation nicht zur Verfügung, hier muss die Webapp unter einem URL die Anzahl der offenen Sessions bereitstellen.

Warum reicht es nicht, die gestartete Browserinstanz bis zum Beenden zu überwachen? Der Grund liegt im verschiedenen Verhalten der Browser: Manche öffnen beim Aufruf eine neue Anwendungsinstanz (IE 6.x), manche verwenden eine bereits gestartete und öffnen dort einen neuen Tab (Firefox) oder ein neues Browserfenster. Natürlich kann es vorkommen, dass der Anwender seine Session ablaufen lässt und dann einen Link in der Webseite klickt. Der Klick läuft ins Leere, da mit der Session auch der Webserver heruntergefahren wird. Mit einem Workaround lässt sich die Session-Timeout überlisten: Ein in die angezeigte Webseite eingebettetes, unsichtbares IFRAME setzt periodisch (mittels Meta-Tag `refresh` oder einer JavaScript-Funktion) Requests ab, die die Session am Leben erhalten. Statt Aufruf des Standardbrowsers ist es auch denkbar, noch ein Stück autonomer vorzugehen und sogar den benötigten, bzw. empfohlenen Browser mitzuliefern! Die portablen Versionen von Firefox und Opera arbeiten auch ohne Installation fehlerfrei, sogar eine auf Adobe AIR [5] basierende Lösung kann umgesetzt

Listing 2

Unser Servlet-Container

```

BOOL bUseTomcat = FALSE;
CString sJvmDll(_T("sDirOfJRE+" "\\bin\\client\\
                                     jvm.dll"));

APXHANDLE hPool = apxPoolCreate(NULL, 0);
_towchar _swJvmDll(sJvmDll);
APXHANDLE hWorker = apxCreateJava(hPool, _
                                     swJvmDll);

if (IS_INVALID_HANDLE(hWorker))
    return FALSE;

CString sStartupJar = bUseTomcat ? "bin\\bootstrap.jar"
                                : "start.jar";
CString sStartupClass = bUseTomcat ? "org/apache/
    catalina/startup/Bootstrap" :
    "org/mortbay/start/Main";
LPSTR _jni_jvmoptions = "-XX:+CMSPermGenSweeping
Enabled";

//eine temporär geschriebene Konfigurationsdatei
//des Servers
//mit Angabe der freien Ports
LPSTR _jni_rparam = "c:\\temp\\etc\\jetty.xml";

apxJavaInitialize(hWorker, sStartupJar, _jni_
    jvmoptions,
    iJavaXms, iJavaXmx, dwThreadStackSize);

apxJavaLoadMainClass(hWorker, sStartupClass, NULL,
    _jni_rparam);

apxJavaSetOut(hWorker, TRUE, _swSCErrFilePathName);
apxJavaSetOut(hWorker, FALSE, _
    swSCOutFilePathName);

//Run worker
return apxJavaStart(hWorker);

```

Listing 3

Beispiel für eine XML-Steuerdatei für Adobe AIR

```

<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://ns.adobe.com/air/
application/1.0">
  <id>myweba pp</id>
  <version>0.1</version>
  <filename>mywebapp</filename>
  <initialWindow>
    <content>mywebapp.htm</content>
    <visible>true</visible>
  </initialWindow>
</application>

```

werden. Für den Einsatz von Adobe AIR spricht, dass der eingebettete, auf Webkit basierende Browser schlank, schnell und ohne Toolbar/Buttonbar verwendbar ist. Die Binaries von Adobe AIR funktionieren auch ohne Installation erwartungsgemäß und sind über das Adobe AIR SDK verfügbar. Die WebOnDisk-Anwendung kann Adobe AIR mit einer XML-Steuerdatei lenken, der Speicherort der temporär erstellten XML-Datei wird dabei als Command-Line-Parameter an AIR übergeben. Die einfachste Form einer solchen Steuerdatei sehen Sie z. B. in Listing 3.

Das angegebene, einfache Beispiel steuert Adobe AIR zur Anzeige der HTML-Datei *mywebapp.htm* (weitere Einstellungen, wie die Größe und Position des zu öffnenden Fensters, bzw. anzuzeigendes Icon sind möglich). Die Angabe eines URL in der XML-Steuerdatei wird von AIR nicht unterstützt. Deshalb muss diese die temporär erstellte HTML-Datei *mywebapp.htm* mittels JavaScript auf den URL der unter *localhost* geladenen Webapp weiterleiten.

Grenzen der Technologie

Die gezeigte Lösung erlaubt es, auch

Procrun: Der Apache commons daemon

Die Library *procrun* (*libprocrun*) der Apache Software Foundation ist eine Sammlung von APIs und Anwendungen, die den Zugriff auf das Java-API aus C++-Anwendungen heraus erleichtern; Projektdateien für verschiedene Versionen von Microsoft Visual Studio sind verfügbar. Die Software steht unter der Apache Software License, Version 2.0. Mithilfe dieser Library wird unter anderem der Servlet-Container Tomcat für Windows (*tomcat.exe*, *tomcatw.exe*) erstellt. Sie erlaubt es unter anderem, eine JRE zu laden, Java-Anwendungen über deren *public static void main(args[])*-Funktion zu starten und beim Herunterfahren der Runtime-Umgebung benutzerdefinierte Aktionen auszuführen. Die Installation der Anwendung als Service wird unterstützt und kann mittels Startparameter gesteuert werden.

Die Library unterstützt auch den Zugriff aus Java-Anwendungen; so ist es möglich, aus Java den Native Daemon zu steuern.

komplexe Webanwendungen auf einem Desktoprechner auszuführen, indem diese mithilfe einer JRE in einen leichtgewichtigen Servlet-Container eingebunden werden. Dabei ist diese Technologie trotzdem einem dedizierten Server nicht gleichzusetzen. Folgendes ist beim Einsatz der WebOnDisk-Lösung zu bedenken:

- Ein Webserver muss für seine Webanwendungen mehr bieten als nur eine Runtime Environment. Datenbankverbindungen und andere zu installierende Softwarekomponenten bzw. Frameworks sind heute Standard bei komplexen Enterprise-Webanwendungen.
- Die mitgelieferten Softwarekomponenten, die ein autonomes Starten ohne Voraussetzungen an den Clientrechner erlauben, benötigen erheblichen Speicherplatz. Die JRE schlägt mit bis zu 70-80 MB zu Buche, der Servlet-Container Jetty ist wiederum mit ca. 6 MB relativ schlank.
- Die entwickelte Lösung kann von jedem beliebigen Laufwerk gestartet werden, auch von CD/DVD, bzw. UNC-Pfad. Allerdings sind die Startzeiten des Servers von einem langsamen Medium entsprechend zeitraubend. Die Startzeit kann mit Anzeigen eines Splash-Screens überbrückt werden.
- Die Implementierung der Lösung für Nicht-Windows-Systeme ist möglich. Statt Verwendung von MFC (Microsoft Foundation Class Library) muss man unter Linux bzw. MAC OS auf die betriebssystemspezifischen Libraries zurückgreifen. Für UNIX-basierte Systeme stellt die Apache Software Foundation statt *libprocrun* das *jsvc*-Framework [6] zur Verfügung.
- Die WebOnDisk-Lösung startet auch unter aktuellen Wine-Implementierungen [7] schnell.

Erfahrungen

Die WebOnDisk-Technologie wird bereits im professionellen Umfeld in Verbindung mit Enterprise-Webanwendungen, basierend auf JSF-Technik, erfolgreich eingesetzt. Die Auftraggeber profitieren von den einmalig anfallenden Entwicklungskosten und bieten die Anwendung

für die Nutzer über das Internet/Intranet und auf CD-ROM, bzw. USB-Stick an. Die Robustheit der verwendeten Open-Source-Frameworks und Libraries wird durch das geringe Hotline-Aufkommen bestätigt. Ein weiterer Vorteil ist das einfache und kostengünstige Customizing der Anwendung durch Verwendung des JSF-Frameworks, von HTML sowie CSS, ohne dass die Codebasis der Anwendung angepasst werden muss.

Fazit

Die Anforderung, die vor ein paar Jahren noch als Vision galt – nämlich Webanwendungen auf dem Desktop anzubieten – ist heute mit vergleichsweise wenig Aufwand umzusetzen. Freie Standards und Open-Source-Software machen es möglich, das Web ins Wohnzimmer zu holen. Die kostengünstige und flexible WebOnDisk-Technologie überzeugt die Hersteller; die modische Oberfläche bietet dem webgewöhnten Nutzer die ihm vertraute Umgebung. Und durch das Mitbringen aller benötigten Softwarekomponenten wird die größtmögliche Unabhängigkeit vom Clientsystem und dessen Einstellungen erreicht; die automatische Ermittlung aller Konfigurationsparameter vermeidet manuelle Konfigurationsfehler.



George Herczeg, Softwarearchitekt bei der SHI Elektronische Medien GmbH in Augsburg. Mit mehr als 10 Jahren Erfahrung als C++, Java-, Perl-, ASP- und PHP-Entwickler unterstützt er die Integration von Softwarekomponenten in unterschiedlichen Sprachen. Kontakt: george.herczeg@shi-gmbh.com.

Links & Literatur

- [1] RAP: <http://www.eclipse.org/rap/>
- [2] <http://tomcat.apache.org/>
- [3] <http://www.mortbay.org/jetty/>
- [4] *libprocrun*, Apache Software Foundation: <http://commons.apache.org/daemon/procrun.html>
- [5] <http://www.adobe.com/products/air/>
- [6] *jsvc* Library, Apache Software Foundation: <http://commons.apache.org/daemon/jsvc.html>
- [7] <http://www.winehq.org/>