Das Zusammenspiel von Components und Container innerhalb der J2EE wissen

J2EE Components

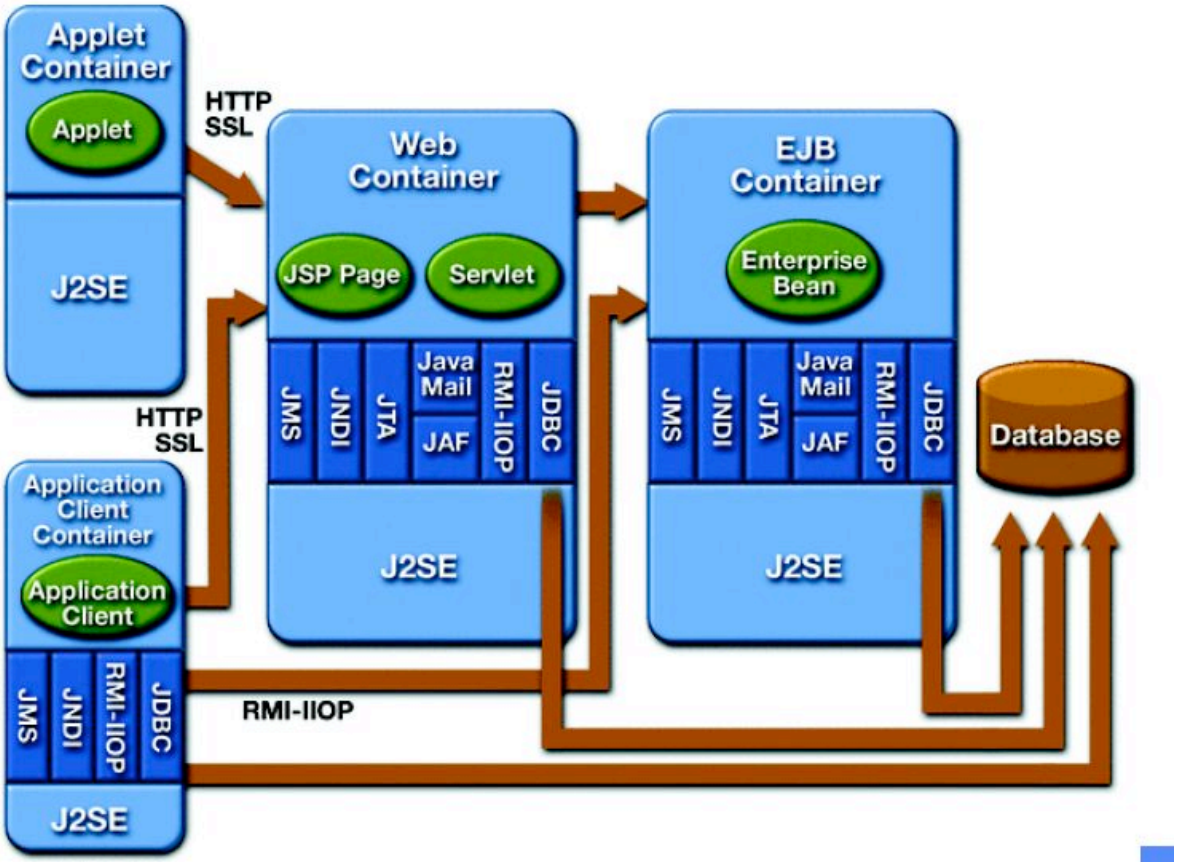
J2EE applications are made up of components. A J2EE component is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and that communicates with other components. The J2EE specification defines the following J2EE components:

- Application clients and applets are components that run on the client.
- Java Servlet and JavaServer Pages technology components are Web components that run on the server.
- Enterprise JavaBeans components (enterprise beans) are business components that run on the server.

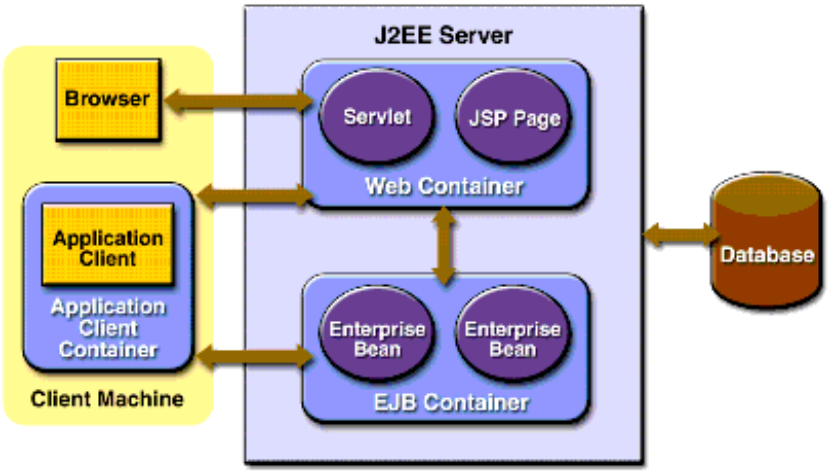
J2EE components are written in the Java programming language and are compiled in the same way as any program in the language. The difference between J2EE components and "standard" Java classes is that J2EE components are assembled into a J2EE application, verified to be well formed and in compliance with the J2EE specification, and deployed to production, where they are run and managed by the J2EE server.

J2EE Containers

Normally, thin-client multitiered applications are hard to write because they involve many lines of intricate code to handle transaction and state management, multithreading, resource pooling, and other complex low-level details. The component-based and platform-independent J2EE architecture makes J2EE applications easy to write because business logic is organized into reusable components. In addition, the J2EE server provides underlying services in the form of a container for every component type. Because you do not have to develop these services yourself, you are free to concentrate on solving the business problem at hand.



Container Types



J2EE Server and Containers

J2EE server

The runtime portion of a J2EE product. A J2EE server provides EJB and Web containers.

Enterprise JavaBeans (EJB) container

Manages the execution of enterprise beans for J2EE applications. Enterprise beans and their container run on the J2EE server.

Web container

Manages the execution of JSP page and servlet components for J2EE applications. Web components and their container run on the J2EE server.

Application client container

Manages the execution of application client components. Application clients and their container run on the client.

Applet container

Manages the execution of applets. Consists of a Web browser and Java Plug-in running on the client together.

Die einzelnen Components und Container namentlich kennen

Components

- Web Components
  - Servlets, JSPs, Filters, Event Listeners
- Enterprise JavaBeans
  - Session (stateful/less), Message Driven, Entity
- Resource Manager Drivers ("Connectors")
- Application Clients
- Applets

Containers
- Web Container
- EJB Container
- Application Client Container
- Applet Container

Die Services welche ein Container leistet beschreiben können

- Life cycle management for components
- Distribution (Corba IIOP)
- Resource management
  - Multi-threading, state management, resource pooling
- Transaction Handling
  - Container managed (EJBs) vs. component managed
- Security management
  - Authentication, authorization, secure communication
  - Declarative vs. programmatic

J2EE-kompatible Applikationsserver stellen Web-Container und EJB-Container bereit. Im Web-Container werden Servlets und Java Server Pages ausgeführt. Der EJB-Container ist die Laufzeitumgebung für serverseitige EJB-Komponenten.

Die web components beschreiben und ihre Grundfunktionalität und Verwendung wissen können

Web Components

- Servlets
  - Sophisticated web server extension
  - Consume HTTP request, produce HTTP response
- JSPs
  - Compiled to servlets
  - Inside-out servlets
  - For producing text-based content (HTML, XML, ...)
  - Taglibs instead of code (?)

Die Inhalte von J2EE Applikationen (.ear Files) beschreiben können

EAR (Enterprise Archive) Archiv-Datei zur Speicherung einer Unternehmensanwendung bestehend aus Web-Ressourcen (Servlets, JSPs, HTML usw.) und Enterprise JavaBeans (EJBs) sowie XML-Deployment-Deskriptoren (siehe **J2EE**).

WAR (Web Archive) Archiv-Datei zur Speicherung einer Web-Applikation (Servlets, JSPs), verschiedener Web-Ressourcen und der XML-Deployment-Deskriptoren (siehe **J2EE**).

Das Deployment von J2EE Applikationen kennen und erklären

Running Web Container, Deploy servlet into the Web Container, use browser to activate the servlet.

Die Rolle und Inhalte des Deployment Descriptors kennen

WJ-3141: Design and Development of Simple Java(TM) Servlet Applications

The deployment descriptor is an Extensible Markup Language (XML) file that specifies the configuration and deployment information about a specific Web application.

Question: In the deployment descriptor, how many Web applications may be defined?

Answer: One. There is one deployment descriptor for every Web application.

Question: In the deployment descriptor, how many servlets may be defined for a single Web application?

Answer: As many as you need.

When a Web application is deployed to the Web container, the directory structure must follow a particular format:

- The static HTML files are stored in the top-level directory of the Web application.
- The servlet and related Java technology class files must be stored in the WEB-INF/classes directory.
- The auxiliary library JAR files must be stored in the WEB-INF/lib directory.
- The deployment descriptor must be stored in the file called web.xml in the WEB-INF directory.

The mapping between the development environment and the deployment environment is a straightforward process consisting of the following steps:

- The static HTML files are copied from the web directory to the top-level Web application directory.
- The servlet and Java technology source files are compiled into the WEB-INF/classes directory.
- The auxiliary library JAR files are copied into the WEB-INF/lib directory.
- The deployment descriptor is copied into the WEB-INF directory.

There is another deployment strategy that is specified in the servlet specification. A Web Archive (WAR) is a JAR file that includes the complete Web application deployment structure in a single archive file. The file structure in the WAR file must include all static Web pages, the WEB-INF directory, the web.xml file, all library JAR files, and all Web component class files. You can create a WAR file by using the jar tool to create the file archive of the Web application's deployment structure.
On the Tomcat server, you can deploy a WAR file in the webapps directory.

Die wichtigsten JSP Direktiven kennen und beschreiben können

http://www.computerbase.de/lexikon/JSP

Direktiven

Eine Direktive dient zum Übermitteln von speziellen Seiteninformationen an den JSP-Compiler; dadurch kann man angeben, ob die JSP eine Taglib einbindet oder wie im Fehlerfall weiter zu verfahren ist.
Die allgemeine Syntax für eine Direktive ist <%@ ... %>. Folgende Direktiven sind vorhanden:

-1 include:

- Diese Direktive weist den JSP-Compiler an, den vollständigen Inhalt einer externen Datei in die Originaldatei zu kopieren.
  - <%@ include file="BeispielDatei.ext" %>

-2 page

- import, ein Java-Import-Statement wird in der Datei generiert
- contentType, gibt die Art des Datei-Inhaltes an. Sollte dann eingesetzt werden, wenn man kein HTML benutzt oder den Default-Zeichensatz nicht verwendet
- errorPage, gibt die Seite an, die im Fehlerfall angezeigt werden soll
- isErrorPage, gibt an ob diese Seite eine Error-Page ist oder nicht, wenn ja ist das exception Objekt verfügbar
- isThreadSafe, gibt an ob das aus der JSP generierte Servlet threadsicher ist oder nicht
  - <%@ page import="java.util.*" %> //import
  - <%@ page contentType="text/html" %> //contentType
  - <%@ page isErrorPage=false %> //die Seite ist keine Error-Page
  - <%@ page isThreadSave=true %> //eine threadsichere JSP

-3 tagLib

- diese Direktive gibt an, dass eine Taglib verwendet werden soll. Es muss ein Prefix und eine URI für die Taglib vergeben werden.
  - <%@ taglib prefix="MeinPrefix" uri="taglib/MeineTagLib.tld" %>

Den Zusammenhang zwischen JSPs und Servlets kennen

JSPs beinhalten Javacode und HTML Code, hingegen Servlets bestehen lediglich aus Java Code. JSP Seite können in einem FORM Block Verweise auf Servlets beinhalten die Parameter entgegennimmt und diese bearbeitet und ein Ergebnis zurückliefert.

Die möglichen Inhalte von Webapplikationen kennen

In WebArchiv kann alles enthalten sein, Servlets, JSP, ... sonst. Files, MP3

Das Model-View-Controller Design Pattern erklären können

Komponenten:

• **Model** (z.B. EJB) enthält Datenstrukturen

• **View** (JSP) enthält nur Layout. Ideal: Kein oder möglichst wenig Scriptlet-Code, sondern nur Actions, Directives und Custom Tags

• **Controller** Servlet steuert ScreenFlow, und gibt kein HTML-Code aus, Servlet hat externe Konfiguratin des ScreenFlows

• **ScreenFlow** am besten extern konfiguriert (vgl. PetStore)

Grundfunktionalität von JDBC sowie die dafür benötigten Softwarekomponenten

JDBC – Java Database Connectivity Grundfunktionalität:

dient in Java zum Zugriff auf relationale Datenbanksysteme. Die Installation von JDBC Treibern für das jeweilige DBMS ist einfach – man muss sie nur vom DBMS Hersteller organisieren

Zugriff auf eine MS Access Datenbank

- ... über die JDBC ODBC Bridge
- Die JDBC ODBC Bridge wird mit J2SE mitgeliefert – Open DataBase Connectivity Treiber ist der Datenbanktreiber von Microsoft, mit dem man z.B. auf ein Access File zugreifen kann

- Datenbanken müssen für den ODBC Zugriff registriert werden

Zugriff auf eine MySQL Datenbank Verbindung:

- Erfordert MySQL Connector/J (http://www.mysql.com/downloads/api-jdbc-stable.html)
- MySQL Connector/J unterstützt Java-2 JVMs, JDK- 1.2.x, JDK-1.3.x und JDK-1.4.x aber nicht JDK-1.1.x oder JDK-1.0.x
  - Um den Connector zu verwenden muss der CLASSPATH auf mysql-connector-java-3.0.7-stable-bin.jar gesetzt werden

Weitere Schritte:
Erstellen einer Datenbank, Eintrag von Daten, Erstellen einer Verbindung des Programmes zu der Datenbank, Schicken entsprechender SQL Statements zu der Datenbank, Verarbeitung der Ergebnisse von der Datenbank… (siehe Frage 17),18))

Die einzelnen Schritte bei der Verwendung von JDBC in einem Programm erklären können

Erstellen einer Verbindung

- Laden des Treibers im Java Sourcecode:
  Für **JDBC_ODBC Bridge**: Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
  Für **MySQL**: Class.forName("com.mysql.jdbc.Driver")

Bei anderern Treibern muss der Treibername in der Dokumentation nachgelesen werden…. ☐ Class.forName(" Treibername ");

Erstellen einer Verbindung

- URL einer Datenbank:
  Für JDBC-ODBC Bridge (nur lokal)
  jdbc:odbc:Datenquellenname
  Für MySQL:
  jdbc:mysql://[host] [,failoverhost...][:port]/[database][?propertyName1] [=propertyValue1] [&propertyName2][=propertyValue2]...

- Ist der Hostname nicht definiert- default IP Adresse '127.0.0.1`
- Ist das Port nicht definiert – default MySQL Server Port '3306`
- Ist die Datenbank nicht definiert - default derzeit aktuell geöffnete Datenb.
- Aufbau der Verbindung zur Datenbank im Programm: Connection con = DriverManager.getConnection(url, "myLogin", "myPassword");

Javacode für MySQL-Verbindung

```
import java.sql.*;
public class mysqlTest{
public static void main(String[] args) {
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con =
DriverManager.getConnection(jdbc:mysql://192.168.1.7:3306/Person",
"User","");
// ... Datenbankbearbeitung der Datenbank Person vom Benutzer User
}
catch (SQLException e) {....}
catch (ClassNotFoundException e) {....}
}}
```

Die wichtigsten SQL-DML Statements zur Abfrage und Manipulation von Datenbankinhalten kennen

SQL Statements

Bsp.:

- **Einfügen** eines Datensatzes in eine Tabelle

– Statement stmt = con.createStatement();
stmt.executeUpdate("INSERT INTO TABELLENNAME " + "VALUES (wert,wert,wert)");

Die Methoden von Statement

- executeUpdate: wird verwendet für
  INSERT, UPDATE, oder DELETE von Datensätzen
  SQL DDL (Data Definition Language) Befehle wie CREATE TABLE, DROP TABLE, and ALTER TABLE
  gibt einen integer Wert zurück (Anzahl der betroffenen Zeilen oder 0)
- executeQuery: wird verwendet um bestimmte Datensätze aus der Datenbank zu lesen (SELECT) und gibt diese in einem ResultSet zurück
- Execute: wird verwendet um ein CallableStatement auszuführen
- executeQuery und executeUpdate schließen des ResultSet des Statement Objects, wenn es offen ist

Wichtig sind auch JOINS:

- Dienen zur Erfragung von Daten aus verschiedenen Tabellen –z.B.
  - Welchen Gehaltsunterschied gibt es zwischen jüngeren und älteren Mitarbeitern?

BSP.:

```
String query= "SELECT Stammdaten.Geburtsdatum, Mitarbeiter.Gehalt " + "FROM
Stammdaten, Mitarbeiter " + "WHERE Mitarbeiter.Vorname LIKE Stammdaten.Vorname "
+ "and Mitarbeiter.Nachname LIKE Stammdaten.Nachname";
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(query);
```

Die unterschiedlichen Statements aus dem JDBC API kennen und ihre Verwendung erklären können

->Siehe auch Statements aus Frage 18)

PreparedStatements

- Ist abgeleitet von der Statement Klasse
- Reduzieren bei mehrmaligen Gebrauch die Executionszeit, da sie nur einmal vom DBMS compiliert werden

PreparedStatements

- PreparedStatements werden (wie Statements)über eine Connection kreiert
- und mit Werten mit Hilfe der setXXX Methoden von PreparedStatement (z.B. set***Int*** oder set***String***) versorgt
- vor allem in Schleifen nützlich – z.B.:

```
PreparedStatement prepStmt = con.prepareStatement ( "UPDATE Mitarbeiter SET
Gehalt = ? WHERE Abteilung LIKE ? ");
int [] divGehaelter = {75, 1500, 600};
String [] arbeitsKlassen = {"Hausfrau", "Schifahrer", "MTA"};
for(int i = 0; i < divGehaelter.length; i++)
{
   prepStmt.setInt(1, divGehaelter[i]);
   prepStmt.setString(2, arbeitsKlassen[i]);
}
```

Die Anwendungsmöglichkeiten von Transaktionen in JDBC

Werden benötigt, wenn man eine Änderung nur dann möchte, wenn auch eine (oder mehrere) andere Änderungen erfolgreich durchgeführt werden Eine Transaction ist ein Set von Befehlen, die als Einheit durchgeführt werden – entweder alle oder kein Befehl wird durchgeführt

Jede Verbindung wird automatisch im *auto-commit* Mode geöffnet – d.h., dass jeder einzelne SQL Befehl als Transaction gehandhabt wird.
Um mehrere SQL Befehle in einer Transaction auszuführen muss man daher den *auto-commit* Mode ausschalten.

Die wichtigsten Unterschiede und Erweiterungen seit JDBC 2.0 (inkl.)

- **Scrollable Resultsets**
- **Änderungen von Daten**

Scrollable Resultset

- Ermöglicht die folgenden Bewegungen des Cursors im ResultSet:
  - vorwärts: rs.next(),
  - rückwärts: rs.previous(),
  - zu einem bestimmten Datensatz:
- rs.absolute(Zeilennummer)
- rs.relative(Zeilen)

Kreieren von Scrollable Resultset

```
Statement stmt = con.createStatement
        ( ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("...");
```

- **1.Argument von createStatement:**

TYPE_FORWARD_ONLY - Cursor bewegt sich im Resultset nur vorwärts (default)
–
TYPE_SCROLL_INSENSITIVE – Cursor kann sich im Resultset auch rückwärts bewegen
–
TYPE_SCROLL_SENSITIVE – Cursor kann sich im Resultset auch rückwärts bewegen (Änderungen, der Datenbank von anderen Benutzern werden im ResultSet auch geändert)

- **2.Argument von createStatement:**

– CONCUR_READ_ONLY - Resultat ist nur lesbar (default)
– CONCUR_UPDATABLE (Resultat kann verändert werden)

- **Da beide Argument int-Werte sind – nicht vertauschen!!!**

Bedingungen für das Updaten Scrollable Resultset

- die Abfrage darf sich nur auf einen Table beziehen (Keine Joins!!)
- alle Schlüssel des Tables müssen im SELECT Statement angeführt sein
- => das ergibt Probleme bei einer MySQL Datenbank ohne Schlüssel – Lösung:

ALTER TABLE Tabellennamen ADD PRIMARY KEY (key1, key2 ...)

Die Unterschiede zwischen einem GenericServlet und einem HTTPServlet

HttpServlet ist von GenericServlet abgeleitet

Die wichtigsten Methoden der beiden Servlet-Typen und ihr Zweck

Die init() Methode

- Wird wie bei Applets beim ersten Laden aufgerufen und dient zum Laden einmaliger zeitintensiver Informationen
  - z.B. Laden von Konfigurationsdaten von einen File
  - oder Starten von unterstützenden Threads

- Muss beendet werden, bevor irgendeine andere Methode aufgerufen werden kann
- Bekommt als Parameter eine Instanz der Klasse *ServletConfig*
  - mit Hilfe derer man Initialisierungsparameter aus dem File *servlet.properties* (im Directory *jsdk2.1\webpages \web-inf* \) entnehmen kann oder
  - eine Instanz der Klasse ServletContext

Die *service()* Methode

- Ist die zentrale Methode von Servlets
- Liest *request* von Client und produziert *response*

Die *destroy()* Methode

- Ermöglicht einem Servlet Aktionen auszuführen, bevor es heruntergefahren wird
- z.B.: das Speichern von Daten (=> Persistenz!)

Die Verwendung von Cookies bzw. Alternativen

Dienen zum Speichern von Informationen einer oder mehrer Sessions
- Anwendung im E-Businessbereich, wo *Daten* von Kunden gespeichert werden

Eigenschaften von Cookies

- Besteht aus einem Stringpaar (Namen/Wert)
- Webbrowser können meist 20 Cookies (jedes mit ca. 4 KB) pro Host speichern – insgesamt aber max. 300

Lebensdauer von COOKIES

- Ein Cookie existiert solange der Webbrowser des Benutzers läuft –
  - man kann aber auch explicit ein längere Lebensdauer (in Sekunden)
  - oder das Löschen eines Cookies veranlassen
  - als Default wird eine negative Zahl angegeben, die veranlasst, dass das Cookie solange existiert wie der Browser geöffnet ist

Achtung – bedingt durch das Verhalten verschiedener Webbrowser ist nicht immer Verlass auf die Cookies (können auch explicit vom Benutzer ausgeschalten werden)

COOKIE Unterstützung vom JSDK

- Cookie Informationen werden im http Response Header mitgeschickt
- Diese Informationen werden durch Agents (meist im Webbrowser) gespeichert
- Das Servlet API erhält alle Cookie durch den Request Header und muß sich das gewünschte Cookie suchen

Alternative zu Cookies

- URL rewriting
- „hidden Variablen„ von Formularen

Die wichtigsten Module der JSTL und ihre Funktionsweise

Die Java Server Pages Standard Tag Library JSTL kapselt mit Hilfe einfacher Tags Kernfunktionalitäten ein, die bei JSP Applikationen Gang und Gebe sind. Dieser Standard ermöglicht es, Tags in verschiedenen Applikationen immer wieder zu verwenden, um beispielsweise den Code im JSP zu verkürzen oder den Programmieraufwand zu reduzieren. Es ist nämlich viel einfacher diese vorgefertigten Funktionalitäten mit einem einfachen Tag zu implementieren als sich alles immer und immer wieder selbst neu zu programmieren. Grundsätzlich unterscheidet man zwischen JSTL und Custom Tag Libraries.

Die wichtigsten Module sind:

- Core
  - Variable Support
  - Flow Control
  - URL Management
  - Miscellaneous
- XML
  - Core
  - Flow Control
  - Transformation
- Internationalization
  - Locale
  - Message Formatting
  - Number and Date Formatting
- Database
  - SQL
- Function
  - Collection Length
  - String Manipulation

Codebeispiel SQL Query:

```
<c:set var="bid" value="${param.Add}"/>
<sql:query var="books" >
select * from PUBLIC.books where id = ?
<sql:param value="${bid}" />
</sql:query>
```

Die Funktionsweise eines TagHandlers erklären können

Tag Handler werden vom Web Container gestartet um ein Tag auszuführen, das in einer JSP Seite referenziert wird.
Folgende Funktionen des Tag Handleres werden angesprochen:

- doStartTag() zu Beginn eines Aufrufs – retouniert SKIP_BODY (DefaultTag body wird ignoriert) oder EVAL_BODY_INCLUDE
- doEndTag() am Ende eines Aufrufs – retouniert EVAL_PAGE wenn der Rest der Seite noch evaluiert werden muss ansonsten SKIP_PAGE
- die Tag Handler Klasse kommuniziert mit dem .jsp über die Variable *pageContext* (*javax.servlet.jsp.PageContext*)

Die Verwendung und Vorteile/Nachteile von JavaMail erklären können

Die JavaMail API beinhaltet ein plattform- und protokollunabhängiges Rahmenwerk um Mail und Messaging Applikationen programmieren zu können. Die JavaMail API ist als Java Platform optional Package implementiert und auch als Teil der J2EE erhältlich.
Beispiel:

In einem Buchungssystem, das eine Java Applikation ist, soll nach einer Buchung immer eine Bestätigungsmail gesendet werden. Dies ist mit JavaMail realisierbar.

Vorteile
- Unabhängig von Platform und Protokoll
- Abstrakte Schicht über Standards wie SMTP, POP3, IMAP
- Offen für neue Protokolle
- High-Level Interface
- Die Details der low-level Protokolle müssen nicht berücksichtigt werden

Nachteile
??? Auf seinen Folien wurden keine erwähnt. Im Internet nicht wirklich etwas dazu gefunden. Gibt es keine? Eventuell ihn fragen.

Die wichtigsten Internet Emailprotokolle kennen

- Senden von e-mail messages
  - Simple Mail Transfer Protocol (SMTP)
  - Multipurpose Internet Mail Extensions (MIME)

- Empfang von e-mail messages
  - Post Office Protocol 3 (POP3)
  - Internet Message Access Protocol (IMAP)

Die wichtigsten Klassen des JavaMail APIs benennen können

Class javax.mail.Session
- Einstiegspunkt, enthält Eigenschaften und Konfigurationen.

Class javax.mail.Message
- Repräsentiert eine email.

Class javax.mail.Store
- Repräsentiert die „Mailbox" mit Mails und Ordnern.

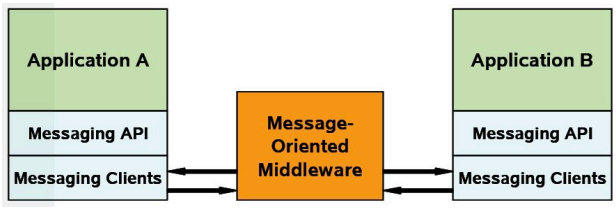
Class javax.mail.Transport
- Repräsentiert das Transportprotokoll.

Die Verwendung von JMS

JMS – Java Message Service: Messaging zur Kommunikation zwischen verschiedenen Applikationen oder Komponenten

- Netzwerk- & protokollunabhängig
- Synchrone & asynchrone Übertragung
- 2 Messaging Domains (Peer-to-Peer &Publish / Subscribe)
- Store-and-Forward Verfahren
  - Speicherung der Messages in der Middleware bei Nicht-Verfügbarkeit des Empfängers (asynchrone Übertragung)
  - Middleware garantiert Auslieferung der Messages
- Unterstützung von Transaktionen

→ Leichte Integration von bereits bestehenden Applikationen

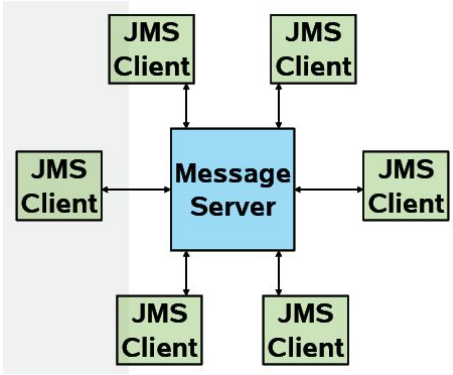


Unterschiede JMS und Java Mail

nichts konkretes in Unterlagen, nachfragen was erwünscht ist!

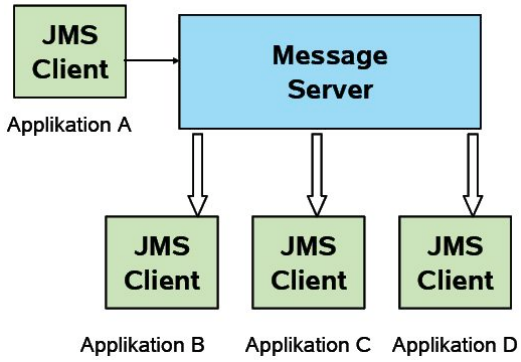
Unterschied Peer-to-Peer und Publish-Subscribe

P2P:



- Ein Empfänger pro Message
- Jeder Absender kennt den Empfänger
- Einsatz vorzugsweise, wenn Empfänger den Erhalt der Message bestätigen muss

PS:



- Viele Empfänger pro Message
- Keine Kenntnis der Applikationen untereinander

Komponenten einer JMS Architektur

- JMS Clients – Java Programme, die Messages senden und empfangen
- Non-JMS Clients – Clients, die die ursprüngliche API des Message Systems nutzen.
- Messages – Jede Applikation definiert eigene Messages die zur Kommunikation zwischen den Clients verwendet werden.
- JMS Provider – Ein Messaging System, das neben einer JMS Implementierung auch Werkzeuge für die Administration und Kontrolle des Systems zur Verfügung stellt.
- Administered Objects – Vorkonfigurierte JMS Objekte, die durch einen Administrator erzeugt wurden, und den Clients zur Verfügung stehen. Hierbei gibt es zwei Typen von JMS administered Objects:

- Connection Factory – Ein Objekt, das der Client zur Verbindung mit dem Messaging System benutzt.
- Destination – Dieses Objekt verwendet der Client um seine Messages zu Adressieren (Zielobjekte).

Softwarekomponenten zur Implementierung von JavaMail und JMS

JMS Server
Mail Server