Zusammenfassung ENAPP - HS2018

Alex Neher

January 12, 2019

Inhalt

Abbildungsverzeichnis

1 JavaEE

JavaEE, oder Jave Enterprise Edition, ist eine Alternative zur bereits bekannten JavaSE, oder Java Standard Edition.
Im Gegensatz zu JavaSE, wo Applikationen als Standalone Programme z.B. via .jar Files gestartet werden, werden JavaEE Applikationen auf Applikationsserver deployed, wo sie dann in bestimmten Containern laufen und stets zur Verfügung stehen.

Mit 'Container' meint man spezielle Laufzeitumgebungen auf diesen Applikationsserver, die der JVM bestimmte Funktionen zur Verfügung stellt, wie z.B. ein Datenbankzugriff. Es wird zwischen vier Hauptarten von Container unterschieden:

- Applet Container
- Application Client-Container
- Web-Container / Servlet-Container
- EJB-Container

Diese Container-Arten beinhalten meist die 'passenden' Anwendungs-Komponenten:

Applet: GUI-Anwendungen im Browser (Eher veraltet heutzutage)

Application Client: Standalone GUI-Applikationen auf dem Client (z.B. als SWING-Applikation)

JSP / Servlet: Web-Komponenten

EJB: Beinhaltet Geschäftslogik einer Umgebung

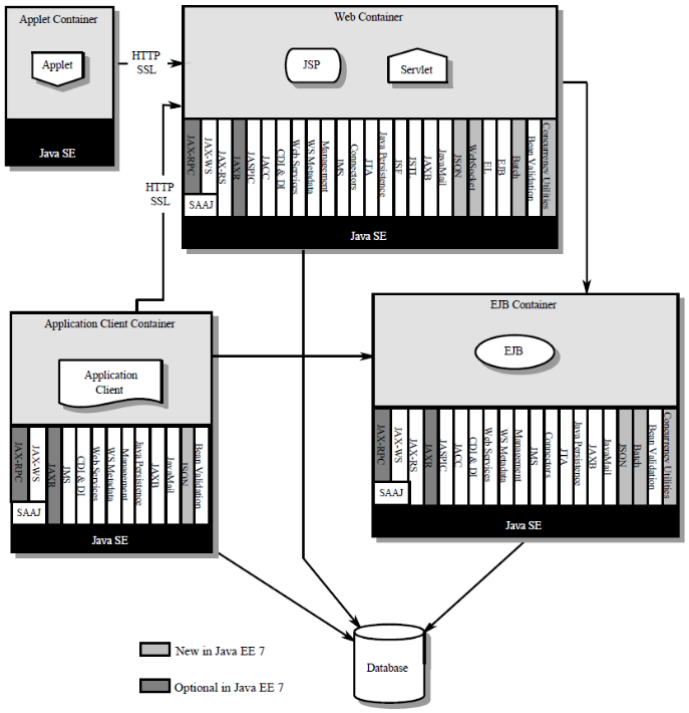

Abb. 1.1: Verschiedene Container- und Anwendungs-Typen

Applikationsserver können entwerder als Full-Profile oder aber als Web-Profile installiert werden. Das Full-Profile enthält alle Container- und Anwendungs-Typen, während das Web-Profil zwar leichter ist, aber auch weniger Funktionen bereitstellt, sondern nur diese, die bei einfacheren Infrastrukturen benötigt werden (so ist z.B. JMS nicht inbegriffen im Web-Profile, da es normalerweise erst bei komplexeren Infrastrukturen benötigt wird)

2 High Availability (JGroups)

3 Web-Tier

3.1 Tiers und Architekturen

Wie bereits in anderen Modulen besprochen wurden, kann eine Applikation in verschiedene *Tiers* unterteilt werden. Normalerweise hat man 2- oder 3-Tier Applikationen, aber theoretisch könnte man so viele Tiers haben wie man will.

Bei 3-Tier Architekturen unterscheidet man zwischen Front-End, Middleware und Back-End, oder *Web Tier, App Tier, DB-Tier*

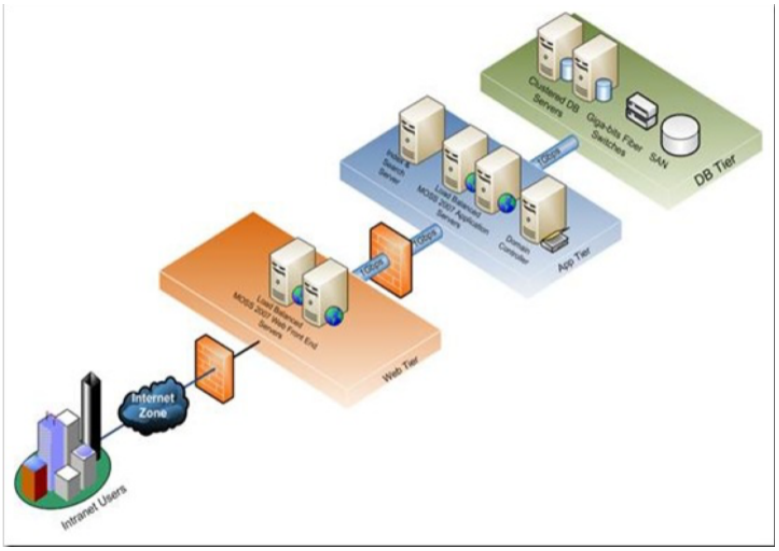

Abb. 3.1: Typische 3-Tier Architektur

Alternativ kann auch eine *Web-zentrische Architektur* verwendet werden, bei welcher der Web- und App-Tier konsolidiert werden:

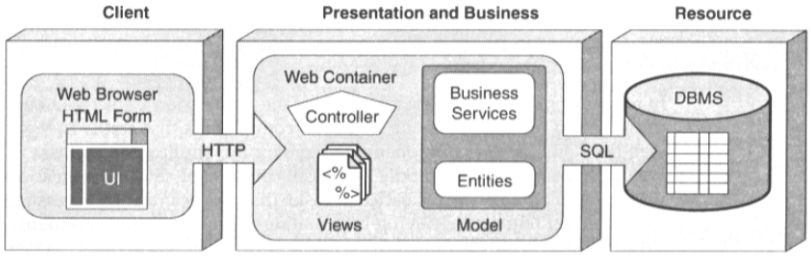

Abb. 3.2: Das Frontend und die Middleware werden zusammengefasst zu einem einzigen Tier

Wiederum eine weitere Möglichkeit bietet die *B2B-Architektur*. Diese verwendet zwei EJB-Server, die je einen Web- und einen EJB-Container haben. Diese beiden Server können untereinander kommunizieren, meist via REST oder SOAP.

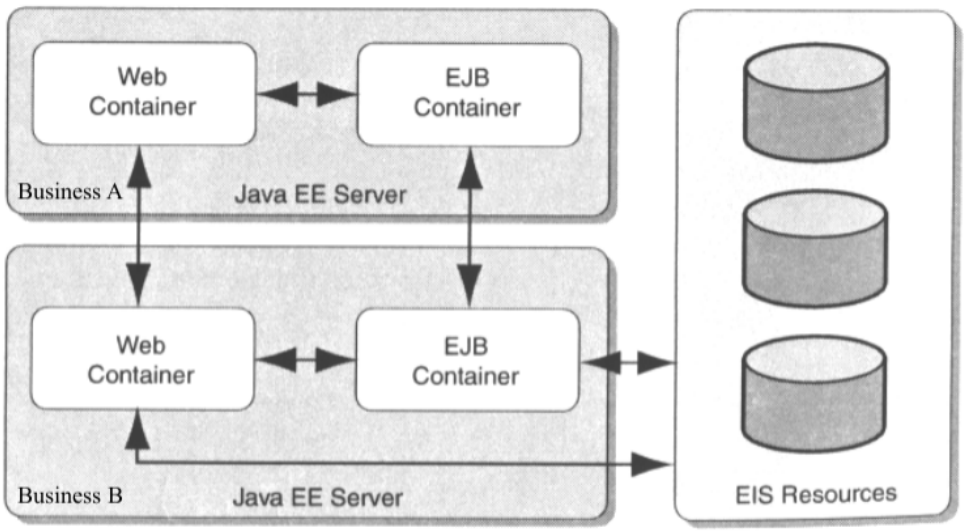

Abb. 3.3: B2B-Architektur

Last but not Least kann auch eine *Web Service Architektur* verwendet werden, wo ein EJB-Container gewisse Funktionen veröffentlicht und eine Stateless Service Bean dient als Zugriffspunkt, die diese veröffentlichte Funktion schliesslich ausführt. Diese Architekturen verwenden meist Messaging.

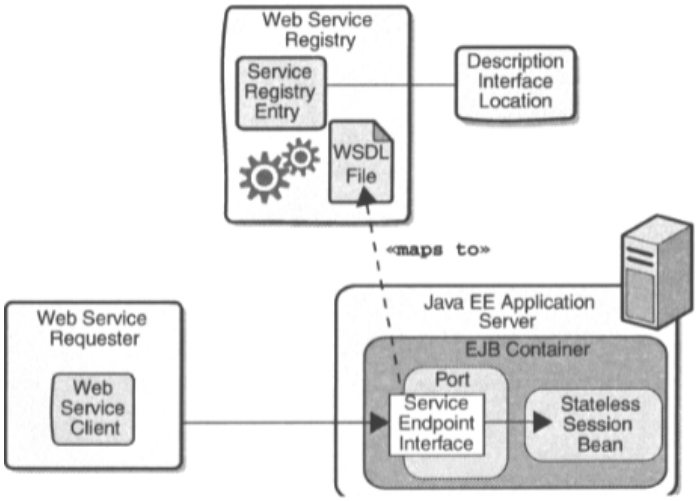

Abb. 3.4: Web Service Architektur

3.2 Komponenten-basiertes Entwickeln

Diese Art von Entwickeln basiert auf dem Motto *Warum das Rad neu erfinden, wenn man es auch einfach einbinden kann?*.

JavaEE unterstützt die Verwendung von Komponenten. Komponenten sind eigenständige Software-Elemente, die 'as-is' eingebunden werden können. Das heisst, sie können ohne Änderungen mit anderen eingebunden Komponenten verknüpft und ausgeführt werden.

In der Webshop Applikation wurde zum Beispiel die Komponente des EntityManagers verwendet, um nicht eine ganze Datenbankanbindung in Java programmieren zu müssen. Jeder

Programmierer kann jedoch auch seine eigenen Komponenten schreiben. So sind alle Servlets eigene Komponenten, auf welche über die extrahierten Interfaces zugegriffen werden kann.

Komponenten-basiertes Entwickeln bietet einige Vorteile:

- Wiederverwendung von Komponenten
- Austausch von Komponenten
- Lose Kopplung
- Getrennte Entwicklung möglich
- Bessere / gezielte Skalierung
- Verschiedene Sprachen
- Verschiedene Environments

Komponenten werden durch *Interfaces* kontrolliert und kommunizieren nicht direkt miteinander. Die Methoden werden ausschliesslich über die Interfaces aufgerufen.

3.3 MVC

MVC steht *Model View Control* und ist eine Design-Pattern, die beschreibt, wie eine Applikation aufgebaut ist.

MVC basiert auf dem Komponenten-basierten Entwickeln und besagt, dass Model (Daten), Control (Logik) und View (Präsentation) voneinander getrennt sein sollten (→ 3-Tier Architektur (Abb. 3.1)) und jede Komponente bei Bedarf ausgetauscht werden können sollte ohne die anderen beiden zu beeinträchtigen.

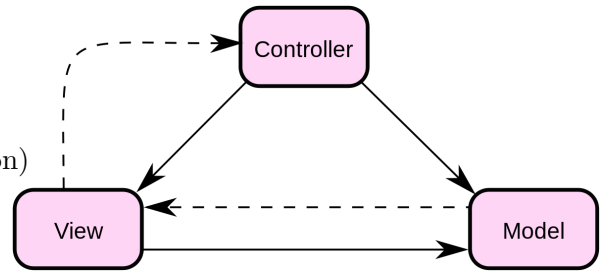

Abb. 3.5: MVC

Model: *Daten Logik*

- Datenbank-Anbindung (oder sonstige Daten, wie z.B. JSON-File) mit CRUD-Funktionalitäten
- Kommuniziert mit mit dem Controller - Controller requested Daten und Model liefert sie
- Je nach dem kann Model auch View direkt mit neuen Daten updaten

View: *Front-End*

- Was der End-User tatsächlich sieht (z.B. HTML/CSS Frontend bei Web-Apps)
- Kommuniziert mit dem Controller, der Daten zur Verfügung - z.B. über dynamische Variablen (siehe JSF mit Servlets oder Angular)

Controller *Logik*

- Verarbeitet User-Input von der View
- Requestet verlangte Daten vom Model und verarbeitet sie
- Übergibt Daten der View

3.4 Thread Safety

Aufgrund dessen, dass Java Multithreading unterstützt, müssen (oder sollten) gewisse Vorsichtsmassnahmen getroffen werden, um Datenkorruption oder Programm-Crashes zu verhindern.
Die wichtigsten Sicherheitsvorkehrungen:

Instanzvariablen: Instanzvariablen sollten so private wie möglich behalten werden (optimalerweise *local* und/oder *final*) um den Zugriff von anderen Threads zu erschweren oder zu verunmöglichen

Klassenvariablen: Möglichst keine Klassenvariablen verwenden. Auch wenn `synchronized` verwendet wird, kann man nicht darauf gehen, dass nicht andere Methoden auch auf diese Variable zugreifen (z.B. via `getter-` und `setter-` Methoden)

Ressourcen-Zugriff: Aufpassen bei der Vergabe von Zugriff auf externe Ressourcen. So ist der Zugriff auf das Filesystem sehr kritisch bei mehreren Threads und kann zu korrupten Dateien führen.

`synchronized` Bei kritischen Codestellen sollte zwingend die `synchronized`-Methode verwendet werden:

```
1  synchronized(this){
2    //kritischer Code - Hier ist immer nur ein Thread auf einmal
3  }
```

3.5 Servlet Programmieren

3.6 Server Domain Mappings und Filter

4 EJB-Tier

EJB steht für "Enterprise Java Bean". Sie sind serverseitige Komponenten, die die Business-Logik einer Applikation zusammenfassen. Sie stellen Services wie z.B. Transationen, Logging oder Security zur Verfügung.
Grundsätzlich wird zwischen vier Arten von Beans unterschieden:

4.1 Typen von EJBs

@Stateless: Stellen zustandlose Dienste zur Verfügung (Jede Anfrage wird als neue Anfrage behandelt, es wird nichts zwischengespeichert)

@Stateful: Stellen zustandsbehaftete Dienste zur Verfügung (Es werden gewisse Informationen zwischengespeichert, die bei späteren Anfragen wiederverwendet werden können)

@Singleton: Existiert genau einmal in der gesamten Applikation und stellt einen zustandsbehafteten Service zur Verfügung

@MessageDriven: Reagieren auf asynchrone Messages

Diese Beans sind nicht automatisch von überall her sichtbar und können auch nicht von überall her benutzt werden. Es muss eine Sichtbarkeit festgelegt werden:

@Local: Von allen Komponenten innerhalb derselben JVM sichtbar (Es können mehrere Applikationen auf einem Applikationsserver laufen. Diese können nun alle auf diese Bean zugreifen)

@Remote: Auch für Anwendungen sichtbar, die ausserhalbe der JVM laufen

@LocalBean: Nur innerhalb des EAR-Projekts der Applikation sichtbar, also ausschliesslich innerhalb dieser Applikation

Typen von EJBs

4.2 Dependency Injection

Anstelle, dass ich zum Beispiel mühsam eine Datenbank-Anbindung mit Java implementiere, kann ich die auch einfach einbinden, da sich jemand anders schon die Mühe gemacht hat, das Ding zu schreiben. Einen solchen `PersistenceContext` kann ich mit zwei Zeilen Code einbinden:

```
1 @PersistenceContext
2 private EntityManager em;
```

Wenn ich nun etwas in die Datenbank speichern will, kann ich auf eine Methode dieses EntityManagers zugreifen

```
1 em.persist(Object object);
```

5 JEE Security

6 Messaging Services

7 REST

8 JNDI & LDAP

9 CDI

10 Architektur & Agiles Vorgehen