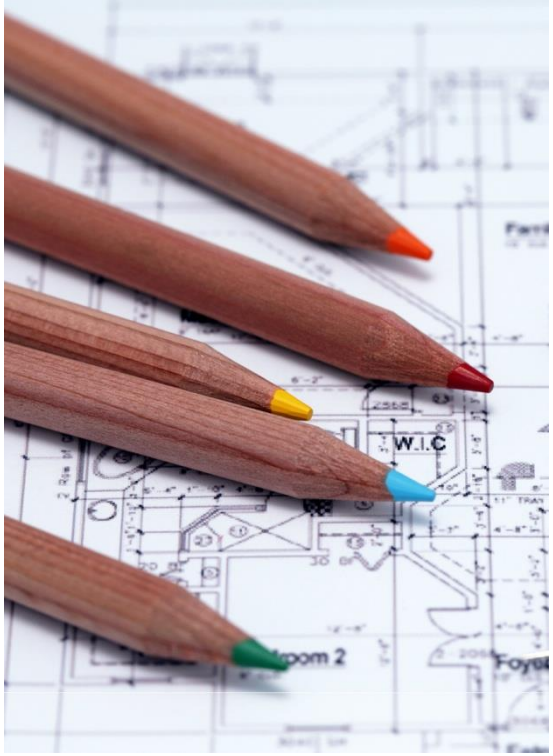


ARS

Java Server Faces

Weiterführende Themen

ARS

Überblick

- Stages
- Pluggability
- Resource Handling
- Custom Scopes

ARS

Stages

Konzept

- Ausführung der Applikation in unterschiedlichen Zielsystemen
  - Entwicklerrechner
  - Testsystem
  - Produktivsystem
- Verhalten in Zielsystemen unterschiedlich
  - Logging
  - Debugging-Informationen
  - Views mit zusätzlichen (automatisierten) Ausgabekomponenten

ARS

Stages

Umsetzung

- Enumeration `javax.faces.application.ProjectStage` für Zielsysteme
- Setzen der Stage
  - Web Deployment Deskriptor

```
<context-param>
  <param-name>javax.faces.PROJECT_STAGE</param-name>
  <param-value>Development</param-value>
</context-param>
```

  - JNDI-Umgebungsvariable `java:comp/env/jsf/ProjectStage`
  - Standardwert: `ProjectStage.Production`
- Abfrage über Application-Objekt

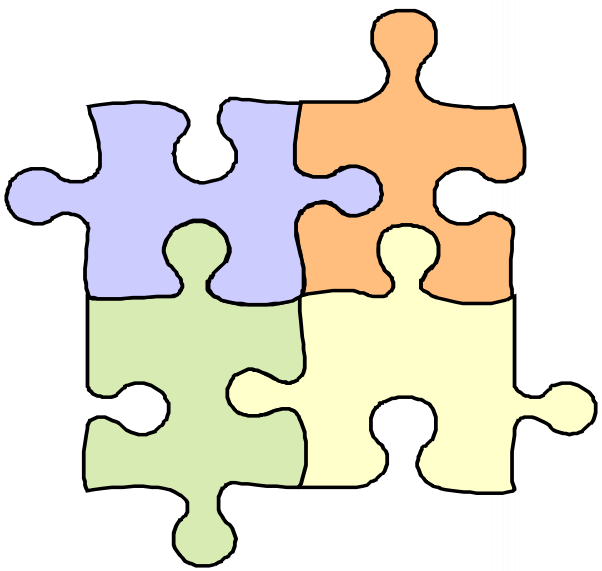
```
FacesContext.getCurrentInstance().getApplication().getProjectStage()
```

ARS

Pluggability

Konzept

- Erweiterbarkeit einer Anwendung um spezielle Funktionalitäten
- Anwendungsfall JSF:
  - Anpassungen des Frameworks (Handler, EL Resolver)
  - Erweiterungen der Applikation (Validatoren, Konverter)
  - Sprachpakete (Resource Bundles)
  - Corporate Designs
  - Tag Libraries
  - UI Components



Pluggability

Umsetzung

- Fragmente im Classpath der Webapplikation (`WEB-INF/lib/*.jar`)
  - Durchsuchen nach Metadaten
    - Annotierte Klassen (Managed Beans, Validatoren etc.)
    - Datei `/META-INF/faces-config.xml`
    - Dateien `/META-INF/*.taglib.xml`

ARS

Resource Handling

Konzept

- Faces Servlet stellt Web Ressourcen (CSS, JS, Bilder, …) bereit
  - Integriertes Caching
  - Pluggability durch Zugriff über Classloader
  - Versionierung / Internationalisierung von Ressourcen
  - Anpassung durch eigenen Resource Handler

ARS

Resource Handling

Umsetzung

- Ablegen von Ressourcen unter `/resources/` (Webmodul) bzw. unter `/META-INF/resources` (Fragment)
- Integration von Versionen im Dateipfad
  `[locale/][libraryName/][libraryVersion/]resourceName[/resourceVersion]`
  - z.B. `/resources/de/css/1_0/layout.css/1_1.css`
  - z.B. `/resources/css/layout.css`
- Verlinkung in Facelet über
  - Expression `#{resource['css:layout.css']}`
  - Facelet Tags zur Integration von Stylesheets, Javascript und Bildern

```
<h:outputStylesheet library="css" name="layout.css" />
<h:outputScript     library="js"  name="script.js" />
<h:graphicImage     library="img" name="logo.gif" />
```

ARS

Resource Handling

Fallstrick: Verlinkte Ressourcen in CSS

- CSS per Resource Handler geladen
  - Erreichbarkeit über Faces Servlet je nach URL Mapping
    - z.B. `/javax.faces.resource/layout.css.xhtml?ln=css`
- Relative URLs in CSS nicht mehr gültig
  - Faces Servlet auf Dateiendung `*.xhtml` gemappt
  - Library Name als URL-Parameter
- Lösungsansätze
  - Expression im CSS

```
body {
  background-image: url("#{resource['css:logo.gif']}");
}
```

  - Eigener Resource Handler mit Ersetzungsmechanismus, siehe http://ralfzahn.wordpress.com/2011/11/21/annoying-jsf-part-1/

Custom Scopes

Konzept

- Erweiterbarkeit der bestehenden Gültigkeitsbereiche um eigene
  - Speziell für Applikation
  - Erweiterung durch Frameworks, Komponentenbibliotheken etc.
    - Pluggability

Umsetzung

- Verwaltung einer Map über eigenimplementierten Mechanismus
  - JSF Managed Bean oder Erweiterung des EL Resolvers
- Angabe durch Referenzierung auf diese Map

```
<managed-bean>
  ...
  <managed-bean-scope>
    #{myCustomScope}
  </managed-bean-scope>
</managed-bean>
```

```
@ManagedBean(name="book")
@CustomScoped("#{myCustomScope}")
public class Book {
  ...
}
```