

Let's Rock

# Portlet-Spezifikation 2.0 (JSR 286)

Reza Nazarian

JSR 286, der Nachfolger der bisherigen Portlet-Spezifikation gemäß JSR 168, bringt viele Neuerungen und Verbesserungen mit sich. Der Standard soll endlich für Vereinheitlichung in der Welt der Java-Portale sorgen.

## Einleitung

Portlets sind präsentationsorientierte Webkomponenten, die jeweils eine individuelle Sicht auf Geschäftsanwendungen bieten. Lose gekoppelt und als fensterbasierte Applikationen werden sie von einem Portlet-Container, der einen Aufsatz zu einem Servlet-Container bildet, zu einer Portal-Applikation im Browser aggregiert. Portlets generieren dabei innerhalb ihres Fensters lediglich Markup-Schnipsel und keine vollständigen Seiten.

Eine Portal-Applikation ist in der Regel personalisiert und anpassbar, sodass in einem Unternehmensportal unterschiedliche Abteilungen wie Vertrieb und Kundenservice jeweils über eigene Sichten auf die gleichen Geschäftsanwendungen verfügen.

Ein Portal-Server bringt neben dem Portlet-Container zusätzlich eine Reihe von Enterprise-Technologien wie ein Content-Management-System und Collaboration-Software mit. Bekannte Portal-Server sind beispielsweise IBM Websphere Portal, SAP Netweaver Portal und Life-ray Portal.

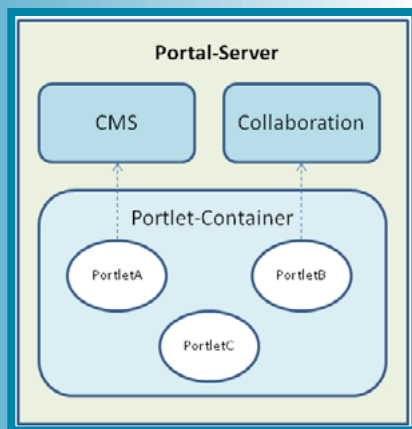


Abb. 1: Komponenten eines Portal-Servers

## Die bisherige Portlet-Spezifikation (JSR 168)

Im Zuge von Web 2.0 und SOA und den damit aufkeimenden aggregierten Webanwendungen wurden von diversen Herstellern Portlet-Technologien für ihre Portal-Produkte entwickelt. Schließlich passte Web 2.0 nicht so richtig in die Business-Welt und man brauchte ein Enterprise 2.0. Inzwischen hat nahezu jeder größere Anbieter von Enterprise-Software einen entsprechenden Portal-Server im Angebot. Häufig wurde aber mit jedem Server auch eine eigene Philosophie und eine eigene Architektur umgesetzt.

Um einen einheitlichen Standard für Java-Portlets zu schaffen, wurde der JSR 168 [JSR168,Kuss05] angeregt und 2003 verabschiedet. Portlets sollten damit auf jedem konformen Portal-Server ohne Anpassungen lauffähig sein. Zusammen mit der Portlet-API wurde für konforme Portlet-Container zudem der Rahmen definiert, in dem Portlets ausgeführt werden.

Portlets durchlaufen einen vierstufigen Lebenszyklus. Dazu gehören die einmalige Initialisierung, die Verarbeitung von Benutzereingaben, das Rendering und die Destruktion, die dann aktiv wird, wenn man das Portlet aus der Portal-Applikation entfernt. Solange das Portlet in der Portal-Applikation aktiv ist, werden sequentiell Eingaben verarbeitet und das Portlet-Markup gerendert. Alle Portlets in einem Portlet-Container durchlaufen die Stationen des Lebenszyklus zudem parallel. Jede dieser Stationen wird durch eine korrespondierende Interface-Methode der API abgebildet. Dementsprechend kann ein Portlet die folgenden Methoden implementieren:

- ▼ `init()` für die Initialisierung,
- ▼ `processAction()` für die Verarbeitung von Benutzereingaben (Action-Phase),
- ▼ `render()` für die Generierung von Markup (Render-Phase),
- ▼ `destroy()` als Destruktor.

Ein Portlet kann sich in einem von drei verschiedenen Modi, die das Verhalten des Renderings beeinflussen, befinden:

- ▼ `View`: für die Anzeige von Daten,
- ▼ `Edit`: für die Bereitstellung von Eingabemöglichkeiten,
- ▼ `Help`: für Informationen zur Benutzung.

Der Modus kann über die Methode `processAction()` modifiziert und von allen Portlet-API-Methoden ausgelesen werden. In der Regel ändert der Benutzer den Modus für einzelne Portlets über Bedienelemente im Portlet-Container.

Wie bei Servlets findet der Aufruf eines Portlets über einen Portlet-Request und die Antwort des Portlets über einen Portlet-Response statt.

## Schwächen von JSR 168

Portlets waren bisher isolierte Komponenten, da es nur unzureichende Möglichkeiten für die Kommunikation und Koordination zwischen den Portlets gab. Gerade aber für eine Portal-Applikation im Enterprise-Umfeld, für die beispielsweise koordiniert die aktuellen Kundendaten innerhalb der gesamten Portal-Applikation verfügbar sein müssen oder wo Produktauswahl und Warenkorb zwei voneinander getrennte Komponenten/Portlets sind, macht sich dieses fehlende Feature schmerzlich bemerkbar. Zwar können Portlets auch über Session-Attribute Objekte austauschen, nur sind diese nicht im selben Portlet-Zyklus zugreifbar, sondern immer erst ab dem Folgenden.

Gängige Web-Frameworks wurden nur unzureichend unterstützt. So lässt sich Ajax auf Client-Seite sehr umständlich ohne Portlet-Kontext verwenden und das Dispatching auf JSPs und Servlets, um z. B. Struts-Applikationen einzubinden, ist nur in der Render-Phase zugelassen.

Die Spezifikation genügte einigen Anbietern von Portal-Servern erwartungsgemäß nicht, sodass proprietäre Erweiterungen oder neue Lösungen entwickelt bzw. weiterhin auf bestehende gesetzt wurden. Bekannte Beispiele hierfür sind die Websphere Portlets von IBM und die iViews von SAP.

Die Folge von proprietären Portlet-Umsetzungen ist Inkompatibilität: Portlets, die speziell für den Portal-Server entwickelt wurden, sind auf den Servern anderer Hersteller nicht lauffähig.

## JSR 286 - und alles wird gut

Im Zuge der neuen Spezifikation machte sich das Gremium von JSR 286 [JSR286] daran, die bekannten Schwachstellen zu beseitigen. Dabei wurde aber auf der bestehenden Spezifikation aufgebaut und nicht wieder bei Null angefangen. Mit der Einführung von Events und Public Render Parameters wur-



de einer der Hauptkritikpunkte, die fehlende Kommunikation und Koordination zwischen Portlets, angegangen. Ebenso wurden die Möglichkeiten zur Unterstützung von Web-Frameworks deutlich verbessert. Mit den Portlet-Filtern wurde zudem ein gängiges Verfahren eingeführt, das man bereits von anderen Frameworks kennt (z. B. Servlet-Filter). Das Gremium des JSR 286 benötigte bis zur Fertigstellung über zweieinhalb Jahre (November 2005 bis Juni 2008).

Neue Portlet-Container, die gemäß JSR 286 entwickelt wurden, sind abwärtskompatibel zu JSR 168 und unterstützen so bestehende Portlets.

### JSR 286 im Überblick

- ▼ Events für aktives Messaging zwischen Portlets
- ▼ Innerhalb des Portals verfügbare Public Render Parameters
- ▼ Verbesserte Ajax-Unterstützung
- ▼ Portlet-Filter
- ▼ Verbessertes JSP/Servlet-Dispatching

## Der erweiterte Portlet-Lebenszyklus

Für die Unterstützung der neuen Portlet-Events wurde das Interface `EventPortlet` eingeführt, das die Methode `processEvent()` anbietet. Die Methode ist nicht direkt eine neue Phase im Portlet-Lebenszyklus sondern vielmehr ein weiterer Schritt, der in der Action-Phase direkt nach `processAction()` und noch vor der Render-Phase ausgeführt wird.

Die neue Resource-Phase, umgesetzt durch die Methode `serveResource()` aus dem Interface `ResourceServingPortlet`, soll vor allem die Arbeit mit Ajax erleichtern.

### Events für die Kommunikation

Ein herausstechendes Merkmal, das die Kommunikation unter den Portlets verbessern soll, ist das neue Eventing. Events von einem Portlet zu einem anderen Portlet zu senden, war vorher nicht möglich. Mit dieser Erweiterung und den im Folgenden vorgestellten Public Render Parameters ist endlich eine echte Kommunikation zwischen den Portlets umgesetzt worden. Nach wie vor wird die lose Kopplung der Portlets aber beibehalten. Sender bzw. Empfänger sind dem Portlet beim Eventing nämlich unbekannt. Der Portlet-Container dient als Event-Broker und die Verdrahtung, also welches Portlet welche Events verschickt und empfängt, wird im Deployment-Deskriptor vorgenommen. Ein Event kann dabei beliebig viele Sender und Empfänger haben. Festgelegt wird hier auch der Datentyp der versendeten Information. Erlaubt sind hierfür Java-Simple-Types (`java.lang.String`, `java.lang.Integer`, `java.lang.Date` usw.) und per JAXB nach XML serialisierbare Java-Objekte.

Trotz aktivem Verschicken von Nachrichten wird das neue Portlet-Eventing allerdings nicht als Ersatz für verlässliches Java-Messaging wie JMS angesehen. Schließlich muss ein Portlet-Container nicht garantieren, dass Events ihren Weg zum Empfänger finden. Der Einsatzzweck vom Portlet-Eventing liegt also eher darin, andere Portlets aktiv zur Änderung ihres Contents zu veranlassen.

Events werden vom Portlet in der Lifecycle-Methode `processEvent()` empfangen und verarbeitet. Gesendet werden dür-

fen Events in den Lifecycle-Methoden `processAction()` und `processEvent()`. In der Regel sollten Events über erstere Methode gesendet werden, weil sie dann in der nächsten Phase desselben Zyklus von den empfangenden Portlets über die `processEvent()`-Methode ausgewertet werden.

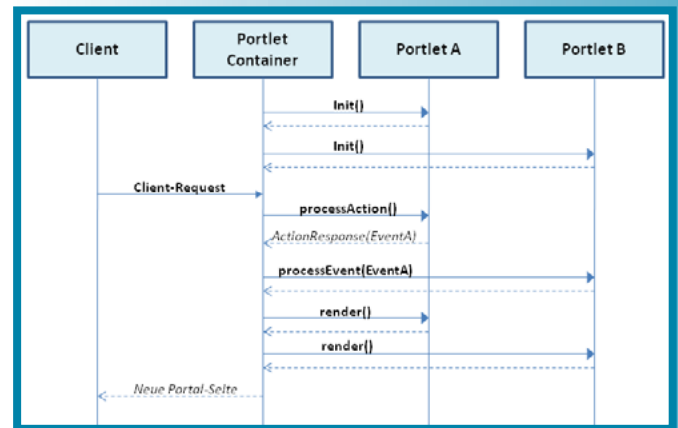


Abb. 2: Lifecycle eines JSR286-Portlets

## Public Render Parameters für die Koordination

Der Austausch von Parametern war bisher zwischen Portlets nur sehr eingeschränkt möglich. So mussten dafür Session-Attribute verwendet werden, die von anderen Portlets erst in der nächsten Render-Phase ausgelesen werden konnten. Als Public Render Parameters sind diese mit JSR 286 nun im gesamten Portlet-Container für Lese- und Schreiboperationen verfügbar, wo sie über den Portlet-Request gelesen und über den Portlet-Response gesetzt werden.

Render-Parameter sind immer Java-Strings und im Deployment-Deskriptor definiert. Über den Deskriptor werden sie dem Portlet-Container bekannt gemacht. Ein Portlet hat dann Zugriff auf den Parameter, wenn dieser ebenfalls im Deskriptor mit dem Portlet verdrahtet wird. Ein Parameter ist also nicht automatisch für alle Portlets in der Portal-Applikation sichtbar. Mit dieser Möglichkeit lassen sich nun solche Parameter austauschen, die für ein oder mehrere Portlets Einfluss auf die Render-Phase haben und für die beteiligten Portlets von ähnlicher Relevanz sind.

Wozu überhaupt Public Render Parameters, es gibt doch Portlet-Events? Im Vergleich zu Events findet über Public Render Parameters in der Tat kein aktives Messaging („Hallo! Es hat sich etwas geändert!“) statt. Über solche Parameter lässt sich aber durchgehend Einfluss auf das Render-Verhalten eines oder mehrerer Portlets nehmen, zudem wird auch weniger Overhead als bei Events produziert, da die betreffenden Portlets keine Event-Phase bedienen müssen. Ein weiteres Feature: Die Parameter werden vom Portlet-Container in der URL kodiert, sodass sich Rendering-Zustände cachen und bookmarken lassen.

### Portlet-Filter

Technologisch identisch zu den Servlet-Filtern wurden auch für Portlets nun entsprechende Filter spezifiziert. Dabei lassen sich ein oder mehrere Portlet-Filter beliebig vielen Portlets zuweisen. Filter haben dann Zugriff auf den Ein- und Ausgabe-

strom des jeweiligen Portlets, um z. B. Parameter umzurechnen, das Ausgabe-Markup eines Portlets anzupassen oder um Berechtigungen zu handhaben.

Ein Portlet-Filter wird im Deployment-Deskriptor definiert und auf ein oder mehrere Portlets gemappt. Hier wird zudem angegeben, in welchen Lifecycle-Phasen der Filter aktiv werden soll.

## Das erweiterte Servlet-Dispatching

Mit JSR 168 war das Servlet-Dispatching lediglich in der Render-Phase möglich, nun steht dies endlich in allen Phasen zur Verfügung. Markup kann von verwendeten Servlets und JSPs dabei nach wie vor nur in der Render-Phase generiert werden.

Durch das erweiterte Servlet-Dispatching, das Includes und Forwards erlaubt, kann sowohl Logik als auch die Generierung von Markup auf Servlets und JSPs ausgelagert werden. Das Portlet agiert dabei als Controller und kann beispielsweise in der Render-Phase je nach Modus (View, Edit, Help) per Include auf eine passende JSP dispatchen, die dann das entsprechende Markup generiert. Portlet-Request und -Response stehen Servlets und JSPs dabei zur Verfügung. Gängige, auf Servlets basierende Web-Frameworks oder einfach vorhandene Webanwendungen lassen sich so deutlich besser einbinden als vorher.

## Resource Serving für Ajax

Gemäß JSR 168 können Portlets Links erzeugen, die direkt auf sich selbst führen (Direct Links). Über diesen Weg ist es erlaubt, zusätzlich auch Nicht-Markup-Ausgaben bereit zu stellen (z. B. binäre Dateien wie PDFs). Der Portlet-Kontext, um beispielsweise Parameter oder Render-Modi auszulesen, steht dann allerdings nicht mehr zur Verfügung, was beispielsweise bei der Generierung von dynamischen Grafiken ein Manko ist. Die neue Spezifikation bietet nun die Erzeugung sogenannter Resource URLs über das Portlet an. Deren Verwendung als Linkziel durch den Client triggert die `serveResource()`-Methode der neuen Resource-Phase. Resource URLs verlinken genau wie Direct Links auch direkt auf das Portlet, jedoch steht bei der Nutzung nun auch der vollständige Portlet-Kontext zur Verfügung. Der Render-Zustand (Public Render Parameters, Portlet-Modus usw.) des Portlets darf dabei aber nicht verändert werden.

Diese Neuerung ermöglicht vor allem eine verbesserte Unterstützung von client-seitigem Ajax. Warum ist dies so? Die `serveResource()`-Methode im Portlet wird direkt über die Resource URL vom Client nach vollendetem Rendering aufgerufen. Da das Portlet nun auch andere Ressourcen als Markup-Fragmente bereitstellen kann, werden Ajax-typische XMLHttpRequests über die Resource URL z. B. mit JSON bedient. Dieser asynchrone Aufruf kann beliebig häufig getätigt werden, ohne dass ein erneuter Lifecycle aller Portlets gestartet wird, der zum Rendering aller Portlets führt.

## Ziel erreicht?

Die Portlet-Spezifikation 2.0 (JSR 286) bringt viele sinnvolle Verbesserungen mit sich. So sind Portlets innerhalb einer Portlet-Applikation keine nahezu isolierten Einzelkomponenten mehr, sondern können untereinander kommunizieren und Daten austauschen. Dafür wird die lose Kopplung der Portlets nicht aufgegeben. Mit dem verbesserten Servlet-Dispatching sowie der Ajax-Unterstützung sind zwei Probleme beseitigt worden, die bisher die Entwicklung ebenfalls erschwert haben.

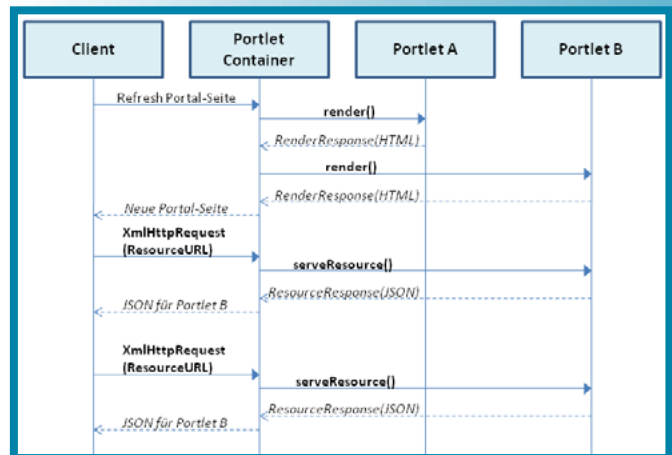


Abb. 3: Lifecycle mit `serveResource()`-Methode

Die Einbindung gängiger Web-Frameworks ist somit verbessert worden und deutlich komfortabler als vorher.

Allerdings spiegeln all diese Neuerungen bereits vorhandene Funktionalitäten von proprietären Technologien wider. Wirklich Neues gibt es nicht. Wünschenswert wäre zudem eine vereinheitlichte Schnittstelle auf User- und Rechtenmanagement gewesen. Nach wie vor werden die Portlets auf das proprietäre API des jeweiligen Portal-Servers zugreifen müssen, um diese Funktionalitäten zu verwenden.

Ein wichtiges Ziel für die Portlet-Welt ist und bleibt aber die Vereinheitlichung. Weg von proprietären Portlets, die nur auf einem Portal-Server lauffähig sind, und hin zu konformen Portlets, die sich auf allen gängigen Portal-Servern betreiben lassen. Da sich alle relevanten Hersteller von (Java-)Portal-Servern zu JSR 286 bekannt und zumeist entsprechende Portlet-Container bzw. Portal-Server veröffentlicht haben, ist die Grundlage hierfür somit geschaffen.

## Links

- [JSR168] Java Specification Request 168: Portlet Specification, <http://jcp.org/en/jsr/detail?id=168>
- [JSR286] Java Specification Request 286: Portlet Specification 2.0, <http://jcp.org/en/jsr/detail?id=286>
- [Kuss05] T. Kussmaul, Die Java-Portlet-Spezifikation, in: JavaSPEKTRUM, 3/05, [http://www.sigs.de/publications/js/2005/03/kussmaul\\_JS\\_03\\_05.pdf](http://www.sigs.de/publications/js/2005/03/kussmaul_JS_03_05.pdf)
- [Naz09] R. Nazarian, Marktübersicht Portlet-Technologien, in: JavaSPEKTRUM, 1/09



**Reza Nazarian** ist Consultant bei der Unternehmensberatung Acando in Hamburg. Seine fachlichen Schwerpunkte liegen in den Bereichen Portal-Technologien und Content-Management-Systeme. Reza Nazarian ist seit 2002 in der IT-Branche tätig und hat Erfahrungen als Berater, Technischer Projektmanager und Software-Ingenieur gesammelt.  
E-Mail: [reza.nazarian@acando.de](mailto:reza.nazarian@acando.de).