Apache Struts

- Designschwächen überwinden -

Oktober 2005

Manfred Wolff, wolff@manfred-wolff.de

Frameworks sollten möglichst unabhängig von einer bestimmten Technologie entwickelt werden. POJI und POJO sind dabei eine der Hauptforderungen: Das Framework sollte an den Schnittstellen nur solche Objekte verwenden, wie sie im Java SE angeboten werden. Schon bei Betrachtung der Signatur der `execute()`-Methode einer Struts Action-Klasse zeigt sich, dass diese Maxime nicht eingehalten wird. Diese und andere Schwächen führen teilweise zu Problemen beim Design einer Software, die mit Struts arbeitet - führen aber spätestens definitiv zum Problem bei der Portierung einer Struts-Anwendung auf eine andere Technologie.

Dieser Artikel beschreibt die Hauptdesignschwächen des bekannten Apache Frameworks und diskutiert Lösungen. Der Hauptschwerpunkt wird auf die Einführung eines Context-Objekts gelegt, weil hier eine Menge von technologieabhängigen Details „versteckt" werden können.

Einleitung

Struts ist ein Präsentationsframework für webbasierte Anwendungen, dessen Ablaufumgebung ein Servlet-Container ist. Von daher ist nichts Falsches daran, wenn in den Signaturen des Frameworks Objekte der Java EE Distribution zu finden sind, wie z.B. der `HttpServletRequest`. Andererseits ergibt sich, wie auch im Bereich EJB die berechtigte Frage, warum eine einmal entwickelte (Präsentations-)Logik nur für einen bestimmten Container entwickelt werden soll. Deshalb sind im Bereich der Geschäftslogik leichtgewichtete Container wie Spring oder Hivemind im Vormarsch.

POJO ist hier das Stichwort. Die Entwicklung von Objekten, welche nur Objekte der Java SE (Java Standard Edition) an den Signaturen kennen, macht die implementierte Logik nicht nur unabhängig von einer bestimmten Ablaufumgebung, sondern hilft diese auch testbar zu machen. Struts-Anwendungen können nur im Servlet-Container getestet werden oder mit bestimmten Frameworks wie *StrutsTestCase* oder *Cactus*.

Dieser Artikel geht auf einige Designschwächen von Struts ein und diskutiert Lösungen, wie diese in Projekten von Beginn an vermieden werden können.

Vorab: Wenn hier von Designschwächen die Rede ist, dann fußen diese aus meinen Erfahrungen aus vielen Struts Projekten. Das schmälert keinesfalls

die Leistung der Struts Autoren. In der Mailing-Liste von Struts sind die Themen dieses Artikels bereits mehrfach diskutiert worden und wie immer gibt es verschiedene Meinungen. Ich persönlich halte es z.B. für ein großes Problem, dass die Struts Action-Klasse nicht als Interface ausgeprägt ist. Die Struts-"Macher" halten dieses für ein gutes Design, weil sonst Objekte der Geschäftslogik durch ein einfaches *implements* zu Präsentationsobjekten mutieren können. Aus meiner Sicht ist dieses Argument nicht schlüssig, weil dieses durch ein *extends* auch in der bisherigen Fassung möglich ist. Hier sieht man: Es gibt unterschiedliche Auffassungen, und das ist auch gut so.

Die Action-Klasse als Interface

Im Struts-Wiki wird es als *best practice* bezeichnet, eine Basisklasse von der Original Struts-Action abzuleiten, um so gemeinsame Funktionalität zu kapseln. Ich selber halte dieses auch für gute Praxis. Tatsächlich wird es eine ganze Vererbungshierarchie geben, in der technische- und domainspezifische Funktionen in Basisklassen gekapselt werden.

Ein weiteres „best practice" ist die Verwendung der mitgelieferten Struts Action-Klassen, wie z.B. der *DispatchAction*. Diese leiten auch direkt von der Struts Action-Klasse ab. Da es unter Java keine Mehrfachvererbung von Implementierungsklassen gibt, bekommt man hier ein Problem: Entweder werden die Funktionalitäten der eigenen Basisklasse verwendet oder die Funktionalität der Standard Struts-Action Klassen. Beides zusammen ist nicht möglich.

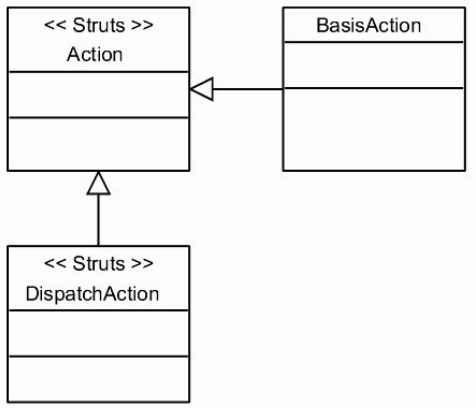

Abbildung 1: Dilemma der Mehrfachvererbung unter Java

In Abbildung 1, wird dieses deutlich. Wozu gibt es überhaupt eine Struts Action-Basisklasse? Damit das Framework eine Methode in der Struts Action-Klasse mit einer bestimmten Signatur vorzufindet. Das Framework ruft

nämlich im *RequestProcessor* die `execute()`-Methode der Struts Action-Klasse auf. Genau dieses ist der Vertrag und damit ist eigentlich schon alles gesagt: Vertragsbestandteile gehören in ein Interface.

Tatsächlich ist für den *RequestProcessor* nur die `execute()`-Methode der Struts Action-Klasse von Bedeutung. Alle anderen Methoden bieten Basisfunktionalitäten, die von der Anwendung aufgerufen werden können. Insbesondere werden hier Werte in den HTTP-Request geschrieben, wie z.B. Fehlermeldungen; eine ideale Aufgabe für eine *ActionHelper*-Klasse. Da die Struts Action-Klasse threadsafe ist, verbietet sich sowieso der Zugriff auf Instanzvariablen. So hält die Struts Action-Klasse tatsächlich nur eine Referenz auf das Servlet, um hier Werte zu lesen und zu schreiben.

Um das Dilemma der Mehrfachvererbung zu umgehen, muss man selbst ein Interface erstellen (wie in Abbildung 2: Lösung des Dilemmas mit Delegation) und eigene Ableitungen der Standard Action-Klassen kreieren. Diese erben von dem Original und delegieren die Basisfunktionalität - die im Interface als Vertrag vereinbart wurde - an die eigene Basis Action-Klasse weiter.

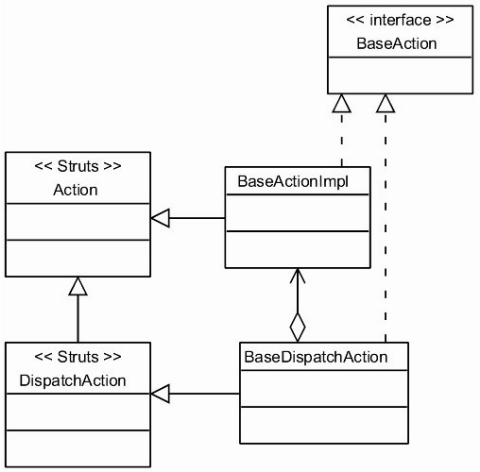

Abbildung 2: Lösung des Dilemmas mit Delegation

So kann z.B. eine *BaseDispatchAction* definiert werden, welche sowohl die Funktionalität der *DispatchAction*, als auch die Funktionen der *BaseActionImpl* verwenden kann.

Bleibt noch die Implementierung eines eigenen Request-Prozessors, der mit dem neuen Interface umgehen kann. Dazu wird die Methode *processActionPerform()* überschrieben. Hier gibt es jetzt auch gleich die Möglichkeit, weitere Erweiterungen vorzunehmen, wie im nächsten Kapitel zu sehen ist.

Portable Action-Klassen schreiben

Struts bietet zwei `execute()`-Methoden an. Diejenigen, die am häufigsten überschrieben wird, ist diese:

```
public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception {

    return null;
}
```

Listing 1: Signatur einer execute() Methode unter Struts

Wie schon in der Einleitung bemerkt, ist es nicht so schön, wenn Technologiespezifische Klassen oder Exceptions an den Signaturen verwendet werden, weil dann eine einmal geschriebene Action nicht mehr in einem anderen Zusammenhang wiederverwendet werden kann. Wird das Programm z.B. auf Swing portiert, müssen alle Action-Klassen wieder verändert werden. Um Action-Klassen portabel zu machen sind grob zwei Dinge nötig:

1. Die Signatur der Methode muss verändert werden. Am Besten haben sich hier Context-Objekte bewährt.
2. Es dürfen für das Abspeichern von Werten nicht mehr die Container-Spezifischen Contexte wie Anwendungscontext, Sitzungscontext etc. genutzt werden.

Zweiteres wird im nächsten Kapitel behandelt.

Context-Objekte für unseren Zweck müssen nicht mehr neu erfunden werden, sie existieren bereits im Jakarta `commons-chain` Framework der Apache Group. Dort wird ein Interface `Context` definiert, welches dann für die verschiedenen Zwecke wie Servlets, Portlets etc. ausimplementiert wird. Für unsere Zwecke ist eine eigene Implementierung sinnvoll.

Um Action-Klassen wirklich vom Struts-Framework zu lösen, ist es aber notwendig noch etwas mehr zu tun: Der Workflow muss aus der Action-Klasse herausgelöst werden, dazu aber später mehr.

Zunächst zu den Context-Objekten. Das `commons-chain` Framework bietet neben dem `Context` Interface eine einfache Basisimplementierung an. Hier wird das Prinzip des „attribute transparency“ umgesetzt. Anwender

einer Context-Klasse brauchen so nicht zu wissen, ob ein Attribut ausgeprägt wurde oder ob es nur durch ein key-value Paar repräsentiert wird. Das ausgeprägte Attribut

```
private String name;

public String getName() { return name;}
public void setName(String aName) { name = aName;}
```

Listing 2: Ausgeprägtes Attribut im Context

kann transparent auch so gesetzt und ausgelesen werden;

```
...
String name = context.get("name");
context.set("name", "Wolff");
```

Listing 3: Lesen und Setzen von Werten durch Attribute-Transparency

Damit wäre es einfach möglich einen Context-Objekt für Struts Anwendungen zur Verfügung zu stellen (hier nur Auszüge):

```
public class StrutsContextBase
        extends ContextBase
        implements Context, StrutsContext {

    /** The actual struts action-mapping*/
    private ActionMapping mapping;

    /** The actual struts action-form*/
    private ActionForm form;

    /** The actual servlet-request */
    private HttpServletRequest request;

    /** The actual servlet-response */
    private HttpServletResponse response;
}
```

Listing 4: Ein einfacher Struts Context

Die Zugriffe könnten ausgeprägt werden oder durch das attribute-transparency durchgeführt werden. Die execute()-Methode der Struts Action wäre jetzt schon verschlankt

Was jetzt noch „stört“ ist vor allem der HttpServletRequest und der HttpServletResponse. Ziel ist es die Servlet-API auf der Ebene der Struts Action-Klasse zu eliminieren, um so ein POJO zu bekommen.

```
public void execute(StrutsContext context);
```

Listing 5: Verschlankte execute() Methode

Portables Session-Memory verwenden

Struts Anwendungen machen es Struts nach: Werte, die sich gemerkt werden, werden direkt im entsprechenden Context gespeichert. Ich halte dieses Vorgehen für nicht so gut:

- Werte, die in einem zentralen Context abgelegt sind bekommen den Status einer globalen Variablen, mit allen Problemen (Seiteneffekte beim Ändern).
- Durch globale Informationen werden Rollen- und Rechtesysteme ausgehebelt. Hier geht es vor allem um Werte, die im Anwendungs-Context abgelegt werden.
- Niemand hat einen Überblick welche Schlüssel existieren und kein Programmteil räumt Schlüssel wieder ab durch explizites Löschen.

Das Problem der nicht gelöschten Schlüssel kann mit Containern, die im entsprechenden Context gespeichert werden, umgangen werden.

Damit ein Session-Memory auch für die Businesslogik zugreifbar wird, muss es jedoch zwingend vom Servlet-Container abgespalten werden. Es ist ein grober Fehler, die Servlet-API auch noch in die Geschäftslogik zu übertragen und so unnötige Abhängigkeiten zu schaffen. Die Geschäftslogik sollte im besten Falle gar nichts von einer Präsentationslogik wissen.

Portierung des Application Context

Der Application-Context wird üblicherweise direkt im Servlet verwaltet. Diesen zu übertragen ist vergleichsweise einfach: Durch ein Singleton. Damit werden die Probleme der „globalen Variable“ allerdings nicht aus der Welt geschafft. Hier gilt es diese Art der Speicherung von Werten so wenig wie möglich zu benutzen.

Portierung des Session Context

Hier wird es komplizierter. Vor allem gibt es das Problem, dass die Anwendung nicht notifiziert wird, wenn eine Session z.B. durch ein Timeout nicht mehr gültig ist. Für einen unabhängigen Session Context sind zwei Dinge nötig:

1. Pro Nutzer des Systems muss ein Context erzeugt und registriert werden. Dazu könnte auch die SessionID genutzt werden, die von der HTTP-Session generiert wird, da diese nur durch einen String gebildet wird. Für andere Ablaufumgebungen müsste dann die Registrierungsmethode überschrieben werden oder eine Zeichenkette gefunden werden, die eine Session eindeutig identifiziert.

2. Der Context muss ein Verfallsdatum bekommen, damit er auch wieder deregistriert werden kann, falls ein Session Timeout aufritt. Dieses kann durch das commons-cache Framework realisiert werden. Gerade bei Webanwendungen kann nicht davon ausgegangen werden, dass ein explizites Logout des Benutzers gemacht wird.

Portierung des HTTP-Request

Der Request ist per Definition read-only. Von daher müssen sowohl die Attribute als auch die Parameter des HttpServletRequest-Objekts in entsprechende Hash-Maps kopiert werden. Dieses geschieht sinnvollerweise im RequestProzessor. TODO: Dazu sollte es eine Methode in den BeanUtils geben.

Portierung der HTTP-Resonse

Die Portierung des Responses ist deutlich komplexer. Tatsächlich müssen wir der Action eine Möglichkeit zu geben, plattformspezifische Parameter setzen zu können. Auch für andere Frameworks mag es notwendig sein, bestimmte Konfigurationsparameter etc. aus einer Action heraus zu setzen.

Den Struts Workflow isolieren

Die Elimierung von Struts aus der Action-Klasse. Das klingt etwas hanebüchen: Ich habe mich ja für Struts entschieden, warum soll ich jetzt Struts wieder elimenieren? Dafür kann es mehrere Gründe geben:

- Große Systeme überleben Technologien. Ich selber habe es erlebt wie ein Buchungssystem von proprietären Techniken zu Struts, Tiles und Hivemind migriert wurde. Das wird auch weiter geschehen z.B. die Portierung auf ein Rich-Client oder einer andere Technologie wie Java Server Faces. Portierungen verlaufen immer dann geräuschlos, wenn das Design von vornherein sehr Technologie-unabhängig gewählt wurde [1].

[1] Siehe dazu: Albrecht M., Burbach M., Stahlhut A., Wolff M., „Struts im praktischen Einsatz bei der TUI“, ObjektSpektrum 1/2005, S. 44-50

- Gerade wenn ein System für mehrere Kanäle konzipiert wird (Rich Client und Webauftritt), ist es wünschenswert nicht nur die Geschäftslogik, sondern auch einen großen Teil der Präsentationslogik wieder zu verwenden.
- Geschäftsprozesssteuerungen oder externe Workflowsystem geben den Workflow des Client vor. Dieses kollidiert mit der Struts Logik, wobei der Workflow vom Client vorgegeben wird. Letztendlich entscheidet die Action als Teil der Präsentationslogik welche Seite als nächstes geladen werden soll.

In medias Res

Prinzipiell ist es ganz einfach: Wir speichern alle Framework-spezifischen Dinge im Context-Objekt, auch den Rückgabewert. Für jedes Framework, welches als Präsentationsframework in Frage kommt wird eine Mapping-Funktion implementiert. Es werden außerdem zentrale, Framework-neutrale Rückgabewerte eingeführt, die von der Businesslogik gesetzt werden (Service to Worker Designpattern). Standard-Rückgabewerte wären z.B.:

```
static public final String succeeded = "succeeded";
static public final String failed = "failed";
```

Listing 6: Vereinbarung von Standard Rückgabewerten

Für das Struts-Framework übernimmt der neue *RequestProcessor* die Umsetzung der allgemeinen Rückgabewerte in ein entsprechendes Action-Forward.

```
public ActionForward convert2StrutsMapping(
        StrutsContext context) {

    ActionMapping mapping = (ActionMapping)
            context.getMapping();
    ActionForward forward = mapping.findForward(
            (String) context.getReturnValue);
    return forward;
}
```

Listing 7: Umwandlung des neutralen Rückgabewerts in ein ActionForward

Resumee

Contexte sind der zentrale Anknüpfungspunkt, die sehr Servlet-spezifischen Bestandteile aus den Struts Action-Klassen herauszulösen. Ein eigener Re-

quest-Prozessor sorgt dafür, dass der Context richtig gefüllt wird und die neutralen Rückgabewerte in ActionForward-Objekte umgewandelt werden. Mit dieser Technik können Action-Klassen als reine POJOs entwickelt werden und sind vollständig unabhängig von Struts und der Servlet API.

Die Einführung eines eigenen Action-Interfaces entkoppelt das POJO von der Struts Action-Implementierung und dient als zweiter Hebel, Probleme beim Design von Struts Anwendungen zu umgehen.

Oft sind es nur einfache Dinge, die geändert werden müssen um Designschwächen von Frameworks auszumerzen.