

Saubere Trennung

Jakarta Struts für PHP

von **Stefan Willkommer** und **Tim Wagner**

Es gibt in **PHP** mittlerweile viele Möglichkeiten eine **Trennung** des **Programmcodes** von der eigentlichen HTML/XML **Ausgabe** zu erreichen. Warum aber sollte man sich nicht eine viel genutzte Lösung einer anderen Programmiersprache zu nutze machen? Hierbei ist die Rede von dem Apache Open Source Projekt [Jakarta Struts](#). Viele Java Entwickler werden jetzt wahrscheinlich die Ohren spitzen. Struts ja, aber in PHP? Für alle anderen die mit Java nicht so viel am Hut haben gibt es eine kleine Erklärung.

Das Struts Framework ist ein Webapplication Framework welches das **Model View Controller** Design Pattern implementiert. Das MVC Pattern ist gerade bei größeren Projekten beliebt um eine strikte Trennung der Businesslogik von den Daten bzw. der Datenhaltung zu erreichen und eine komponentenbasierte Softwarearchitektur aufzubauen. Oben genanntes Pattern gehört hierbei zu den Struktur- und Architekturpattern. Struts übernimmt die Kontrolle des Systemflusses und ist somit die Controller Komponente die je nach Aktionen die Geschäftslogik ausführt und die Daten zurückliefert.

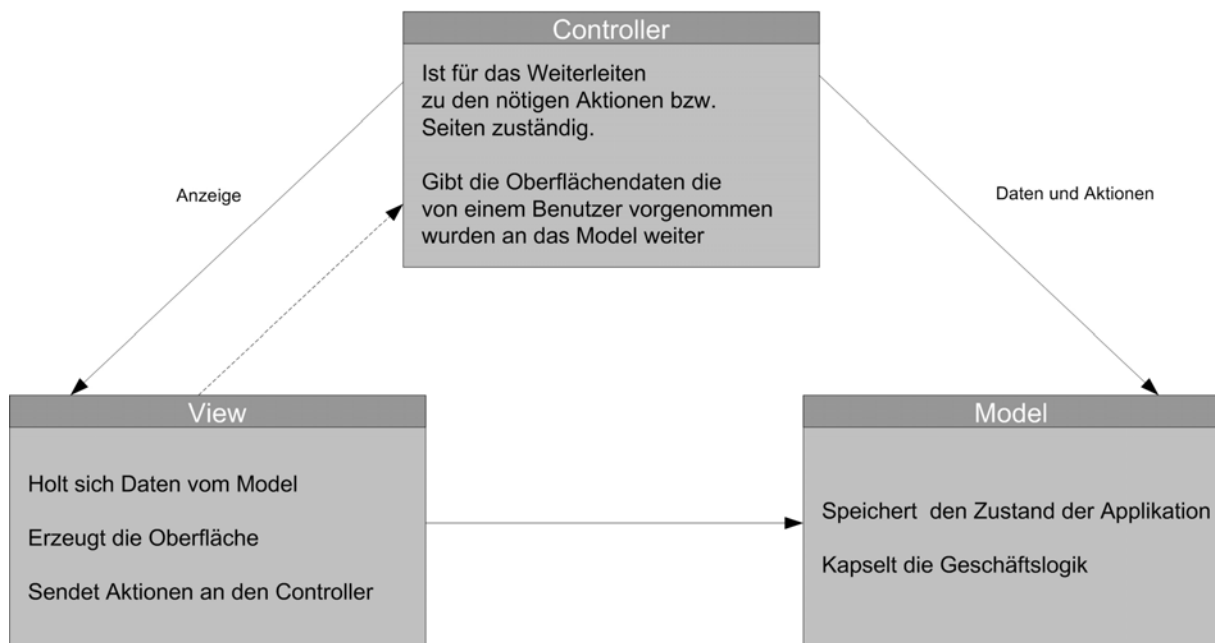


Abb. 1: Diagramm MVC Pattern

Aus dem Wunsch heraus auch für PHP eine solche Softwarearchitektur aufzubauen und besonders jetzt durch PHP5 die objektorientierten Vorteile der Sprache besser nutzen zu können ist Struts4Php entstanden. Bei der Entwicklung des Frameworks ist größtes Augenmerk auf eine möglichst nahe kommende Entwicklung zu dem Java Pendant gelegt worden. Somit ist es für Entwickler die mit Jakarta Struts schon gearbeitet haben relativ einfach eine Anwendung mit struts4php zu entwickeln.

Die eigentliche saubere Trennung der Anzeige von der Businesslogik sowie der Datenhaltung kann natürlich nur dann erreicht werden, wenn das MVC Design Pattern verstanden wurde. struts4php bietet hierfür nur eine Umgebung bzw. einen Ansatz um dieses Ziel zu erreichen. Letztendlich sollte sich aber jeder Entwickler selbst Gedanken machen wie er bestimmte Probleme, wie z.B. einen Datenbankzugriff und das daraus resultierende zurückgeben der Daten an eine Action oder einen View, löst. Hierbei denke ich gilt der Leitspruch „Viele Wege führen nach Rom“.

Der Steuermann

Wie aus Abb. 1 ersichtlich übernimmt die Controller Komponente das Steuern der Aktionen. Die Frage die sich hier natürlich stellt ist folgende: Woher weis der Controller wie er auf bestimmte Aktionen reagieren muss?

Hierfür gibt es die Konfigurationsdatei struts-config.xml die vom Controller in seine interne Struktur - das ActionMapping - eingelesen wird. Diese enthält alle wichtigen Information zum Steuern der Anwendung.

Die struts-config.xml besteht dabei aus den vier folgenden Hauptknoten:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE struts-config SYSTEM "http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
<struts-config>
    <data-sources/>
    <form-beans>
    <global-forwards>
    <action-mappings>
</struts-config>
```

Listing 1

Das Data-Source Element hat in struts4php keine Verwendung. Es wird nur angegeben um eine Validierung mittels der struts-config.dtd des Jakarta Projektes zu ermöglichen.

Innerhalb des FormBeans Elements werden die ActionForms mit deren Namen und Typ registriert.

```
<form-bean name="personSearchForm" type="PersonSearchForm" />
```

Listing 2

In den GlobalForwards können globale Weiterleitungen z. B. zu Fehlerseiten oder Systemmeldungen angelegt werden. Dabei wird *name* als Schlüssel für den Aufruf im Framework und *path* entweder als Pfad zu einer Datei oder zu einer passenden Action verwendet.

```
<forward name="systemerror" path="systemerror.tpl.html" />
```

Listing 3

Der größte Teil der struts-config.xml besteht aus dem Element action-mappings welches die einzelnen Action Definitionen enthält. Ein Block für eine Action sieht folgendermaßen aus:

```
<action path="/person_detail" name="PersonForm" type="PersonDetailAction"
scope="request" input="/person_overview" parameter="" validate="false">
    <forward name="Success" path="person.tpl.html" redirect="no" />
    <forward name="Failure" path="person_overview.tpl.html" redirect="no"/>
</action>
```

Listing 4

Der *path* definiert hierbei wie die Action erreichbar ist. Unter *name* wird der Name des ActionForms angegeben. Dieses muss natürlich im FormBeans Element registriert worden sein. Sollte für die Action kein Form benötigt werden so kann das Feld leer gelassen werden. Der *type* gibt den Typ der Action, welcher der Klassenname ist, an. *Scope* gibt an wo das ActionForm gespeichert wird. Dies kann entweder in der Session oder im Request sein. Das Attribute *input* gibt den Namen der aufrufenden Action oder Datei an. Bei einem Validierungsfehler wird auf diese zurückgesprungen. Wird *validate* auf false gesetzt so wird die Validate Methode im ActionForm nicht aufgerufen und die Daten werden nicht validiert. Das Action Element besteht nun nur noch aus beliebig vielen Forwards. In der Regel sollten aber die Standard Forwards Success und Failure ausreichen.

Was läuft

Nahezu jeder Entwickler internetbasierter Software stand in seiner Karriere schon einmal vor dem Problem, die Übersicht über seine oder die von seinem Team entwickelte Software verloren zu haben. Wie kommt's? Nun ja, sicherlich können hierbei verschiedene Gründe eine Rolle spielen, oft jedoch trägt eine mangelnde Struktur sowie eine schnelle Änderung der Anforderungen im Laufe des Projektes zur Konfusion bei. Um diesem Problem entgegenzuwirken bietet sich der Einsatz eines Frameworks wie struts4php an. Dieses zwingt den Entwickler dazu sich bereits im Vorfeld zu überlegen welche Abläufe seine Software abdecken soll, wie er diese in seiner Software abbilden kann.

struts4php arbeitet wie sein großer Bruder aus der Java Welt mit Actions, die, wie das zugrunde liegende MVC Pattern bereits ausdrückt, die Applikation steuern und beispielsweise Daten über eine Persistenzschicht wie PEAR_DB aus der Datenbank holen und an die Präsentationsschicht, z. B. Smarty, für die Weiterverarbeitung übergeben. Die eigentliche Controllerklasse wird lediglich mit dem Pfad zur Konfigurationsdatei instanziiert. Anschließend wird die Methode process() mit den Daten des Requests, der auch in Form einer einfachen Zeichenkette, die Anweisung für die interne Verarbeitung des Controllers enthält, als Parameter aufgerufen. Alles andere übernimmt der Controller selbst.

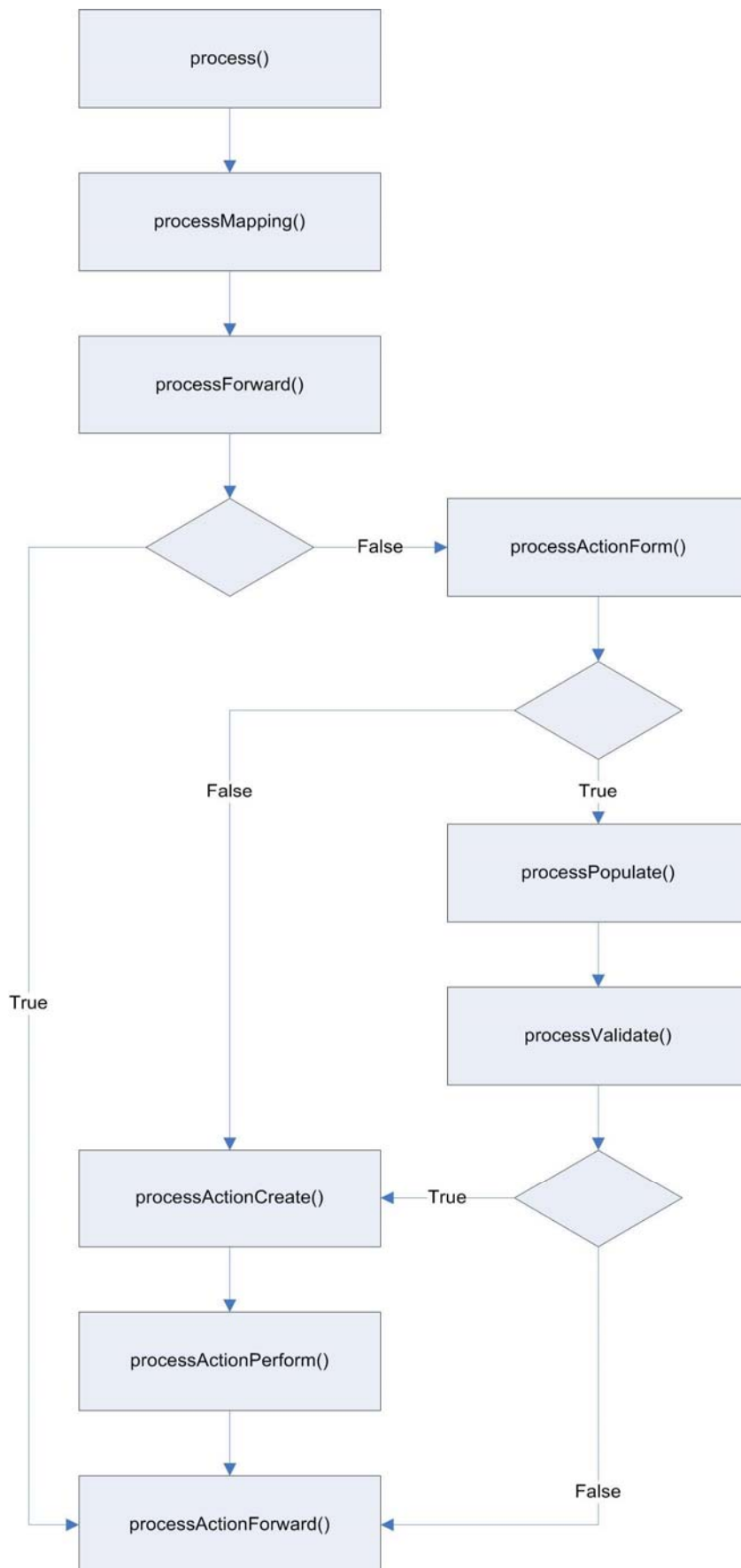


Abb. 2: Ablaufdiagramm der internen Prozesse von struts4php

- Im ersten Schritt wird die Konfigurationsdatei überprüft und eingelesen. Innerhalb des Controllers wird die XML Struktur mit den für die Verarbeitung notwendigen Informationen in eine Klassenstruktur umgewandelt, so dass auf die entsprechenden Parameter und Attribute bequem über Methoden zugegriffen werden kann.
- In nächsten Schritt wird überprüft, ob der per Request übergebene Schlüssel für die aufzurufende Action in der Klassenstruktur gefunden wurde. Ist dies der Fall, was normalerweise immer zutreffen sollte, wird ein ActionMapping Objekt instanziiert, das alle notwendigen Informationen für die weitere Verarbeitung enthält. Hierzu gehören neben dem Namen der aufzurufenden Action der Name des zugeordneten ActionForms sowie die möglichen ActionForwards.
- Wurde der aufgerufenen Action über die Konfigurationsdatei ein ActionForm zugeordnet so wird dieses automatisch mit den durch den Benutzer eingegebenen Daten aus dem Request gefüllt. Hierbei muss beachtet werden, dass die Felder im HTML Formular gleich den Membervariablen des ActionForms sind, da die Initialisierung der Werte per Reflection durchgeführt wird.
- Wurde die automatische Validierung für diese Action aktiviert, wird durch den Controller automatisch die validate() Methode (Listing 6) des initialisierten ActionForms aufgerufen. In diese kann der Entwickler die Überprüfung der vom Benutzer eingegebenen Daten vornehmen. Treten hierbei Fehler auf, so werden diese automatisch im Request registriert und das Framework gibt den als input Parameter zur aktuellen Action hinterlegten Wert zurück. Dies ist im Normalfall das Formular in das der Benutzer seine Werte eingegeben hat. Hier können nun die Fehlermeldungen ausgegeben werden.
- Im nächsten Schritt erstellt der Controller eine Instanz der in der Konfigurationsdatei angegebenen Action Klasse und ruft deren perform() Methode (Listing 5) auf. Diese Methode hat als Parameter das aktuelle ActionMapping, eine Referenz auf das initialisierte und validierte ActionForm sowie auf den aktuellen Request. Mit den darin enthaltenen Informationen kann der Entwickler nun den entsprechenden Geschäftsvorfall, z. B. das Holen von Daten aus der Datenbank und das anschließende setzen im ActionForm (Listing 5), implementieren.
- Je nach Konfiguration gibt die process() Methode des Controllers dann den Pfad zum aufzurufenden Smarty Template zurück, oder arbeitet die nächste Action ab.
- Über diesen Mechanismus kann die Applikation in beliebig kleine Teile zerlegt werden. Was die Wartung, Pflege und Erweiterung erheblich vereinfacht. Was der Entwickler innerhalb einer Action an Anwendungslogik integriert bleibt seinem Geschmack überlassen. Man sollte jedoch berücksichtigen, dass bei zu großen Actions einer der Vorteile des Frameworks, nämlich die Übersichtlichkeit, schnell wieder verloren gehen kann.

Woher nehmen wenn nicht stehlen

Da stellt sich natürlich die Frage: Wie komme ich an möglichst einfach an struts4php und wie integriere ich es in meine Applikation? Zum einen besteht über die URL

<http://www.struts4php.org/pear/struts4php-current.tgz> die Möglichkeit das Framework über den PEAR Installer das Framework in das PEAR Verzeichnis zu installieren. Diese Möglichkeit stellt den einfacheren Weg dar. Zum anderen können die Sourcen direkt über die URL

<http://www.sourceforge.net/projects/struts4php> heruntergeladen und in das eigenen Projekt integriert werden. Die Datei struts4php.php inkludiert hierfür alle notwendigen Dateien und muss vor der Instanziierung des Controllers in das Skript eingebunden werden.

Mehr Action

Entwickler die bisher nur wenig oder noch gar nicht in Kontakt mit einem Framework wie Struts gekommen sind, werden sich vielleicht fragen: Wofür den ganzen Aufwand. Nun ja, die Antwort auf

diese Frage ist nicht einfach. Je nach Art und Größe des Projektes überwiegen die Vor- oder die Nachteile. Faktoren die für den Einsatz eines Frameworks sprechen sind jedoch neben einer hohen Skalierbarkeit die saubere Trennung von Oberfläche und Anwendungslogik, was auch unerfahrenen Programmierern oder Webdesignern den Einstieg und die effektive Mitarbeit an einem Projekt erleichtert. Zusätzliche Vorteile bieten beim Einsatz eines Build Tools wie z. B. Phing automatisierbare Abläufe wie die automatische Generierung von Datenbankzugriffsklassen.

Nach einer kurzen Einarbeitungsphase kann der versierte Entwickler mit der Implementierung der im Systemkonzept definierten Geschäftsvorfälle beginnen. Als sinnvoll hat sich erwiesen, die Actions, die die eigentliche Funktionalität der Anwendung beinhalten, möglichst klein zu halten. Hiervon ausgehend, sind somit für die Verwaltung eines BusinessObjectes in vielen Fällen fünf Actions notwendig. Ein kurzes Beispiel soll aufzeigen, wie eine Action zum Laden und Anzeigen eines Datensatzes aufgebaut sein könnte.

```
public function perform($actionMapping, &$actionForm, &$request) {
    define("SUCCESS", "Success");
    define("FAILURE", "Failure");
    try {
        $person = $this->db->getPersonViewData(
            $request->getAttribute("personId"));
        $actionForm->setLastname($person->getLastname());
        $actionForm->setFirstname($person->getFirstname());
        $actionForm->setCity($person->getCity());
        return($actionMapping->findForward(SUCCESS));
    } catch(DatabaseException $de) {
        $message = new ActionMessage("failure", "An error occurred.");
        $this->addActionMessages($message, $request);
        return($actionMapping->findForward(FAILURE));
    }
}
```

Listing 5

Eine Action wird immer von der Klasse Action abgeleitet. Jede Action muss mindestens die perform() Methode implementieren. Diese Methode wird automatisch durch den Controller im Laufe der Verarbeitung aufgerufen. Als Parameter werden jeweils eine Instanz des aktuellen ActionMappings, des aktuellen ActionForms und des HTTPRequests übergeben. Das ActionMapping spiegelt die Konfigurationsdaten, die in der Konfigurationsdatei struts-config.xml für diese Action definiert wurden, wieder. Die Instanz des ActionForms wurde mit den Werten des letzten Requests initialisiert, d. h. wenn ein Benutzer im zugehörigen Formular Werte eingegeben hat, so werden diese automatisch durch das Framework im ActionForm gesetzt. Der Entwickler hat somit eine einfache Möglichkeit auf die vom Benutzer eingegebenen Werte zuzugreifen.

Im Beispiel (Listing 5) werden über eine Datenbankschicht *\$this->db* Daten aus der Datenbank geladen. Um den zu ladenden Datensatz identifizieren zu können wird die im Request enthaltene Id, der vom Benutzer im Formular ausgewählten Person, an die Methode übergeben. Die zurückgegebenen Daten werden im ActionForm gesetzt, welches wiederum im View aus dem

Request/Session geholt und weiterverarbeitet werden kann. Die Instanz des HTTPRequests dient somit hauptsächlich als Container zum Austausch der Daten zwischen Controller und View.

Validierung

Die einfache Überprüfung der Benutzereingaben stellt neben den oben genannten Vorteilen einen weiteren Pluspunkt für die Verwendung des Frameworks dar. Die Validierung erfolgt über ActionForm Klassen die in der Konfigurationsdatei zu jeder Action als Parameter mitgegeben werden können. Für jedes Feld im Formular muss eine gleichnamige Klassenvariable inklusive Getter- und Settermethoden angelegt werden. Per Reflection werden die nach jedem Request übermittelten, vom Benutzer eingegebenen Werte, im ActionForm gesetzt dieses wiederum an die entsprechende Action Klasse per Referenz übergeben. Dort können die Daten dann weiterverarbeitet werden.

Ebenfalls in der Konfigurationsdatei wird definiert, ob die validate() Methode (Listing 6), die jede ActionForm Klasse implementieren muss, automatisch durch den Controller aufgerufen wird. Wird der Parameter validate="true" gesetzt, so wird nach dem Initialisieren des ActionForms automatisch durch den Controller die validate() Methode (Listing 6) aufgerufen. In dieser Methode kann der Entwickler die Eingaben des Benutzers validieren. Entsprechen die Daten nicht den dort vorgegebenen Kriterien, so wird für jeden Fehler ein neues ActionError Objekt erzeugt und dem ActionErrors Container hinzugefügt. Der Container wird anschließend zurückgeben und automatisch durch das Framework unter dem Key *ACTION_ERRORS* im Request registriert. Enthält dieser ActionError Objekte, so bricht der Controller die Verarbeitung ab und gibt den in der Konfigurationsdatei als input Parameter spezifizierten Pfad zurück. Die Action wird dann nicht mehr initialisiert und aufgerufen. Jeder Action kann immer nur ein ActionForm zugewiesen werden, was die Flexibilität des Frameworks leider etwas einschränkt.

```
class PersonForm {
    var $name = null;
    function setName($name) {
        $this->name = $name;
    }
    function getName() {
        return $this->name;
    }
    function validate($actionMapping, &$request) {
        $errors = new ActionErrors();
        if(empty($this->name)) {
            $errors->add(new ActionError("name",
                "Bitte geben Sie Ihren Namen ein!"));
        }
        return $errors;
    }
}
```

Listing 6

Datenspeicher

Neben der Funktionalität zum Validieren der Benutzereingaben bieten die ActionForms die Möglichkeit die vom Benutzer getätigten Eingaben nicht nur während eines Request zu speichern, sondern auch während einer Session. Voraussetzung hierfür ist ein aktiviertes Session Handling. Um diese u. U. sehr nützliche Funktion zu verwenden, muss lediglich in der Konfigurationsdatei der scope

Parameter auf „session“ anstatt „request“ gesetzt werden. Beim ersten Aufruf des Formulars wird dieses nicht im Request, sondern im Session Container abgelegt. Auf diesen kann man entweder komfortabel über das HttpSession Objekt, das aus dem HttpRequest mit der Funktion getSession() geholt werden kann, zugreifen, oder über die globale PHP Variable \$_SESSION. Der Schlüssel unter dem das Form abgelegt wurde ist der Name des ActionForms der über die Konfigurationsdatei definiert wird.

Wohin führt der Weg

Wie teilt man dem Framework nun mit, wie es nach dem erfolgreichen Abarbeiten einer Action weitergehen soll. Hierzu wird dem ActionMapping, das der Action ebenfalls als Referenz übergeben wurde, mittels der Methode findForward() anhand des übergebenen Parameters mitgeteilt, welche Action oder welches Formular als nächstes aufzurufen ist. Soll auf eine weitere Action weitergeleitet werden, so muss der übergebene Parameter in der Konfigurationsdatei ebenfalls als Action angelegt worden sein. Wird als Parameter ein Wert angegeben, der durch das Framework nicht in der Konfigurationsdatei gefunden wird, so wird dieser als Rückgabewert der Methode perform() zurückgegeben und kann, z. B. durch Smarty weiterverarbeitet werden.

Über diesen Mechanismus, auch Action-Chaining genannt, können beliebig viele Actions nacheinander aufgerufen werden. Da hierbei das Framework für jede weitere Action erneut durchlaufen wird, sollte man sich aus Performancegründen gut überlegen, wo der Einsatz Sinn macht und wo nicht. Weiterhin ist zu bedenken, dass das ActionForm für jede Action erneut mit den Werten des Requests initialisiert wird, und Werte aus der vorherigen Action somit verloren gehen.

Die Ausgabe

Nach der Abarbeitung aller Actions müssen die gewonnenen Daten im Normalfall irgendwie ausgegeben werden. Hierzu bietet sich geradezu die bekannte Template Engine Smarty an. Die Integration wird über die Konfigurationsdatei von struts4php vorgenommen. Hierzu muss lediglich als Forward der Name des aufzurufenden Smarty Templates angegeben werden. Das Framework gibt diesen als Rückgabewert der process() Methode zurück. Smarty kann das Template dann bequem weiterverarbeiten.

Üblicherweise werden die aus der Datenbank geholten Werte, wie in Listing 1 gezeigt im Request gespeichert. Im Template können diese entweder direkt mit {\$smarty.request.KEY} wieder herausgeholt werden, oder, was den wesentlich eleganteren Weg darstellt, über Smarty Plug-Ins.

```
function smarty_function_html_text_tag($params, &$smarty) {
    extract($params);
    global $request;
    global $actionBanana;
    $actionMapping = $actionBanana->findMapping($request->getAttribute("path"));
    $actionForm = $request->getAttribute($actionMapping->getName());
    $method = "get" . $field . "()";
    $value = $actionForm->{$method};
    $html_result =
        '<input name="'. $name. '" class="'. $class. '" type="text" value="'. $value. '">';
    return $html_result;
}
```

Listing 7

Durch die Verwendung von Smarty Plug-Ins werden die Templates nahezu frei von Anwendungslogik gehalten, was zu einem einfacheren Handling für den HTML Entwickler führt.

Zu beachten ist lediglich, dass weder die Controller noch die Request Instanz in der Plug-In Funktion zur Verfügung stehen und diese deshalb über das global Keyword bekannt gemacht werden müssen. Danach kann über die Funktionen des Controllers das aktuelle ActionForm aus dem Request geholt, und dessen Daten, z. B. in einem Textfeld, ausgegeben werden.

und nun?

Der Einsatz von struts4php bringt neben den in diesem Artikel aufgezählten Vorteilen, wie nahezu alles im Leben, selbstverständlich auch Nachteile mit sich. Neben der Grundsatzfrage ob der Einsatz eines Frameworks in einem Projekt überhaupt notwendig ist, kommt ein stark erhöhter Konfigurationsaufwand. Dieser könnte durch die Entwicklung von Zusatztools erheblich gesenkt werden. Beispielsweise könnte man sich, wie aus der Java Welt bekannt, einen Generator vorstellen, der auf Basis von Tags in den Actions die Konfigurationsdatei automatisch generiert.

struts4php wurde, wie der Name schon sagt für PHP entwickelt. Da nun schon seit geraumer Zeit PHP5 mit seinen erweiterten OO Eigenschaften bereit steht, ist die Umstellung ein aktuelles Thema. Im Rahmen dessen würde sich ein Neudesign anbieten, das u. U. die Möglichkeit zur Integration eigener Plug-Ins beinhalten könnte.

Stefan Willkommer (s.willkommer@techdivision.com) ist Geschäftsführer der Internetagentur TechDivision aus Roseneheim, Tim Wagner ist ehemaliger Mitarbeiter der TechDivision GmbH. Beide sind maßgeblich in die Entwicklung und die Weiterentwicklung von struts4php involviert.

Links & Literatur

- [1] struts4php: <http://www.sourceforge.net/projects/struts4php>
- [2] struts4php Homepage: <http://www.struts4php.org>
- [3] PEAR The PHP Extension and Application Repository: <http://pear.php.net>
- [4] Smarty Template Engine: <http://smarty.php.net>
- [5] The Apache Struts Web Application Framework: <http://struts.apache.org>
- [6] Phing Build System: <http://phing.info>

Quelle:

PHPMagazin 2.05

Seitenangabe:

21 - 24