MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY

Renovace MVC frameworků

DIPLOMOVÁ PRÁCE

Bc. Jan Vondrouš

Brno, jaro 2008

Prohlášení

Prohlašuji, že tato diplomová práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: RNDr. Jan Pavlovič

Poděkování

Rád bych poděkoval především mému vedoucímu práce, RNDr. Janu Pavlovičovi, za poskytování cenných rad a vstřícný přístup.

Shrnutí

Tato diplomová práce analyzuje možnosti modernizace MVC frameworků. Nejprve obecně vysvětluje pojem MVC a zabývá se výhodami MVC frameworků, následně je zde prezentován obecný způsob modernizace MVC frameworku. Dále se práce zaměřuje na konkrétní MVC frameworky a ukazuje postup modernizace frameworku Struts 1 na Struts 2. Součástí práce je i případová studie převodu konkrétní aplikace z frameworku Struts 1 do frameworku Struts 2.

Klíčová slova

MVC framework, Struts, Struts 2, modernizace, Kalkulátor, převod frameworku

Obsah

Kapitola 1

Úvod

Podoba internetových stránek prošla během krátké doby velkým vývojem. Zpočátku byly internetové stránky pouze statické a umožňovaly tak svým čtenářům pouze jednotně zobrazit svůj obsah. Postupem času se však internetové stránky stávaly stále více a více interaktivní a vedle statických stránek začaly vznikat i složité a rozsáhlé webové aplikace. Vývoj a údržba těchto webových aplikací se tedy stávala stále častější záležitostí a postupem času se vypracovaly ověřené postupy, jak tyto webové aplikace vytvářet.

Pro vývoj webových aplikací se v praxi osvědčilo využít architektury MVC (Model-View-Controller), jejíž použití rozděluje aplikaci do tří základních částí a usnadňuje tak vývoj a následnou správu aplikace. Zároveň stále stoupaly požadavky na to, aby se webové aplikace vytvářely co nejrychleji a nejlevněji. V té době se začaly vytvářet MVC frameworky, které usnadňují použití MVC architektury a navíc poskytují další funkce usnadňující vývoj webové aplikace, například validaci vstupních dat nebo mechanismy pro zobrazování chybových zpráv. Při tvorbě webových aplikací se stalo téměř pravidlem využití některého MVC frameworku. Postupem času vzniklo mnoho různých MVC frameworků a ty tak prošly svým vlastním vývojem. Dnešní MVC frameworky jsou založeny na zkušenostech s používáním starších frameworků a ještě více tak zlehčují vývoj a údržbu webových aplikací. Mnoho aplikací však dodnes používá starší webové frameworky.

Cílem práce je shrnout obecně možnosti, jimiž lze v aplikaci provést modernizaci MVC frameworku. A dále pak na konkrétních příkladech tento převod prezentovat.

První kapitola je věnována vysvětlení architektury MVC. Dále jsou zde uvedeny výhody a nevýhody spjaté s použitím MVC frameworků. V této kapitole je čerpáno především z [2] a [15].

Druhá kapitola je pak věnována modernizaci MVC frameworku bez vazby na konkrétní frameworky. Jsou zde uvedeny obecné důvody, které mohou vést k modernizaci, spolu s nevýhodami, jež jsou s ní spojené. Dále jsou zde uvedeny obecné postupy, jimiž lze modernizaci provést, současně s uvedením výhod a nevýhod každého postupu. Na závěr je uveden obecný postup, kterým lze provést modernizaci aplikace používající požadavkově orientovaný MVC framework.

Ve třetí kapitole je uveden stručný přehled nejběžnějších MVC frameworků společně se srovnáním frameworků Struts 1 a Struts 2. Část této kapitoly je také věnována tomu, jak často se jednotlivé frameworky používají. Jako hlavní zdroje byly použity [17], [13], [16] a [8].

Čtvrtá kapitola se poměrně podrobně věnuje konkrétním ukázkám jednotlivých případů

převodu aplikace používající framework Struts 1 na aplikaci používající framework Struts 2. Je zde popsán převod klíčových částí, jako jsou konfigurační soubory, akce, JSP stránky. Na místech, kde je to možné, je rovněž zmíněno, jakým způsobem lze využít stávající kód. V kapitole se lze rovněž dočíst o některých dalších možnostech frameworku Struts 2, které lze při převodu využít. Pro sestavení této kapitoly byly využity především následující zdroje: [9], [17], [7], [10] a [11].

Obecnou analýzu z předchozí části doplňuje poslední kapitola, která se věnuje převodu konkrétní aplikace z frameworku Struts 1 do Struts 2. Formou případové studie, která je realizována na aplikaci Kalkulátor energetické náročnosti technologií pro čistění kontaminovaných médií, je tak předvedena konkrétní ukázka modernizace.

Kapitola 2

MVC

MVC (Model-View-Controller), někdy též nazývaný Model 2, je architektonický vzor pro vývoj aplikací. První implementaci vytvořil Trygve Reenskaug roku 1978.

Zjednodušeně MVC slouží pro rozdělení aplikace do tří různých vrstev. První vrstvou je tzv. Model, který představuje logiku aplikace (business logic). Další vrstva, Pohled, představuje uživatelské rozhraní a zajišťuje interakci s uživatelem. Řadič pak spojuje vrstvy Model a Pohled. V následujícím textu jsou jednotlivé vrstvy vysvětleny o něco podrobněji.

- **Model (Model)**

  Představuje logiku aplikace. Tato vrstva v podstatě zajišťuje základní činnost aplikace. Umožňuje přístup k uloženým datům a zároveň představuje výkonnou část aplikace. Bývá nejsložitější vrstvou aplikace a často obsahuje další logické členění. Celá tato vrstva bývá přístupná přes nadefinované rozhraní.

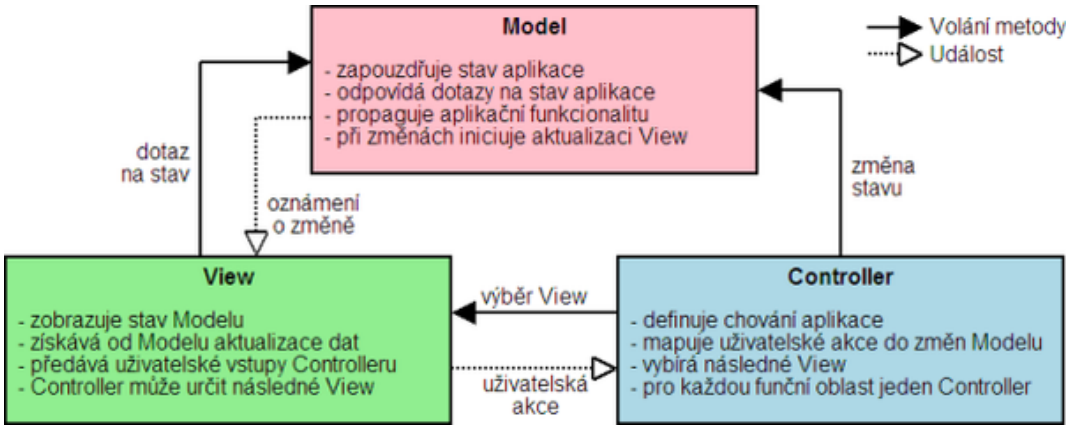
- **Pohled (View)**

  Představuje uživatelské rozhraní aplikace. Je to vrstva zodpovědná za získávání dat od uživatele a za zobrazení dat (představovaných modelem) vhodnou formou uživateli. Pro jeden model je možné vytvořit více různých pohledů, které potom slouží pro odlišnou prezentaci stejných dat různým uživatelům nebo pro odlišnou prezentaci dat na různých zařízeních. Pokud se stav modelu změní, pak se pohled aktualizuje. Existují dvě možnosti, jak pohled může získávat data od modelu:

  - **Pull**, při němž pohled sám získává data od modelu, vždy když chce zjistit aktuální stav modelu.
  - **Push**, při němž se pohled zaregistruje u modelu a model potom při jakékoli změně stavu automaticky pohled na tuto změnu upozorňuje.

- **Řadič (Controller)**

  Je umístěn mezi vrstvami model a pohled. Jeho úkolem je reagovat na události přicházející z vrstvy pohledu. Například reaguje tak, že zajistí aktualizaci dat v modelu a podle výsledku této akce pak vybere správný pohled pro další zobrazení.



Obrázek 2.1: Vztahy mezi vrstvami MVC modelu (zdroj: [14])

2.1 Výhody MVC frameworků

Obecně lze říci, že každý framework nabízí určitou předprogramovanou funkčnost, kterou lze při vývoji webové aplikace využít. Výhodou je, že na funkce, které framework nabízí, se lze většinou spolehnout, neboť framework je již použit ve spoustě jiných aplikací. Je proto již dobře otestován a z toho důvodu je i mnohem spolehlivější než jakýkoli vlastní kód. Použitím frameworku lze tedy snadno zvýšit spolehlivost aplikace.

Další výhodou je, že framework je v podstatě částečně hotová aplikace. Již samotným použitím frameworku tak získáme část funkční aplikace.

Oddělení prezenční vrstvy od datové u MVC frameworků umožňuje snadnější změny v prezenční vrstvě, protože ta je díky tomu mnohem jednodušší, tím i přehlednější a snáze udržovatelná. Toto oddělení je vhodné především z toho důvodu, že ve většině aplikací dochází k nejvíce změnám právě v prezenční vrstvě.

Ve spoustě případů je velkou výhodou to, že takovéto rozdělení aplikace do vrstev s přesně definovaným rozhraním umožní snadný paralelní vývoj jednotlivých vrstev.

Toto rozdělení také umožňuje vyměnit implementaci jedné vrstvy za jinou, a to bez vlivu na ostatní vrstvy. Je možné dokonce vyměnit vrstvu modelu, ačkoli nejčastěji se využívá možnosti změnit vrstvu pohledu.

Vrstva pohledu nemusí být pouze vyměněna za jinou, ale je zde možnost využívat více různých pohledů, které pracují s jedním modelem. Data získaná z modelu může každý pohled prezentovat uživateli odlišným způsobem v závislosti na tom, o kterého uživatele se jedná nebo na jakém zařízení s aplikací pracuje.

V aplikaci lze nadefinovat jasné a ucelené rozhraní pro práci s modelem. Díky tomu ho mohou ostatní vrstvy snadno používat a manipulovat sním bez znalosti jeho vnitřní struktury. Toto usnadní práci především vývojářům prezenční vrstvy.

Použití MVC architektury zjednoduší rozšiřování aplikace. Vrstvu pohledu a řadiče lze totiž postupně zvětšovat podle toho, jak se zvětšuje vrstva modelu.

Ve zkratce lze říci, že použití MVC frameworku umožňuje rozdělit aplikaci do jednotlivých modulů. Toto rozdělení je výhodou i z hlediska obecných požadavků na vývoj webové aplikace. Díky němu se dosáhne celkového zjednodušení vývoje a tím i snížení počtu chyb v aplikaci. Celkově se tedy zvýší spolehlivost aplikace. Zjednodušený vývoj s sebou navíc nese snížení nákladů, což je také velice příznivé. Modulárnost celé aplikace potom dále přinese možnost paralelního vývoje, čímž se zvýší jeho rychlost. Zároveň bude dosaženo znovupoužitelnosti jednotlivých částí aplikace, a tím se zjednoduší i rozšiřování a upravování aplikace. Stejně tak bude zjednodušeno i její udržování a oprava chyb.

Jak je patrno z předchozího odstavce, použití MVC frameworku přináší zlepšení většiny vlastností, na které je obecně kladen důraz při vývoji webové aplikace. Z toho důvodu jsou MVC frameworky při vývoji aplikací velmi často využívány.

2.2 Nevýhody MVC frameworků

I přes nesporné výhody lze najít i nevýhody spjaté s použitím MVC frameworku:

- Před použitím Frameworku je nutné se jej nejprve naučit. Jak obtížné je naučit se framework používat, záleží na konkrétním frameworku. Některé frameworky mohou představovat velké usnadnění při vývoji webové aplikace, ale může být obtížné je zvládnout. Jiné frameworky mohou být snadno použitelné, ale při vývoji aplikace nemusí představovat tak velké usnadnění.

- Protože použitý framework se stává součástí aplikace, vyvstává zde i nutnost přidat k aplikaci knihovny potřebné pro běh frameworku.

- Na vývoji aplikace se pak může podílet jen osoba obeznámená s konkrétním frameworkem. Různých MVC frameworků je poměrně velké množství a každý z nich má své výhody i nevýhody. Proto se v praxi můžeme setkat s různými frameworky a často je třeba se je neustále doučovat.

- Konkrétní framework je skupina tříd, abstraktních tříd a rozhraní, které potom musíme v aplikaci rozšiřovat a implementovat, čímž vzniká poměrně silná vazba aplikace na konkrétní framework. Novější frameworky se ale většinou snaží tuto závislost cíleně minimalizovat.

- Framework často vytváří mnoho pomocných objektů a to může mít negativní vliv na výkon aplikace.

Architektura MVC nijak neřeší otázku implementace vrstvy modelu, to znamená, že MVC se nijak nezabývá například otázkou, jak jsou data aplikace ukládána. Některé MVC frameworky sice poskytují řešení i těchto otázek, ale to je již nad rámec architektury MVC.

2.3 Typy MVC Frameworků

Podle úrovně abstrakce nad HTTP protokolem se MVC Frameworky dělí do dvou základních typů:

- **Požadavkově orientované (Request based)**

  Používají řadič (controller) a akce, kterými přímo zpracovávají příchozí požadavek (request). Každý požadavek je pak v podstatě bezstavový. Tento typ komunikace je poměrně blízko HTTP protokolu.

  Příkladem takovýchto frameworků jsou například Struts 1, Struts 2 a Stripes.

- **Komponentově orientované (Component based)**

  Tyto odstiňují od samotného zpracování požadavku a zapouzdřují logiku aplikace do znovupoužitelných komponent, které mohou být dokonce nezávislé na webovém rozhraní. Stav je automaticky spravován frameworkem. Je zjišťován pomocí dat obsažených v instancích komponent. Vývoj webového rozhraní pomocí těchto frameworků má potom podobné rysy jako vývoj desktopových aplikací.

  Komponentově orientované frameworky nejsou tak staré jako požadavkově orientované frameworky a jejich širší používání je záležitostí pouze posledních několika let. Příkladem takovýchto frameworků jsou například JSF, Tapestry, Wicket.

Kapitola 3

Modernizace MVC frameworku

Tato kapitola se zabývá modernizací MVC frameworku obecně a nezaměřuje se na žádné specifické vlastnosti konkrétních frameworků. Jsou zde zmíněny obecné výhody a nevýhody spojené s modernizací. Dále jsou zde rozebrány obecné způsoby, jak modernizace může probíhat. Současně je zde také uveden obecný postup modernizace požadavkově orientovaného MVC frameworku a některé typické problémy, které mohou při modernizaci nastat.

Je však nutné upozornit, že modernizace MVC frameworku záleží především na tom, jaký konkrétní framework aplikace používá a na jaký je modernizována.

3.1 Důvody pro modernizaci MVC frameworku

Větší vývoj a používání MVC frameworků pro vývoj webových aplikací je záležitostí především posledních deseti let. Novější frameworky jsou vytvářeny na základě starších frameworků, a díky tomu se vyvarovávají některých chyb v návrhu, jichž se dopouštěli jejich předchůdci. Díky tomu většinou novější framework umožňuje snazší a rychlejší vývoj webové aplikace.

Dalším důvodem hovořícím pro modernizaci může být fakt, že novější frameworky často nabízí podporu nových technologií, které lze využít při vývoji aplikace. Mohou to být takové technologie, se kterými starší framework na začátku vývoje vůbec nepočítal nebo v době jeho vzniku dokonce ještě neexistovaly. Někdy jsou do staršího frameworku tyto technologie dodatečně přidány, avšak mohou s tím být spojeny další problémy. Stává se například, že se tyto nové technologie obtížně používají (právě z toho důvodu, že se s nimi při návrhu nepočítalo). Jako příklad takových technologií mohou být například javové anotace nebo AJAX.

Jedním z důvodů změny frameworku mohou být mimo jiné problémy s výkonem. Každý framework při svém běhu spotřebovává systémové prostředky tím, že vytváří pomocné objekty a provádí další režii spojenou s jeho během. Každý framework lze nakonfigurovat tak, aby se tato režie snížila, avšak u některých frameworků lze dosáhnout většího výkonu než u jiných.

Mohou existovat aplikace, u nichž je třeba dosáhnout vysoké úrovně spolehlivosti, ale při tom zároveň ponechat možnost relativně častých úprav kódu. V tom případě může být hodně přínosné usnadnění při testování. To může být dalším důvodem pro přechod na nový framework. Novější frameworky totiž většinou usnadňují psaní testovacích tříd.

Jak již bylo zmíněno v minulém odstavci, v některých aplikacích může docházet k častějším úpravám. U starších frameworků často i poměrně jednoduché změny v aplikaci vyžadují úpravu kódu v mnoha různých souborech. U novějších frameworků bývá provádění změn jednodušší, není většinou nutné editovat tolik různých souborů a navíc se změny provádějí jednodušeji.

Existují také open-source aplikace, na jejichž vývoji se podílí velká komunita vývojářů. U těchto aplikací je vhodné, když používají framework, který ovládá co největší skupina lidí. Tím se zjednoduší hledání dalších spolupracovníků podílejících se na vývoji takového projektu. I z tohoto důvodu může být výhodné přepsat aplikaci do nového frameworku.

V jiném případě může být kladen důraz na to, aby framework, který aplikace používá, byl co nejjednodušší a co nejrychleji zvládnutelný, tak aby se noví členové vývojového týmu mohli rychle zapojit do aktivního vývoje. To může být vhodné u projektů, na kterých se podílejí například studenti nebo jiní vývojáři, u nichž nelze předpokládat znalost nějakého konkrétního frameworku.

Zcela jiným důvodem k modernizaci může být například situace, kdy vývoj a podpora stávajícího frameworku byla, nebo v blízké době bude, ukončena. Pak z dlouhodobého hlediska může být mnohem vhodnější přepsat aplikaci do nového frameworku, jehož podpora a vývoj budou dál pokračovat.

Na závěr je vhodné podotknout, že pro modernizaci většinou nehovoří pouze jeden z uvedených důvodů. Při modernizaci frameworku se většinou zlepší aplikace v několika různých ohledech.

3.2 Nevýhody modernizace

Často výhody, které přinese nový framework nevyváží problémy a náklady spojené s jeho modernizací. Kompletní přechod na nový framework většinou znamená velké změny v aplikaci. Pokud má být navíc aplikace spolehlivá, pak si změna frameworku vyžádá i velké náklady spojené s testováním aplikace, neboť změna zasáhne do velké části aplikace.

Další nevýhodou může být to, že je nutné se nový framework před modernizací naučit, popřípadě přeškolit zaměstnance na jeho používání. S tím mohou souviset vysoké náklady.

Pokud je navíc starý framework v aplikaci používán již delší dobu, všichni vývojáři většinou práci s ním dobře zvládají. Nový framework tak může zpočátku zapříčinit snížení efektivity práce.

Dalším rizikem je, že při modernizaci frameworku se do aplikace mohou zanést nové chyby, které při převodu téměř s jistotou vzniknou.

Při modernizaci hodně záleží na tom, jaký konkrétní framework aplikace používá a na jaký se má modernizovat. Obecně lze však říci, že modernizace frameworku není příliš triviální, a pokud jí není věnována dostatečná pozornost a není provedena dostatečně dobře, pak se může výrazně zhoršit kvalita celé aplikace.

3.3 Způsoby modernizace

Před samotnou modernizací je třeba zvážit, zda se tato modernizace opravdu vyplatí. Podle konkrétního případu se lze potom rozhodnout pro jednu z následujících možností.

- **V aplikaci se nadále ponechá stávající framework.**

  K tomuto rozhodnutí je vhodné se přiklonit, pokud:

  - Stávající aplikace funguje víceméně bezproblémově.
  - Od aplikace se neočekává její další rozsáhlejší rozšiřování funkčnosti.
  - Aplikace je příliš rozsáhlá, její modernizace by byla nepřijatelně nákladná a nevyplatí se použití dvou frameworků současně. To je například v případě, kdy se do aplikace přidává nová funkčnost, ale zároveň se často mění i stávající funkčnost (používání dvou různých frameworků potom zbytečně komplikuje provádění změn).
  - Je nedostatek pracovníků ovládajících nový framework.

- **Nově přidávaná funkčnost se vytváří v novém frameworku a stávající část se ponechá beze změny.**

  To je vhodné pokud:

  - Nebudou prováděny rozsáhlejší změny ve stávající části aplikace.
  - Funkčnost aplikace se bude v budoucnu významně rozšiřovat.
  - Aplikace umožňuje provozování dvou frameworků současně.

  V této variantě se nám zjednoduší přidávání nových funkcí (v případě, že nejsou nevhodným způsobem svázány s funkcemi předešlými). Zároveň ušetříme úsilí a náklady spojené s modernizací stávajících funkcí.

  Pokud tedy z nějakých důvodů nevadí současné provozování dvou frameworků, má tento způsob řadu pozitivních vlastností. Nenese s sebou totiž náklady a problémy spojené s přepisováním stávajících částí aplikace. Tyto části mohou zůstat nezměněny a používat dál starý framework. Nové funkce využívající nový framework je navíc možné přidávat do aplikace ihned.

  Velkou výhodou je také to, že v budoucnu se lze kdykoliv rozhodnout a přepsat některou část nebo postupně i celou aplikaci tak, aby používala nový framework.

  Nevýhodou je, že současným provozem dvou frameworků dojde prakticky k rozdělení aplikace na dvě téměř nezávislé části. To potom může do určité míry zkomplikovat zprávu celé aplikace. Navíc mohou existovat aplikace, u nichž takové rozdělení může být značně nevhodné.

- **Do aplikace se přidá nový framework a postupně se do něj budou přepisovat i stávající části aplikace.**

  Toto je ve většině případů pravděpodobně nejvhodnější varianta. Umožňuje rovnou vyvíjet nové části aplikace v novém frameworku, zatímco stávající části se přepisují do nového frameworku postupně podle stupně důležitosti a momentální potřeby.

  Při tomto postupu se modernizují pouze malé části aplikace, čímž se samotný převod stává jednodušším. Výhodou zde je, že provozování dvou MVC frameworků současně je pouze dočasná záležitost a výsledkem je aplikace kompletně napsaná v novém frameworku.

  Tento způsob modernizace se však může zkomplikovat v případě, že stávající aplikace není čistě implementována. To potom následně zapříčiní komplikace při přechodu na nový framework.

- **Celá aplikace se znovu vytvoří pomocí nového frameworku.**

  Při přepisování aplikace je vždy vhodné zvážit, zda nepoužít nový framework. Pokud je přepis nutný, zavedení nového frameworku nepřinese prakticky žádné další náklady navíc, naopak použití nového vhodnějšího frameworku je může snížit.

  Důvody hovořící pro znovuvytvoření aplikace mohou být následující:

  – Aplikace je špatně navržena. To může zabraňovat dalšímu rozšiřování nebo správě aplikace.
  – Aplikace je napsána velmi nečistým způsobem, což znemožňuje její další spravování, popřípadě rozšiřování. Navíc v takovém případě je téměř nemožné provést modernizaci stávajícího kódu.
  – Pracovníků ovládajících starý framework je nedostatek nebo je jejich zaškolování příliš drahé.
  – Stávající aplikace vykazuje přílišnou chybovost.

  Nevýhodou je, že cena tohoto postupu je stejná jako cena vytvoření celé aplikace. I to však může být levnější, než dlouhodobé udržování stávající aplikace.

  Na druhou stranu je toto jediný způsob, kdy se lze obejít i bez znalosti starého frameworku, což může být v některých případech velmi podstatným důvodem hovořícím ve prospěch této varianty.

Ještě je vhodné zmínit, že řešení, kdy v jedné aplikaci kombinujeme dva MVC frameworky, může být vhodné pouze v případě, kdy upravujeme stávající aplikaci. V žádném případě bych toto řešení nedoporučoval v případě vytváření nové aplikace. Důvodem je, že MVC frameworky nabízejí většinou velmi obdobné funkce, takže jejich kombinace nepřinese téměř žádné možnosti navíc, ale jen aplikaci zbytečně zkomplikuje.

3.4 Obecná strategie modernizace frameworku

Na obrázku 3.1 je pomocí UML diagramu aktivit znázorněna obecná strategie převodu požadavkově orientovaného frameworku. V následující části je pak celý postup podrobněji popsán.

1. Nejprve je třeba opravdu důkladně zvážit, zda se přechod na nový framework vyplatí, nebo zda je lepší zůstat u stávajícího frameworku.

2. S prvním bodem úzce souvisí i druhý bod, a sice výběr konkrétního frameworku, na nějž se bude aplikace převádět.

3. Následuje přidání nového frameworku do stávající aplikace.

4. Podle zvolené modernizační strategie se rozhodne, zda tímto bodem modernizace skončí (a přejde se k bodu 7), nebo zda se bude pokračovat v převádění stávajících částí aplikace.

5. Další body spočívají v převodu nejmenší smysluplné části, kterou lze v požadavkově orientovaném frameworku převést, a sice převod jedné zvolené akce. To většinou spočívá v následujících krocích:

   - Přepsání třídy reprezentující akci.
   - Zapsání akce do konfiguračního souboru nového frameworku.
   - Přepsání pohledů, v kterých se tato akce používá. Tato část je ve většině aplikací realizována pomocí JSP stránek nebo pomocí obdobných technologií.
   - Převod dalších věcí souvisejících s akcí. V aplikaci založené na Struts 1 to může být například ActionForm.

6. Následuje test převedené části aplikace, oprava chyb a převod další části aplikace (podle bodu 4).

7. Nakonec proběhne celkový test aplikace.

V některých případech může být výhodnější k bodu 5 přistupovat trochu odlišným způsobem, a sice nevolit pro převod jednu akci, ale zvolit vždy jednu stránku pohledu, kterou kompletně převedeme. U původního postupu je výhoda, že se převede kompletní jedna akce a její správná funkčnost se otestuje ve všech situacích. K této akci se pak už není nutné později vracet. Na druhou stranu je nevýhoda, že musíme nalézt všechny stránky pohledu, kde se tato akce používá. V průběhu převodu pak může vznikat spousta stran pohledu, které využívají starý i nový framework současně.

Pokud se převod realizuje po jednotlivých stránkách vrstvy pohledu, získáváme vždy stránky kompletně převedené na nový framework. Na druhou stranu se při převodu tímto

způsobem může stát, že se zapomene, které akce již byly převáděny, a budou převedeny vícekrát. Další a ještě podstatnější nevýhodou je, že stránka pohledu, kterou převádíme, nemusí využívat veškerou funkčnost, kterou akce nabízí. Tím se může stát, že akci nepřevedeme kompletně a při převodu další stránky pohledu nezbyde nic jiného, než ji do akce dodatečně dodat. To dá pak více práce, než převést celou akci najednou.

3.5 Typické problémy modernizace

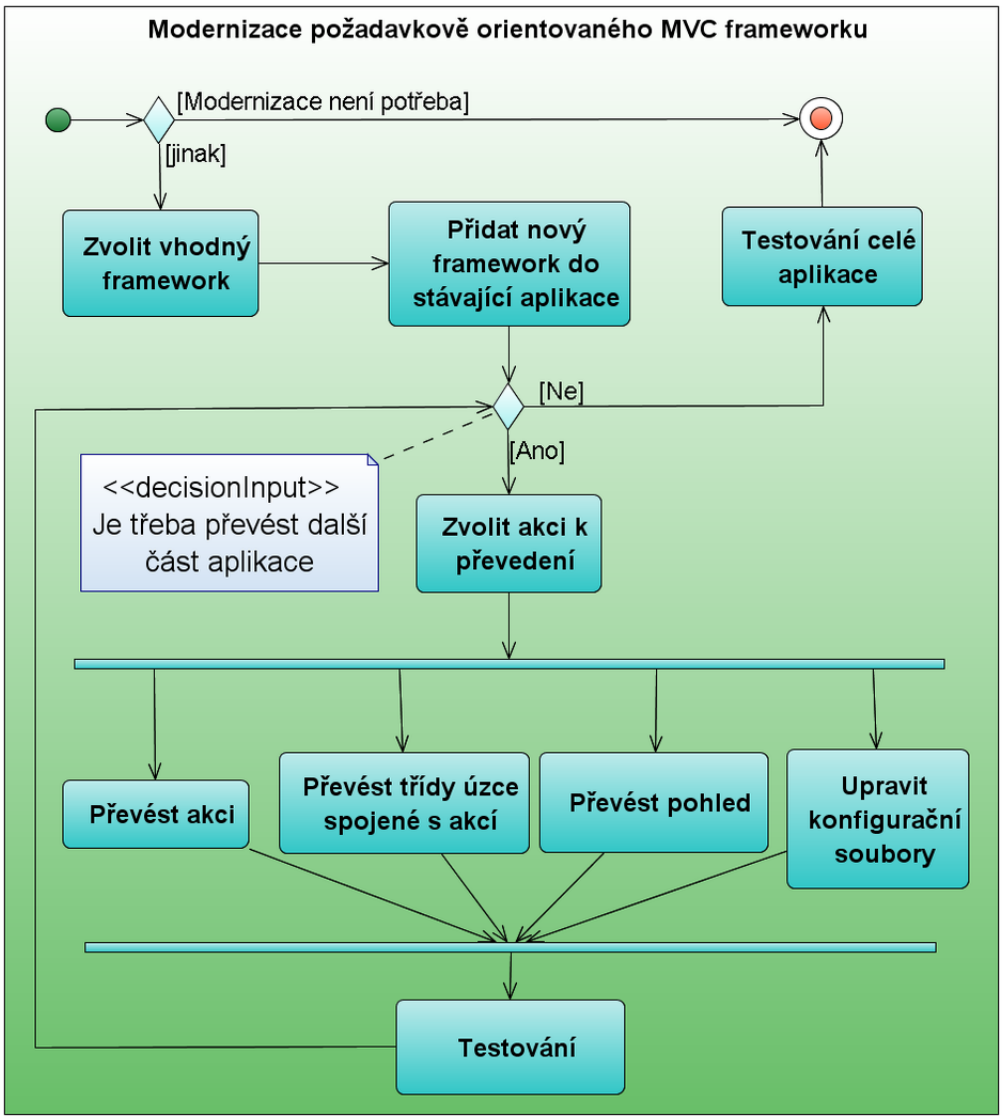
Převod aplikace se značně komplikuje, pokud aplikace používá třídy závislé na frameworku v místech, kde nemají co dělat. Například: Framework může mít objekty, které slouží pro zprávu chyb. Problém při převodu nastává, pokud aplikace tyto objekty předává jako parametry hluboko do jádra aplikace. Tím vzniká zbytečná závislost vnitřních tříd aplikace na třídách frameworku. Pokud se potom aplikace převádí na nový framework, je nutné měnit třídy, které nemají s frameworkem nic společného. Těmto třídám se musí přidat další metody, ale nemůžou se odstranit metody staré, protože ty stále používá dosud nepřepsaná část aplikace. Tím dochází k duplikaci kódu. Pokud je takovýchto případů v aplikaci víc, začíná být přepisování aplikace zmatené a namáhavé. Navíc se při převodu musíme seznamovat i s objekty z jiných vrstev aplikace, což převod výrazně zpomalí.

U některých frameworků může nastat problém v případě pokusu o jejich současné použití v jedné aplikaci. Důvodem může být to, že oba frameworky používají stejnou JAR knihovnu, avšak v různých verzích. Jeden framework nedokáže fungovat se starší verzí této knihovny a potřebuje novější, druhý to pak vyžaduje přesně naopak. Podle [3] může být příkladem takovýchto dvou frameworků Struts a Cocoon. Naštěstí u velké většiny frameworků se s tímto problémem nesetkáme. V případě, že je tento problém nutno bezpodmínečně vyřešit, nabízejí se následující varianty.

- Přepsat jeden z frameworků tak, aby používal verzi knihovny kompatibilní s druhým frameworkem. To však může být někdy obtížné, a navíc potom není možné používat nové verze přepsaného frameworku bez jejich předešlé úpravy.
- Zajistit, aby každý framework používal jinou verzi knihovny. Toho lze docílit tím, že každý framework necháme běžet na jiném virtuálním stroji. Další možností je zajistit, aby každý framework používal jiný classloader. Toho lze na většině servletových kontejnerů docílit tím, že necháme každou aplikaci běžet v jiném kontextu.

  Takovéto řešení je dostačující, ale pouze v případě, že není potřeba, aby spolu frameworky komunikovaly. Pokud je nutné zajistit komunikaci mezi frameworky, musí se do aplikace přidat ještě další prostředky, které potřebnou komunikaci umožní.

V některých aplikacích může být přechod na nový framework relativně jednoduchý, ale může mu bránit přílišný rozsah stávající aplikace.



Obrázek 3.1: Modernizace požadavkově orientovaného MVC frameworku

Kapitola 4

Konkrétní MVC frameworky

4.1 Nejběžnější MVC frameworky

4.1.1 Struts 1

Vývoj frameworku Struts 1 začal v květnu 2000 a první verze vyšla v červnu 2001. Od té doby se Struts 1 stal jednoznačně nejpopulárnějším frameworkem a dodnes se o něm hovoří „de facto“ jako o standardu pro vývoj javových webových aplikací. Dnes spočívají jeho hlavní přednosti ve velké členské základně, dobré dokumentaci a dobré podpoře softwarových nástrojů, usnadňujících vývoj aplikací s tímto frameworkem. Většina pozdějších frameworků se jím inspirovala a snažila se opravit některé jeho nedostatky.

4.1.2 Struts 2

Historie frameworku Struts 2 je o něco složitější, než je tomu u jiných frameworků. V květnu roku 2002 vyšel framework WebWork. Od začátku bylo jeho cílem především vylepšení stávajícího frameworku Struts 1 současně s integrací nejlepších nápadů z jiných frameworků. V prosinci roku 2005 bylo oficiálně oznámeno, že frameworky WebWork a Struts se spojí a vytvoří tak nový framework Struts 2. Jako důvod ke spojení těchto frameworků bylo uvedeno, že je to dobrý krok k dalšímu vývoji frameworku Struts a že WebWork již implementoval téměř vše, co bylo v plánu implementovat do frameworku Struts. První verze frameworku Struts 2 vyšly na konci roku 2006. Více podrobností o tomto frameworku bude řečeno později.

4.1.3 Java Server Faces

Java Server Faces (JSF) patří mezi komponentově orientované frameworky. Tvorba aplikací pomocí JSF v sobě kombinuje jednak prvky klasických MVC frameworků a jednak typické prvky známé z vývoje desktopových aplikací.

Velmi stručně se dá funkčnost JSF popsat následujícím způsobem. Uživatelské rozhraní tvoří JSP stránky. Každá stránka obsahuje ovládací prvky, takzvané JSF komponenty (to mohou být například formuláře nebo tlačítka). Komponenty se do sebe mohou zanořovat, čímž vzniká strom komponent. Data z komponent jsou potom ukládána do JavaBeanů. Pokaždé, když uživatel udělá nějakou akci, jako například kliknutí na tlačítko, vyvolá tím událost, která je zaslána na server. Hlavní podobnost s programováním desktopových aplikací je

právě v tom, že na komponentách můžeme zaregistrovat posluchače událostí (listenery), kteří jsou vyvoláni v případě, že tato událost nastane.

4.1.4 Tapestry

Je komponentově orientovaný framework. Tapestry nabízí velké množství předem hotových komponent a snaží se o vysokou znovupoužitelnost kódu. Programování je v podstatě čistě na objektové úrovni a framework uživatele odstiňuje od nízkoúrovňových věcí, jako je zpracování požadavku.

Tapestry je poměrně velký a komplexní framework (větší než je například Struts 1). Programování v něm je velmi efektivní, ale jen v případě, že jej ovládáme. Výhodou je, že umožňuje dobrou spolupráci programátorů s webdesignéry. Jeho hlavní slabinou je právě to, že je obtížné se jej naučit.

4.1.5 Spring MVC

Spring je rozsáhlý framework a jedna z jeho částí je právě implementace MVC frameworku nazývaná Spring MVC. Framework Spring se nezabývá zdaleka jenom webovou vrstvou, ale i ostatními částmi aplikace. Spring MVC vznikl jako reakce na nedostatky tehdejších frameworků. Především tedy frameworku Struts 1.

Spring MVC je požadavkově založený framework, a tak není zcela vhodný pro aplikace, kde se vyplatí používat komponentově založené frameworky. Definuje řadu rozhraní, která jsou zodpovědná za to, co musí MVC framework poskytovat. Tato rozhraní jsou dostatečně jednoduchá na to, aby uživatel v případě zájmu mohl vytvořit svoji vlastní implementaci.

4.1.6 Stripes

Framework, který se snaží o to, aby byl jednoduchý a přímočarý. Snaží se o minimalizaci konfiguračních souborů a k tomu využívá princip „convention over configuration“. To znamená, že pojmenování se řídí podle předem určených konvencí, díky kterým pak není potřeba další konfigurace. Samozřejmě se Stripes zcela bez konfigurace neobejde. Konfiguraci však nezapisuje do konfiguračních souborů, ale využívá se anotací, které přinesla Java 5.0.

Stripes se soustřeďuje na to, aby bylo snadné jej začít rychle používat i bez zdlouhavého počátečního učení. K tomu napomáhá i jeho dobrá dokumentace. Dalším kladem je, že přidává do aplikace minimum závislostí a snaží se být snadno rozšiřitelný. Hodí se zejména v aplikacích, kde je potřeba spravovat velké formuláře. Je to především proto, že hodně usnadňuje typovou konverzi, validaci a mapování hodnot z formulářů do cílových objektů. Dalším důvodem, který usnadňuje správu velkých formulářů, je způsob návrhu frameworku, díky kterému jsou všechny věci spojené s konkrétním formulářem umístěny na jednom místě.

Stripes je však poměrně mladý framework a jeho uživatelská komunita není příliš velká. Vývoj frameworku Stripes není také tak bouřlivý, jako je tomu u ostatních MVC frameworků.

Často tomuto frameworku bývá vytýkáno, že URL adresy jsou napevno uvedeny ve třídě představující akci, což je důsledkem konfigurace pomocí anotací.

4.2 Srovnání Struts 1 a Struts 2

Struts 2 začal vznikat později a jeho předlohou byl framework Struts 1. Díky tomu se Struts 2 vyvarovává chyb, které přetrvávají ve Struts 1, a zlepšuje jeho návrh. Výhody, které Struts 2 přináší:

- Zjednodušené akce - Akce nemusí rozšiřovat žádnou třídu ani rozhraní závislé na frameworku. Ve frameworku Struts 2 může být akcí libovolný POJO objekt, který má metodu execute vracející typ String. Toto je možné díky tomu, že je zde uplatněn princip „Inversion of Control“, známý především z frameworku Spring.

- Zrušení ActionFormů - Ve Struts 2 k získávání hodnot z formuláře není nutné vytvářet speciální třídu rozšiřující rozhraní ActionForm.

- Vylepšený návrh - Ve Struts 1 je nutné rozšiřovat abstraktní třídy závislé na frameworku. Většina tříd ve Struts 2 implementuje pouze rozhraní a na rozdíl od Struts 1 lze říci, že Struts 2 je nezávislý na protokolu HTTP.

- Jazyk OGNL - Ve Struts 2 je možné pro výrazy používat jazyk OGNL (Object Graph Notation Language).

- Typová konverze - Struts 2 používá pro typovou konverzi jazyk OGNL. Díky tomu poskytuje možnosti typové konverze pro základní primitivní a objektové datové typy.

- Jednodušší testování - Díky lepšímu návrhu lze akce jednodušeji testovat i bez použití mock objektů.

- Lepší nastavení implicitních hodnot - Struts 2 má lépe nastaveny implicitní hodnoty parametrů. Díky tomu ve velkém množství případů vyhovuje implicitně nastavená hodnota a není třeba jí uvádět. To zjednodušuje a zpřehledňuje výsledný kód.

- Vylepšené značky - Za to, jaký kód vygeneruje značka frameworku Struts 2, zodpovídá šablona spojená s touto značkou. Šablony lze snadno upravovat, navíc samotný Struts 2 již obsahuje několik různých šablon. Díky tomu lze opakující se kód přesunout z JSP stránek do definice šablony a tím se zjednoduší a zpřehlední vzhled JSP stránek.

- Anotace - Jedna z novinek jazyka Java 5 byly anotace. Struts 2 umožňuje využívat anotace namísto psaní konfigurace do XML nebo properties souborů. Ve Struts 2 lze využít anotace například při validaci.

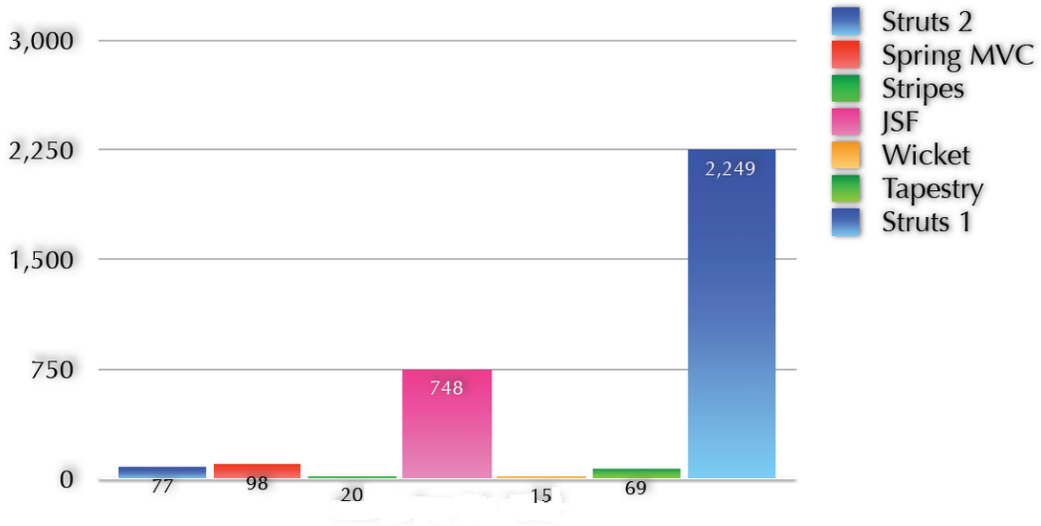
- Zjednodušená tvorba doplňků (pluginů) - Doplněk pro Struts 2 aplikaci lze udělat pouhým vytvořením JAR archivu, který připojíme k aplikaci. Není potřeba žádná další konfigurace, stačí pouze přidat nebo odebrat JAR archiv. To je možné díky domluvené struktuře JAR archivu.
- Jednoduchá integrace do Springu - K použití akcí ve frameworku Spring stačí pouze přidat JavaBeany frameworku Spring.
- AJAX - Struts 2 obsahuje značky používající AJAX, které umožňují: validaci dat na straně klienta, vzdálené potvrzení formuláře, obnovení části HTML stránky, možnost vzdáleného nahrání a spuštění javascriptového kódu, AJAXové záložky a automatické doplňování.
- Zlepšení názvů - Struts 2 používá ve většině případů jednodušší, kratší a výstižnější názvy, které napomáhají snazšímu zvládnutí frameworku a přehlednosti kódu. Na druhou stranu dělají přechod od frameworku Struts 1 trochu složitější.

Oproti Struts 1 je Struts 2 lepší téměř ve všech ohledech. Lze však nalézt i některé nevýhody. Hlavní nevýhodou je především menší členská základna. Další nevýhodou je i menší počet softwarových nástrojů. V neposlední řadě je pak frameworku Struts 2 vytýkána také nepříliš kvalitní dokumentace.

4.3 Populárnost frameworků

Framework Struts 1 v době svého vzniku zcela překonal všechny dosud existující MVC frameworky. Díky tomu se velmi rychle stal nejpoužívanějším frameworkem a je již považován de facto za standard. Ačkoli dnes již existují lepší frameworky, které se pomalu začínají prosazovat, stále je dnes Struts 1 nejpoužívanějším frameworkem. Důvody jsou především v tom, že ve Struts 1 je dnes napsána většina aplikací a že má největší členskou komunitu. Pro Struts 1 existuje nejvíce softwarových nástrojů usnadňujících vývoj a existuje k němu také množství kvalitní dokumentace.

V posledních letech se však při vývoji aplikací uplatňuje stále více framework JSF. JSF byl prohlášen za standard a stal se součástí specifikace Java 5 Enterprise Edition, což velmi pomáhá jeho rozšíření. Společně s tím vzniká pro JSF mnoho softwarových nástrojů.



Obrázek 4.1: Počet pracovních míst na serveru dice.com 14. listopadu 2007 (zdroj: [8])

Kapitola 5

Modernizace aplikace ze Struts 1 na Struts 2

Tato kapitola nejprve srovnává frameworky Struts 1 a Struts 2 a potom na konkrétních příkladech ukazuje, jak provést převod aplikace používající framework Struts 1 na aplikaci používající framework Struts 2. Jsou zde postupně rozebrány příklady toho, jak převést jednotlivé části aplikace.

Všude, kde je zmíněn framework Struts 1, myslí se tím jeho verze Struts 1.3.8 a všude, kde je zmíněn Struts 2, je míněna jeho verze 2.0.11. Avšak téměř vše, co je zde uvedeno, platí i v předchozích verzích těchto frameworků a je značně pravděpodobné, že nic z toho se nezmění ani v následujících verzích.

Proč je však demonstrován právě převod aplikace z frameworku Struts 1 do Struts 2? Zvolení frameworku Struts 1 je poměrně jednoznačná volba, a to především z toho důvodu, že se jedná o nejrozšířenější javový MVC framework. Většina dnešních aplikací používá pravě tento framework, navíc hodně pozdějších frameworků vychází ze Struts 1 a vylepšuje jeho nedostatky. Z toho plyne, že dnes jsou již vhodnější frameworky pro psaní aplikací, a proto může být v některých případech vhodné zamyslet se nad případnou modernizací. Dalším pádným důvodem je, že Struts 1 se již nebude dál nijak výrazně vyvíjet a v budoucnu skončí pravděpodobně i jeho podpora.

Základní důvody, proč je zvolen převod právě do frameworku Struts 2, jsou tyto:

- Stejně jako u Struts 1 i u Struts 2 se jedná o požadavkově orientovaný framework. Díky tomu lze převod realizovat bez zásadních změn v koncepci celé aplikace. U rozdílnějších frameworků by se potom již nejednalo o modernizaci stávající aplikace, ale spíše o nové vytvoření celé aplikace.
- Struts 2 vychází ze Struts 1. Vylepšuje a především výrazně zjednodušuje své použití.
- Distribuce Struts 2 obsahuje zásuvný modul umožňující použití akcí a formulářů přímo z původní Struts 1 aplikace.
- Struts 2 rovněž nabízí zásuvný modul umožňující použití JSF komponent. Tato vlastnost může být výhodou především při dalším rozšiřování aplikace.

5.1 Hlavní rozdíly mezi Struts 1 a Struts 2

Při přepisování aplikace z frameworku Struts 1 na Struts 2 je třeba vědět, v čem se tyto frameworky odlišují. Proto jsou v následujícím textu uvedeny hlavní rozdíly mezi těmito

frameworky:

- Ve Struts 1 jsou všechny akce singletony[1] a zároveň musejí být vláknově bezpečné, což někdy komplikovalo programování akcí. Ve Struts 2 je pro každý požadavek vytvářena nová instance třídy akce[2]. Díky tomu již není nutné, aby akce byly vláknově bezpečné. Tím je programování akcí o něco jednodušší.
- Ve Struts 1 třídy akcí rozšiřovaly abstraktní třídu závislou na frameworku. Akce ve Struts 2 mohou rozšiřovat rozhraní Action, popřípadě rozšiřovat třídu ActionSupport, která toto rozhraní již implementuje. Implementace rozhraní Action však není povinná a jako akce může být použita libovolná třída s metodou execute.
- Ve Struts 2 se již nepoužívá třída ActionForm. K získávání hodnot z formulářů lze použít libovolný JavaBean. Většinou tento JavaBean představuje objekt použitý pro obsluhu akce a hodnoty z formuláře jsou nastaveny přímo do proměnných tohoto objektu. Lze však použít i jiný libovolný JavaBean. Dokonce lze některé hodnoty formuláře nastavit do proměnných libovolného JavaBeanu a zbylé do proměnných objektu akce.
- Značky frameworku Struts 1 používají JSTL, z toho důvodu pro výrazy používají jazyk JSTL-EL. Struts 2 může rovněž používat JSTL, avšak podporuje především jazyk „Object Graph Notation Language“ (OGNL), který poskytuje mnohem větší možnosti.
- Pro přístup k objektům ve vrstvě pohledu používá Struts 1 standardní mechanismus JSP stránek. Naproti tomu Struts 2 používá „ValueStack“, který umožňuje přistupovat k hodnotám objektů i bez udání jejich konkrétního typu. Díky tomu lze použít jednu stránku z vrstvy pohledu k zobrazení různých proměnných se stejným jménem, ale různým typem.
- Ve Struts 1 jsou hodnoty získávané od uživatele obvykle všechny typu String. Pro typovou konverzi se používá Commons-Beanutils, konvertor je nastaven pro třídu a není možné jej měnit pro různé instance. Ve Struts 2 se pro typovou konverzi používá jazyk OGNL. Součástí Struts 2 jsou již konvertory pro primitivní a základní datové typy.
- Narozdíl od Struts 1 Struts 2 umožňuje vytvořit různý životní cyklus zvlášť pro každou akci. K tomuto účelu ve Struts 2 slouží takzvaný "Interceptor Stack", který lze nastavit pro každou akci různý.
- Validace vstupních hodnot je ve Struts 1 většinou zajištěna jedním ze dvou základních způsobů. Buď je validace naprogramována ručně v metodě validate, nebo se využívá

1. Singleton je návrhový vzor. Pokud je nějaká třída singleton, znamená to, že v jeden okamžik může existovat maximálně jedna instance této třídy.
2. Při každém požadavku servletový kontejner generuje mnoho objektů, proto vytvoření jednoho objektu navíc již nemá téměř žádný vliv na výkon.

rozšíření Commons Validator. Jedna třída může být ve Struts 1 validována různým způsobem v závislosti na tom, v jakém kontextu je použita. Pro validaci ve Struts 2 se používají rovněž dva základní způsoby. Prvním způsobem je opět ruční validace pomocí metody validate. Druhý způsob je pomocí XWork Validation Framework, ten je zdánlivě podobný řešení ze Struts 1, avšak navíc umožňuje zřetězovat jednotlivé validace.

- Akce ve Struts 1 jsou závislé na Servlet API, protože metoda získává jako své parametry objekty HttpServletRequest a HttpServletResponse. Ve Struts 2 se ve většině případů místo s těmito objekty pracuje s obyčejným objektem typu Map. V případě potřeby však Struts 2 stále umožňuje přistupovat přímo k objektům HttpServletRequest a HttpServletResponse. Pokud se však vyhneme použití těchto objektů, zjednoduší se i testování akcí. Při testování lze pak snadno nastavit vlastnosti akce. Lze ji pouze spustit a není třeba pro ni vytvářet mock objekty.

V tabulce 5.1 jsou pak uvedeny hlavní části frameworku Struts 1 a jim odpovídající části ve frameworku Struts 2. Dále je pak na obrázcích 5.1 a 5.2 znázorněno, jak probíhá zpracování požadavku ve frameworku Struts 1 a Struts 2 (zdroj: [7]).

Struts 1	**Struts 2**
ActionForm	Action nebo POJO
struts-config.xml	struts.xml
ActionServlet	FilterDispatcher
RequestProcessor	Interceptors
validation.xml	<Název akce>-validation.xml
ActionForward	String

Tabulka 5.1: Srovnání hlavních částí frameworků Struts 1 a Struts 2

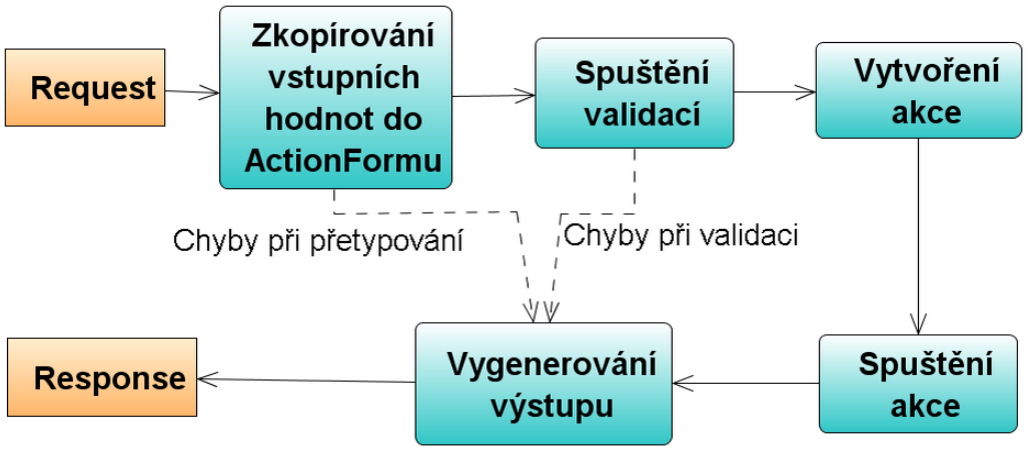
5.2 Přidání frameworku Struts 2

Prvním krokem při převodu aplikace je přidání frameworku Struts 2 do stávající aplikace. To obnáší dva základní kroky. Jednak přidání knihoven frameworku Struts 2 a jednak editaci konfiguračního souboru web.xml.

- **Přidání knihoven**

  Knihovny se nacházejí v distribuci frameworku Struts 2 v adresáři *struts-2.0.11/lib/*. Je ale pravděpodobné, že aplikace nebude muset používat všechny knihovny z tohoto adresáře. Minimální knihovny, které jsou nutné k běhu frameworku Struts 2, jsou:

  - commons-logging-1.0.4.jar
  - freemarker-2.3.8.jar



Obrázek 5.1: Životní cyklus frameworku Struts 1

- ognl-2.6.11.jar
- struts2-core-2.0.11.jar
- xwork-2.0.4.jar

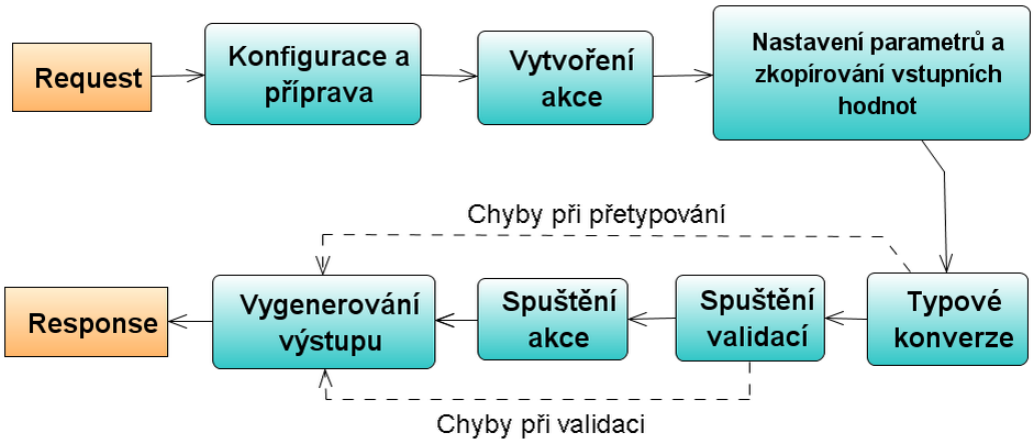
Knihovna *commons-logging-1.0.4.jar* však může být už v aplikaci obsažena, protože ji používá i framework Struts 1. Typicky se tyto knihovny v aplikaci umísťují do adresáře *WEB-INF/lib/*.

- **Editace souboru web.xml**

  Soubor web.xml se nachází v adresáři WEB-INF. V aplikaci, která používá framework Struts 1, vypadá tento soubor ve zkrácené podobě takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <!-- Místo pro vložení filtru pro Struts 2 -->

  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class
     >org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
```



Obrázek 5.2: Životní cyklus frameworku Struts 2

```
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>

</web-app>
```

Jak je vidět, Struts 1 je do aplikace vložen jako servlet. Struts 2 se do aplikace již nevkládá jako servlet, ale jako filtr. Tento filtr se přidává tak, že na místo vyznačené komentářem vložíme následující definici filtru:

```
<filter>
  <filter-name>struts2</filter-name>
  <filter-class
   >org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
</filter>

<filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Tím byl do aplikace přidán framework Struts 2, přičemž na chování aplikace se nic nezměnilo. Všechny URL končící **.do* bude nadále spravovat Struts 1 a všechny URL končící **.action* bude spravovat Struts 2. Tato přípona je implicitně nadefinována v souboru default.properties u klíče struts.action.extension. Lze jí změnit předefinováním tohoto klíče v souboru struts.properties.

5.3 Další konfigurace frameworku

5.3.1 Soubor struts.xml

Ve Struts 1 se hlavní konfigurační soubor jmenuje *struts-config.xml*. Ve Struts 2 je ekvivalentem tohoto souboru soubor s názvem *struts.xml*, který je třeba umístit do classpath aplikace. Běžně to bývá adresář *WEB-INF/classes*. Ještě je vhodné zmínit, že do struts.xml je automaticky vkládán soubor *struts-default.xml*, který je umístěn v knihovně *struts2-core-2.0.11.jar*.

5.3.2 Soubor struts.properties

Soubor struts.properties je rovněž umístěn v classpath (klasicky tedy *WEB-INF/classes*). Lze v něm nastavit mnoho parametrů, které mají vliv na chování frameworku. Pokud je nějaký parametr nastaven, přepíše se tím výchozí nastavení, které je uloženo v souboru default.properties. Tento soubor se nachází rovněž v knihovně *struts2-core-2.0.11.jar*.

5.3.3 Nastavení zdroje zpráv

Klasicky Struts 1 používá properties soubor pro zobrazování zpráv. Pro zjednodušení popisu nastavení zdroje zpráv bude uveden následující příklad: properties soubor je uložen v balíku „balik“ a jmenuje se „Texty.properties“.

Aby framework Struts 1 používal jako zdroj zpráv tento soubor, je v souboru struts-config.xml nakonfigurován následující řádek:

```
<message-resources parameter="balik.Texty" />
```

Stejné konfigurace lze ve Struts 2 dosáhnout dvěma způsoby. První možnost je nastavit properties soubor v souboru struts.properties. To se zajistí přidáním následujícího řádku:

```
struts.custom.i18n.resources=balik.Texty
```

Druhou možností je nastavit properties soubor přímo ve struts.xml. Toho docílíme přidáním následujícího kódu:

```
<constant name="struts.custom.i18n.resources"
          value="balik.Texty" />
```

Tímto nastavením lze tedy dosáhnout toho, že frameworky Struts 1 i Struts2 budou pro zobrazování zpráv používat totožný soubor (v tomto případě Texty.properties). Využití jednoho lokalizačního souboru pro oba frameworky může být obzvlášť výhodné v aplikacích, kde mají být Struts 1 i Struts 2 provozovány současně.

5.4 Jak přepsat struts-config.xml na struts.xml

Jak již bylo řečeno, Struts 1 používá soubor struts-config.xml, kterému ve Struts 2 odpovídá soubor struts.xml. Tato část se tedy zabývá problematikou převodu souboru struts-config.xml na struts.xml.

V následujícím textu je popsán způsob, jak převod realizovat. Tento postup sice nepostihuje úplně všechny možnosti a parametry, které se mohou v souboru struts-config.xml vyskytnout, ale popisuje převod nejdůležitějších značek a jejich parametrů.

Obecný postup k přepsání struts-config.xml do struts.xml tedy vypadá následovně:

1. Změna DTD dokumentu na:

```
<!DOCTYPE struts PUBLIC
 "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
 "http://struts.apache.org/dtds/struts-2.0.dtd">
```

2. Změna značky struts-config na značku struts
3. Odstranění značky form-beans i se vším, co obsahuje.[3]
4. Změna značky action-mappings na značku package s parametry name="default" a extends="struts-default"
5. Postupná změna všech značek action i s jejich obsahem podle následujícího postupu:
   1 Ve značce action se odstraní parametr name.[4]
   2 Ve značce action se změní parametr path na parametr name.[5]
   3 Ve značce action bude změněn parametr type na parametr class.[6]
   4 Značka forward se změní na result. Její parametry projdou následující úpravou:
   - parametr name zůstane beze změny
   - parametr path bude odstraněn a jeho hodnota se stane obsahem značky result
   - parametr redirect bude odstraněn. Pokud měl hodnotu true, pak se do značky result přidá parametr type="redirect". V případě, že se přesměrování provádí na jinou akci, pak se místo hodnoty redirect použije hodnota redirect-action.

   5 Ve značce action se odstraní parametr forward a jeho hodnota se zkopíruje do obsahu značky result.

Následující příklad ukazuje výše uvedený postup v konkrétní podobě, kdy struts-config.xml vypadá následovně:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts-config PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
```

3. ActionForm se ve Struts 2 normálně nepoužívá. To, co ve Struts 1 zajišťoval action form, ve Struts 2 většinou zajišťuje přímo třída představující akci.
4. Parametr name ve Struts 1 totiž definoval ActionForm.
5. To je mnohem přirozenější vzhledem k tomu, že tento parametr obsahuje jméno akce.
6. Název class je mnohem vhodnější pojmenování, protože obsahuje název třídy, která za akci zodpovídá.

```
<struts-config>
  <form-beans>
    <form-bean name="TestForm" type="balik.TestForm"/>
  </form-beans>
  <action-mappings>
    <action path="/test" type="balik.TestAction"
            name="TestForm" validate="true" scope="request">
      <forward name="vysledek" path="/vysledek.jsp" />
    </action>
  </action-mappings>
</struts-config>
```

Převedený soubor struts.xml potom tedy vypadá takto:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
 "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
 "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <package name="default" extends="struts-default" >
        <action name="test" class="balik.TestAction" >
            <result name="vysledek">/vysledek.jsp</result>
        </action>
    </package>
</struts>
```

Následující ukázky jsou příkladem převodu akce, která používá parametr forward. Akce zapsaná v konfiguračním souboru Struts 1 může vypadat takto:

```
<action path="/uvitani" forward="/uvitani.jsp"/>
```

Tuto akci je potom možno převést do konfiguračního souboru Struts 2 tímto způsobem:

```
<action name="uvitani" >
    <result>/uvitani.jsp</result>
</action>
```

Další příklad se týká akce, která používá přesměrování. V konfiguračním souboru frameworku Struts 1 je potom tato akce nadefinována například takto:

```
<action path="/presmerovani" type="balik.PresmerovaniAction" >
  <forward name="success" path="/welcomeStruts.jsp" redirect="true" />
</action>
```

Po převodu je ve frameworku Struts 2 tato akce nadefinována následujícím způsobem:

```
<action name="presmerovani" class="balik.PresmerovaniAction" >
  <result type="redirect">/welcomeStruts.jsp</result>
</action>
```

U posledního příkladu je patrné, že u značky result chybí parametr name. Vynechání parametru name je možné díky tomu, že implicitní hodnota tohoto parametru je success. Ve Struts 2 aplikacích se této implicitní hodnoty často využívá, a proto je vhodné se o ní zmínit.

V případě, že je u předešlé ukázky vyvolána akce s názvem presmerovani, je spuštěna automaticky její metoda s názvem execute. Pokud je při převádění aplikace potřeba zajistit spuštění jiné metody než metody execute, využívá se k tomu parametru method u značky action. Následující příklad definuje akci zobrazeni, která používá opět třídu balik.PresmerovaniAction. Při vyvolání akce zobrazení však není spuštěna metoda execute, ale metoda zobraz. Toho lze využít a použít tak jednu třídu pro definici více různých akcí, které se liší pouze spouštěnou metodou.

```
<action name="zobrazeni" class="balik.PresmerovaniAction" method="zobraz">
  <result>/zobrazeni.jsp</result>
</action>
```

5.5 Převod akce

Přepisování akcí je možno velmi jednoduchým způsobem demonstrovat na následujícím příkladu. Ukázková akce ve Struts 1 se zde skládá ze dvou tříd.

První třídou je action form:

```
package forms;
import org.apache.struts.action.ActionForm;
public class HelloForm extends ValidatorForm {
    private String message;
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

Druhou třídou je třída samotné akce:

```
package actions;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class HelloAction extends Action {
    public ActionForward execute(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
    throws Exception {
        HelloForm input = (HelloForm) form;
        input.setMessage(MESSAGE);
```

```
        return mapping.findForward(SUCCESS);
    }
    public static final String MESSAGE = "Hello World!";
    public static final String SUCCESS = "success";
}
```

Typicky se při převodu do Struts 2 obě tyto třídy sloučí do jediné, která může vypadat následovně:

```
package actions;
import com.opensymphony.xwork2.ActionSupport;
public class Hello extends ActionSupport {
    public String execute() throws Exception {
        setMessage(MESSAGE);
        return SUCCESS;
    }
    public static final String MESSAGE = "Hello World!";
    private String message;

    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
}
```

Takto převedená akce zajišťuje funkcionalitu původní akce a navíc i funkcionalitu action formu. Díky tomu není v metodě execute nutné uvádět již řádek:

```
HelloForm input = (HelloForm) form;
```

Další změnou je, že metoda execute nemá návratový typ ActionForward, ale pouze obyčejnou instanci třídy String (tedy textový řetězec). To psaní akce zjednodušuje a zároveň se tím zmenší závislost aplikace na třídách frameworku.

Jak je vidět, v převedené třídě není definována konstanta SUCCESS a přitom je v této třídě použita. To je možné z toho důvodu, že konstanta SUCCESS je již definována ve třídě ActionSupport, kterou tato třída rozšiřuje. Kromě konstanty SUCCESS lze ještě využít konstant ERROR, INPUT, LOGIN a NONE. Všechny tyto konstanty jsou typu String a jejich hodnota je vždy stejná jako jejich název, pouze napsaný malými písmeny.

Je dobré upozornit, že zrušení tříd action form je pro Struts 2 výhodou. Ve většině Struts 1 aplikací je většinou každá třída action form používána pouze v jedné akci. Pokud je action form používán ve více akcích, pak většinou tyto akce zajišťují velmi podobnou funkcionalitu. Při převodu je dobré všechny tyto akce sloučit do jedné třídy a v této třídě použít více různých metod (každá metoda pak zastupuje jednu starou akci). Potom stačí v konfiguračním souboru struts.xml nadefinovat různé akce používající stejnou třídu s tím rozdílem, že každá akce bude volat jinou metodu této třídy. Jak lze toto nakonfigurovat, je zmíněno

v předchozí podkapitole. V případě, že z nějakého důvodu není vhodné, aby vstupní parametry byly definovány přímo ve třídě představující akci, pak lze k tomuto účelu použít libovolný POJO (Plain Old Java Object).

Další podstatnou změnou je, že metoda execute nemá žádné parametry. Pochopitelně mohou existovat situace, kde Struts 1 pracuje běžně s objektem request nebo response. Při přepisování aplikace by obcházení použití těchto objektů mohlo být přinejmenším velmi náročné. Pokud chce akce ve frameworku Struts 2 přistupovat k objektu request, musí implementovat rozhraní ServletRequestAware. Pokud chce přistupovat k objektu response musí implementovat rozhraní ServletResponseAware. Příklad takové akce:

```
package actions;
import com.opensymphony.xwork2.ActionSupport;
import javax.servlet.http.*;
import org.apache.struts2.interceptor.*;
public class RequestAction extends ActionSupport
    implements ServletRequestAware, ServletResponseAware {
    private HttpServletRequest request;
    private HttpServletResponse response;
    public String execute() {
        if(request.getParameter("next") != null)
           return SUCCESS; else return ERROR;
    }
    public void setServletRequest(HttpServletRequest request) {
        this.request = request;
    }
    public void setServletResponse(HttpServletResponse response) {
        this.response = response;
    }
}
```

Aby před zavoláním metody execute opravdu došlo k předání objektů request a response, musí se spouštět interceptor servletConfig. Tento interceptor se implicitně spouští, takže pokud nedojde ke změně nastavení interceptorů, vše funguje bez jakýchkoli problémů.

Obdobným způsobem může akce přistupovat k objektu session. Stačí pouze naimplementovat rozhraní SessionAware.

Ačkoli ukázkové příklady akcí ve Struts 2 vždy rozšiřovaly třídu ActionSupport, vůbec tomu tak nemusí být. Ve Struts 2 může být akcí POJO objekt. Stačí, aby měl metodu execute, která se spustí v případě, že je tato akce vyvolána. Následující kód ukazuje, jak jednoduchá může být akce ve Struts 2. Ačkoli nemá smysl takovou akci vytvářet, demonstruje to schopnost frameworku Struts 2 používat jako akci třídu, která není na API frameworku vůbec ničím závislá.

```
package struts2;
public class MinimalAction {
  public String execute(){
      return "success";
```

```
  }
}
```

5.6 Převod JSP stránek

Převod JSP stránek může být někdy poměrně obtížný. Například z toho důvodu, že značky frameworku Struts 2 generují jiný kód než značky frameworku Struts 1. To pak může mít nemalé důsledky na výsledný vzhled stránky.

Nejprve stručné porovnání značek frameworků Struts 1 a Struts 2. Společně se Struts 1 jsou dodávány 4 knihovny značek. Jsou to:

- **bean**

  Obsahuje značky užitečné pro práci s JavaBean objekty. Umožňují přístup k těmto objektům, získávání jejich vlastností i jejich vytváření.

- **html**

  Obsahuje značky používané k vytváření HTML nebo XHTML značek, především formulářů a vstupních polí. Tyto značky zprostředkovávají spojení mezi JSP stránkami a ostatními komponentami webové aplikace.

- **logic**

  V této knihovně jsou především značky umožňující podmíněné generování kódu a procházení kolekcí objektů.

- **nested**

  Tyto značky rozšiřují standardní značky Strutsu a přidávají možnost je do sebe zanořovat.

Na JSP stránce potom definice těchto knihoven vypadá následovně:

```
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<%@ taglib uri="http://struts.apache.org/tags-html" prefix="html" %>
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<%@ taglib uri="http://struts.apache.org/tags-nested" prefix="nested" %>
```

Oproti tomu Struts 2 má jen jednu knihovnu značek, jejíž definice na JSP stránce vypadá následovně:

```
<%@ taglib prefix="s" uri="/struts-tags" %>
```

Struts 1 nabízí celkem 97 různých značek, zatímco Struts 2 pouhých 58. Jeden z důvodů je, že Struts 1 obsahuje některé značky, které lze zaměnit za značky z JSTL (JavaServer Pages Standard Tag Library) knihovny, a Struts 2 tyto značky již neobsahuje. Takovéto značky se nacházejí především v knihovnách bean a logic, příkladem takové značky je logic:redicect

```
<logic:redirect page="/example.jsp" />
```

Tuto značku lze nahradit značkou c:redirect z JSTL knihovny:

```
<c:redirect url="/example.jsp" />
```

Dalším důvodem, proč má Struts 2 méně značek, je, že se značky neopakují, jako je tomu ve Struts 1 v knihovně nested. Další příčinou zmenšení počtu značek je, že Struts 2 používá jazyk OGNL, díky němuž lze například značky logic:empty, logic:equal, logic:greaterEqual, logic:greaterThan, logic:lessEqual, logic:lessThan, logic:match, logic:notEmpty, logic:notEqual, logic:notMatch nahradit pouze jednou značkou, a sice s:if.

Nyní následuje vysvětlení, jak probíhá výpis parametru nějakého beanu. Nejprve je popsán případ, kdy ve Struts 1 aplikaci je na nějaké JSP stránce vypisován parametr message z beanu s názvem ExampleForm:

```
<bean:write name="ExampleForm" property="message"/>
```

Tento kód vrátí návratovou hodnotu, kterou získá zavoláním metody getMessage na příslušném beanu. V případě, že se stejná věc provádí ve Struts 2, je provedena většinou následujícím způsobem:

```
<s:property value="message"/>
```

U značky s:property není nijak specifikováno, u jakého beanu se má metoda getMessage zavolat. Je to proto, že tato značka prochází takzvaný ValueStack, v kterém postupně hledá první bean, jenž má metodu getMessage. Může ovšem nastat případ, kdy jsou na ValueStacku dva beany a oba mají metodu getMessage. V tom případě se zavolá tato metoda u prvního z nich. Pokud chceme určit, že se má použít například bean s názvem exampleBean, pak to lze zajistit následujícím kódem:

```
<s:push value="exampleBean" >
    <s:property value="message" />
</s:push>
```

Díky této značce s:push zajistíme to, že se požadovaný exampleBean dostane na vrchol ValueStacku. Tím se stane prvním nalezeným při vypisování pomocí s:property.

V předchozím textu byl zmíněn pojem ValueStack, který zaslouží trochu více pozornosti. Jak již bylo řečeno, Struts 2 používá jazyk OGNL. ValueStack se ve Struts 2 aplikaci nazývá objekt, který Struts 2 nastaví jazyku OGNL jako kořenový. ValueStack je ve skutečnosti složen z více objektů, jazyku OGNL se ale jeví jako jeden jediný objekt. V předchozím případě se tedy hledá první objekt ValueStacku, který má metodu getMessage. Pořadí prohledávaných objektů je následující:

1. Dočasné objekty

   To jsou takové, které jsou vytvořeny na JSP stránce. Takovéto soubory vytváří například značka s:url nebo vznikají při iterovaní přes nějakou kolekci.

2. Objekty modelu

3. Objekty akce

   Přistup k datům, které obsahuje právě vyvolaná akce.

4. Pojmenované objekty

   To jsou objekty #application, #session, #request, #attr a #parameters.

5.6.1 Převod formulářů

Při převodu aplikace, která nějakým způsobem komunikuje s uživatelem, bude dozajista potřeba převádět formuláře. Základem je převod triviálního formuláře, který ve Struts 1 vypadá následovně:

```
<html:form action="formAction">
  <table>
    <tr><td><label for="name_id">Name:</label></td>
      <td><html:text property="name" styleId="name_id" /></td></tr>
    <tr><td><html:submit property="send" /></td></tr>
  </table>
</html:form>
```

Po převedení do Struts 2 by tento formulář mohl vypadat takto:

```
<s:form action="formAction">
  <s:textfield name="name" label="Name" />
  <s:submit name="send" />
</s:form>
```

Hned na první pohled je vidět, že formulář je mnohem jednodušší. Zbylo v něm pouze vstupní pole a potvrzovací tlačítko, všechen ostatní kód týkající se tabulky a popisku vygeneruje Struts 2 automaticky.

Při přepisování existující aplikace, jsou však veškeré popisky a styly utvářející vzhled stránky již vytvořeny. Prostým použitím Struts 2 značek pak téměř jistě dojde k zdeformování vzhledu stránky, protože kód, který vygenerují značky Struts 2, se bude značně lišit od kódu, který vygenerují značky frameworku Struts 1. To, co bude Struts 2 generovat, lze však lehce ovlivnit pomocí nastavení „tématu“. Pokud není definováno jinak, Struts 2 používá téma s názvem xhtml, které implicitně formátuje formulář do tabulky.

Aby Struts 2 značky vytvářely výstup, který potřebujeme v naší aplikaci, můžeme vytvořit vlastní téma nebo upravit nějaké již existující. Pokud naším cílem není zjednodušení JSP stránek, chceme zachovat maximum z existujícího kódu a nechceme se zabývat opětovným formátováním stránek, pak je vhodné použít téma s názvem simple, které generuje kód velmi podobný kódu generovaného značkami frameworku Struts 1. Téma se u značky nastavuje pomocí parametru theme. Pokud tento parametr použijeme přímo u značky s:form, pak všechny prvky uvnitř tohoto formuláře budou toto téma také automaticky používat. Pokud u některé značky chceme, aby používala jiné téma, lze to provést rovněž pomocí parametru theme přímo u konkrétní Struts 2 značky.

Pokud je příklad převeden pomocí tématu simple, pak bude vypadat následovně:

```
<s:form action="formAction" theme="simple">
  <table>
    <tr><td><label for="name_id">Name:</label></td>
     <td><s:textfield name="name" id="name_id" /></td></tr>
    <tr><td><s:submit name="send" /></td></tr>
  </table>
</s:form>
```

Na první pohled je vidět, že kód již není zdaleka tak elegantní. Když však tento kód srovnáme s původním kódem, zjistíme, že jediné, co bylo třeba změnit, byly značky frameworku Struts 1. Tento přístup umožní zachovat maximum z původního kódu, převod je díky tomu jednodušší, rychlejší a méně namáhavý. Nevýhodou však je, že nijak nezkrátíme a nezpřehledníme stávající kód JSP stránek.

Parametry značek frameworku Struts 1 a frameworku Struts 2 se poměrně liší. Proto následuje porovnání parametrů značek html:text a s:textfield. Jako první jsou uvedeny parametry, které mají stejnou funkci, ale odlišný název. Tyto parametry jsou uvedeny v tabulce 5.2.

Parametr značky html:text	**Odpovídající parametr značky s:textfield**
property	name
style	cssStyle
styleClass	cssClass
styleId	id

Tabulka 5.2: Převod parametrů u značek html:text a s:textfield

U některých parametrů neexistuje parametr, který by jim zcela odpovídal, ale je možné jej přepsat jiným způsobem. Například pokud je u značky html:text použit parametr s hodnotou titleKey="example.key", lze to převést jako parametr title u značky s:textfield, a sice následujícím způsobem: title="%{'getText('example.key')}". Dalším takovým příkladem je u značky html:text parametr bundle, který slouží k nastavení zdroje zpráv. Pokud ve Struts 2 chceme na JSP stránce pro nějakou značku nastavit alternativní zdroj zpráv, je možno to ve Struts 2 udělat pomocí značky s:i18n tímto způsobem:

```
<s:i18n name="package.AlternativeBundle">
  <s:textfield name="name"
     title="%{getText('key.from.alternativeBundle')}" />
</s:i18n>
```

Další skupinou parametrů u značky html:text jsou errorKey, errorStyle, errorStyleClass, errorStyleId. Tyto parametry slouží pro úpravu stylu zobrazení chybových hlášení. Pokud toto chceme realizovat ve Struts 2, lze to udělat pomocí šablony (template), nebo tématu (theme). Parametr alt a s ním související parametr altKey ve Struts 2 žádnou obdobu nemají.

Zbylé parametry již zůstávají stejné. Jsou to accesskey, readonly, disabled, maxlength, onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onmou-

sedown, onmousemove, onmouseout, onmouseover, onmouseup, tabindex, title, value. Parametr maxlength se ve straších verzích Struts 2 jmenoval maxLength.

Aby bylo srovnání parametrů úplné, jsou následně zmíněny ještě parametry, které jsou pouze u značky s:textfield:

- *key* Slouží k současnému nastavení parametrů name, value a label.
- *label, labelSeparator, labelposition* Slouží pro automatické vygenerování popisku. Nevýhodou je, že na JSP stránce není možné jednoduše upravit jeho vzhled. Pokud jej chceme změnit, musíme to udělat pomocí šablony.
- *onselect* Parametr z jazyka HTML, který u značky html:text chyběl.
- *required, requiredposition* Slouží k označení pole jako vyžadovaného (většinou tím, že zobrazí hvězdičku vedle tohoto pole).
- *template, templateDir, theme* Slouží k nastavení šablony, podle které se bude kód generovat.
- *tooltip, tooltipConfig, tooltipCssClass, tooltipDelay, tooltipIconPath* Slouží k zobrazení malé ikony s vysvětlujícím popisem.

Ostatní formulářové prvky mají většinu parametrů shodnou s parametry již rozebraných značek html:text a s:textfield.

Jak již bylo zmíněno, značky obou frameworků se rovněž liší. Následující text bude věnován srovnání základních značek frameworků Struts 1 a Struts 2. V tabulce 5.3 jsou značky frameworku Struts 1 z knihovny html a jim odpovídající značky frameworku Struts 2.

V tabulce je uvedeno několik značek, kterým neodpovídá žádná značka ve frameworku Struts 2. Tyto značky nejsou převáděny do značek, jež jsou součástí Struts 2, ale do běžných značek jazyka HTML.

Značky jazyka Struts 1 jsou výrazně podobné značkám jazyka HTML. Oproti tomu značky frameworku Struts 2 nabízejí více možností a často lze tak výsledný zápis podstatně zkrátit. Dobře to vystihuje následující jednoduchý příklad. Následující kód ukazuje jednoduché roletové menu vytvořené ve frameworku Struts 1:

```
<html:select property="option">
  <html:option value="1">Marie</html:option>
  <html:option value="2">Josef</html:option>
</html:select>
```

Přepis tohoto formulářového prvku do Struts 2 může vypadat takto:

```
<s:select name="option" list="%{#{'1':'Marie', '2':'Josef'}}" />
```

Hodnotu atributu list tvoří výraz jazyka OGNL.

Při přepisování aplikace se může stát, že kód, který generují značky frameworku Struts 2, nebude zcela odpovídat našim potřebám. V tom případě Struts 2 nabízí možnost předefinovat šablonu, podle které se kód generuje. Jednou z výhod Struts 2 je právě jednoduchost,

s jakou lze šablony předefinovat. Je však důležité poznamenat, že všechny šablony dodávané s distribucí Struts 2 nejsou napsány v JSP, ale v jazyku FreeMarker.

Nyní však zpět k tomu, jak lze změnit šablonu. Například změna šablony pro značku s:a v implicitním tématu xhtml se provede následovně. V aplikaci se do adresáře <Class path>/template/xhtml vytvoří soubor s názvem a.ftl. Potom vždy, když se bude vykreslovat značka s:a, bude použita dříve nadefinovaná šablona a.ftl. Většinou je zapotřebí v šabloně udělat jen malou změnu, proto je nejlepší zkopírovat do nového souboru a.ftl obsah původního a.ftl a ten pak pouze upravit podle našich potřeb.

Takovéto upravení šablony v tématu xhtml má vliv na celou aplikaci. Často však můžeme vyžadovat, aby se naše šablona uplatnila pouze v některých případech. Toho lze docílit nadefinováním nového tématu, což není o moc složitější než předefinování šablony. Například pro nadefinování nového tématu s názvem mytheme stačí požadované šablony umístit do adresáře /template/mytheme. Kdykoliv pak nastavíme u nějaké značky téma mytheme, bude tato značka generovat kód podle šablon umístěných v tomto adresáři. Navíc lze tématu mytheme nastavit, které téma rozšiřuje, a to tak, že se v adresáři /template/mytheme vytvoří soubor theme.properties, jehož obsah bude:

```
parent = simple
```

Potom pokud Struts 2 nenajde v adresáři /template/mytheme šablonu pro danou značku, použije šablonu tématu simple. Tímto způsobem lze tedy velmi snadno upravit nějaké existující téma.

5.7 Validace vstupních hodnot

Jak již bylo zmíněno, Struts 1 i Struts 2 mohou provádět validaci dat různými způsoby. Oba frameworky mohou provádět validaci pomocí metody validate. Ačkoli by se na první pohled mohlo zdát, že při převodu stačí metodu validate pouze zkopírovat, rozhodně tomu tak není. Změny, které je třeba provést, jsou názorně vidět na následujícím příkladu.

Příklad ukazuje jednoduchý formulář s jedním vstupním polem pro jméno. Validace pouze ověří, zda je jméno zadáno. Ve Struts 1 lze toto realizovat následujícím způsobem.

Konfigurace v souboru struts-config.xml vypadá takto:

```
<form-beans>
    <form-bean name="validateForm" type="struts.ValidateForm"/>
</form-beans>
<action-mappings>
    <action input="/form.jsp" path="/ValidateAction"
            name="validateForm"  type="struts.ValidateAction" >
        <forward name="success" path="/result.jsp" redirect="true" />
    </action>
</action-mappings>
```

Na JSP stránce form.jsp je uveden následující formulář:

```
<html:form action="ValidateAction" method="post" >
    <html:errors property="name" />
    Name:<html:text  property="name"/><br/>
    <html:submit />
</html:form>
```

Třída ValidateForm pak vypadá následovně:

```
package struts;
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.*;

public class ValidateForm extends ActionForm {
  private String name;
  public String getName() {  return name; }
  public void setName(String string) { name = string; }
  public ActionErrors validate(ActionMapping mapping,
                               HttpServletRequest request) {
    ActionErrors errors = new ActionErrors();
    if (getName() == null || getName().length() < 1) {
      errors.add("name", new ActionMessage("errors.required","name"));
    }
    return errors;
  }
}
```

Třída ValidateAction obsahuje následující kód:

```
package struts;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class ValidateAction extends org.apache.struts.action.Action {
  public ActionForward execute(ActionMapping mapping, ActionForm  form,
         HttpServletRequest request, HttpServletResponse response)
         throws Exception {
    return mapping.findForward("success");
  }
}
```

Přepsání této jednoduché validace do Struts 2 pak může vypadat následovně. V konfiguračním souboru struts.xml je akce definována takto:

```
<action name="ValidateAction" class="struts.ValidateAction">
    <result name="input">/form.jsp</result>
    <result>/result.jsp</result>
</action>
```

Formulář na JSP stránce form.jsp ve Struts 2 vypadá následovně:

```
<s:form method="post" >
    <s:textfield label="Name" name="name"/>
```

```
    <s:submit />
</s:form>
```

Protože je použito implicitní xhtml téma, v případě chybně zadané hodnoty se hlášení o chybě samo zobrazí vedle odpovídajícího vstupního pole. Pokud je použito téma simple, pak se tato chybová hlášení sama nezobrazují. K jejich vypsání se musí na JSP stránce uvést značka s:fielderror. Pokud je na nějakém místě třeba uvést pouze chybové hlášení pro pole se jménem name, pak toho lze docílit rovněž pomocí značky s:fielderror, a sice následujícím způsobem:

```
<s:fielderror>
    <s:param value="%{'name'}" />
</s:fielderror>
```

Ve Struts 2 se třída ValidateForm stane součástí třídy ValidateAction. Tato nová třída bude ve Struts 2 vypadat následovně:

```
package struts;
import com.opensymphony.xwork2.ActionSupport;
import java.util.ArrayList;
public class ValidateAction extends ActionSupport {
  public void validate() {
    if( name == null || name.trim().isEmpty() ) {
     addFieldError("name",getText("errors.required",new String[]{"name"}));
    }
  }
  private String name;
  public String getName() { return name; }
  public void setName(String name) { this.name = name; }
}
```

Ve Struts 1 vrací metoda validate typ ActionErrors a má čtyři parametry, oproti tomu ve Struts 2 tato metoda nevrací nic a navíc nemá žádné parametry. Díky tomu, že ve Struts 2 metoda nic nevrací, není potřeba vytvářet a vracet proměnnou errors. Pokud se zjistí, že hodnota nějakého pole nevyhovuje, nepřidává se hlášení o chybě do proměnné errors, ale pomocí metody addFieldError. Tato metoda je v rozhraní ValidationAware, které je implementováno ve třídě ActionSupport, kterou rozšiřuje naše třída ValidateAction.

Struts 2 používá několik druhů zpráv o průběhu akce. Prvním druhem je již zmíněný FieldError. Ten se používá k zobrazení chybových hlášení spojených s konkrétním vstupním polem nějakého formuláře. Druhým typem je ActionError, který slouží k zobrazení chybových hlášení spojených s akcí. K vypsání těchto chybových hlášení na JSP stránce slouží značka s:actionerror. Posledním druhem je potom ActionMessage, tento druh se používá pro šíření zpráv o průběhu akce, například o tom, že se podařilo data úspěšně uložit. K vypsání těchto zpráv na JSP stránce slouží značka s:actionmessage. Pokud je nutné uvnitř akce zjistit zda došlo k nějaké chybě, lze k tomu využít metody hasActionErrors a hasFieldErrors nebo metodu hasErrors, která zjišťuje oba dva druhy chyb. Stejně tak pro zjištění přítomnosti zpráv lze využít metodu hasActionMessages.

Při psaní Struts 2 aplikace lze využít implicitního nastavení, které se chová tak, že pokud je akce vyvolána s metodou input, back,cancel nebo browse, pak se neprovádí validace. Díky tomu lze snadno udělat odkaz z jiné části aplikace. V příkladu uvedeném v předešlém textu lze z libovolné JSP stránky odkázat na vstupní formulář následujícím způsobem:

```
<s:url id="exampleForm" method="input" action="ValidateAction" />
<s:a href="%{exampleForm}" >form</s:a>
```

Metodu input v akci ValidateAction není třeba implementovat, protože ta je již implementována ve třídě ActionSupport. Metoda vrátí řetězec input, díky kterému se zobrazí stránka /form.jsp. Protože nebyla spuštěna validace, nezobrazí se chybová hlášení o chybně zadaných hodnotách. Ta jsou zobrazena až v případě vyplnění a odeslání formuláře. Díky tomuto přístupu lze potom zaměnit nebo přejmenovat vstupní JSP stránku (v našem příkladě form.jsp). Potom stačí předefinovat název stránky pouze u definice akce v souboru struts.xml.

Druhá možnost, jak zajistit validaci ve Struts 1, je pomocí modulu Validator, který validuje podle údajů nastavených v jeho konfiguračním XML souboru. Aby se v předchozím příkladu prováděla validace pomocí tohoto modulu, vyžaduje to provedení následujících změn. Nejprve se musí tento modul nadefinovat ve struts-config.xml, to vyžaduje přidání následujícího kódu:

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
   <set-property property="pathnames"
        value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

Třída ValidateForm pak musí místo třídy ActionForm rozšiřovat třídu ValidatorForm. Metoda validate již není dále potřeba (validaci zajistí modul Validator). Další, co je třeba udělat, je přidání následující konfigurace do souboru validation.xml:

```
<formset>
  <form name="validateForm">
    <field property="name" depends="required">
      <arg0 key="validateForm.name"/>
    </field>
  </form>
</formset>
```

Toto nastavení říká, že se má validovat formulář validateFrom, který je nadefinován ve struts-config.xml, a že se má zkontrolovat hodnota uvedená v poli name. Nastavení parametru depends na hodnotu required pak znamená, že hodnota v poli name musí být zadána. Jak přesně se validace bude provádět a jak bude vypadat případná chybová zpráva, je nastaveno v souboru validator-rules.xml.

Jak již bylo zmíněno, Struts 2 standardně používá XWork Validation framework. Validace pomocí tohoto frameworku je potom ještě o něco jednodušší. Aby Struts 2 zajistil validaci, stačí vytvořit XML soubor, který bude umístěn ve stejném balíku jako třída akce, a

jeho název musí být <Jméno třídy akce>-validation.xml nebo <Jméno třídy akce>-<Název akce>-validation.xml.

Konkrétně u třídy ValidateAction to znamená v balíku pojmenovaném struts vytvořit soubor ValidateAction-validation.xml, jehož obsah bude:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC
      "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
      "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
  <field name="name">
    <field-validator type="requiredstring">
      <message key="error.required">Name is required</message>
    </field-validator>
  </field>
</validators>
```

Metoda validate už pak není ve třídě ValidateAction potřeba. O validaci se postará Struts 2. Je však potřeba zajistit, aby se společně s akcí spouštěl interceptor validation, který ale je při implicitním nastavení automaticky spouštěn. Jak musí vstupní hodnota vypadat, aby prošla validací, je určeno nastavením parametru type u značky field-validator. Nejběžnější validátory jsou ve Struts 2 již nastaveny a další lze nadefinovat v souboru validators.xml. Chybová zpráva, která je uvedena uvnitř značky message, se použije pouze v případě, že v lokalizačních souborech není nalezen klíč nastavený jako hodnota parametru key.

Ve Struts 2 lze pro validaci vstupních hodnot využít i anotací, díky kterým se navíc ušetří práce spojená s vytvářením XML souboru. Pokud má být validace zajištěna pomocí anotací, je třeba v ní udělat dvě následující změny. Nejprve je třeba třídu označit anotací Validation a přidat do ní potřebné importy. To u třídy ValidateAction vypadá následovně:

```
...
import com.opensymphony.xwork2.validator.annotations.*;

@Validation
public class ValidateAction extends ActionSupport {
...
```

Druhou věcí je přidat anotace ke všem set metodám u proměnných, které se mají validovat. Ve třídě ValidateAction se tato proměnná jmenuje name a požadujeme u ní, aby byla vyplněna. Proto anotace u její set metody vypadá následovně:

```
...
@RequiredStringValidator(type = ValidatorType.FIELD,
         message="Name Required", key="error.required")
    public void setName(String string) { name = string; }
...
```

Opět i zde platí, že to, co je nastaveno u parametru message, je použito pouze v případě, že v souborech s lokalizací není nalezen klíč uvedený jako hodnota u parametru key.

5.8 Lokalizace

Běžně jsou údaje o uživatelově lokalizaci uloženy v jeho session. Ve frameworku Struts 1 je lokalizace typicky zajištěna následujícím způsobem. Pro lokalizaci je vytvořena speciální akce, která má za úkol nastavit lokalizační údaje. V našem případě tuto akci bude představovat třída LocaleAction, která je uvedena v Příloze. Kód pro změnu lokalizace potom na JSP stránce vypadá následovně:

```
<%@ taglib prefix="html" uri="http://struts.apache.org/tags-html" %>
...
<html:link action="/Locale?language=es">Espa&#241;ol</html:link>
```

Ve frameworku Struts 2 je lokalizace značně zjednodušena. Je to především díky tomu, že již není třeba pro lokalizaci vytvářet speciální akci. Stačí na JSP stránce uvést následující kód:

```
<%@ taglib prefix="s" uri="/struts-tags" %>
...
<s:url id="es" action="Cokoli">
  <s:param name="request_locale">es</s:param>
</s:url>
<s:a href="%{es}">Espa&#241;ol</s:a>
```

Pro akci Cokoli zde nemusíme vytvářet žádnou speciální třídu. Stačí, když je akce s tímto názvem uvedena v souboru struts.xml.

5.9 Struts 1 plugin

V předchozím textu bylo rozebíráno, jak převést Struts 1 akce na akce frameworku Struts 2. Struts 2 však nabízí ještě jednu alternatvní možnost, kterou je využití zásuvného modulu se jménem Struts 1 Plugin. Tento zásuvný modul umožňuje použít třídy napsané pro Struts 1 přímo ve Struts 2 aplikaci. V následujících bodech jsou uvedeny možnosti tohoto zásuvného modulu.

Výhody tohoto zásuvného modulu:

- Umožňuje použít třídy akcí ze Struts 1 a nakonfigurovat je jako akce frameworku Struts 2.
- Zároveň umožňuje ve Struts 1 akcích používat jejich ActionFormy.
- Umožňuje validaci ActionFormů pomocí původního Commons Validator.

Nevýhody tohoto zásuvného modulu:

- Ke svému běhu vyžaduje knihovny frameworku Struts 1.

- Neumožňuje na JSP stránkách používat Struts 1 značky. To znamená, že všechny JSP stránky se musí přepsat do značek frameworku Struts 2. Uvažuje se však, že tato funkcionalita bude do tohoto zásuvného modulu doprogramována. Bohužel se nyní nedá odhadnout, kdy se tak stane. Zatím se na podpoře Struts 1 značek pracovat nezačalo.

Plugin funguje tak, že poskytuje akce a interceptory frameworku Struts 2, které jsou schopny používat Struts 1 akce tak, jak to dělá framework Struts 1. Například pokud je potřeba spustit jednoduchou akci frameworku Struts 1 s názvem RandomAction, která rozhodne, zda se má zobrazit jedna, nebo druhá strana. Konfigurace ve struts.xml pak vypadá následovně:

```
<package name="struts1" extends="struts1-default" namespace="/struts1">
  <interceptors>
      <interceptor-stack name="minS1Stack">
          <interceptor-ref name="staticParams"/>
      </interceptor-stack>
  </interceptors>
  <default-interceptor-ref name="minS1Stack"/>
  <action name="RandomAction" class="org.apache.struts2.s1.Struts1Action">
      <param name="className">struts1.RandomAction</param>
      <result name="one" >one.jsp</result>
      <result name="two" >two.jsp</result>
  </action>
</package>
```

Aby vše správně fungovalo, musí balík (package) rozšiřovat balík struts1-default. Dále je nadefinována akce RandomAction, kterou představuje třída Struts1Action. Tato třída je součástí zásuvného modulu Struts 1 Plugin a zajišťuje pouze zprostředkování komunikace mezi frameworkem Struts 2 a Struts 1 akcí RandomAction. Aby vše správně fungovalo, musí být k aplikaci přibaleny základní knihovny frameworku Struts 2, dále pak potřebné knihovny frameworku Struts 1 a zásuvný modul Struts 1 plugin, který je součástí distribuce Struts 2 a je pojmenován struts2-struts1-plugin-2.0.11.jar.

Tímto způsobem lze tedy spouštět ve Struts 2 jednoduché akce napsané pro Struts 1. Běžně však tyto akce nejsou takto jednoduché a používají navíc k získávání a zobrazování dat ActionForm, který dále validují pomocí frameworku Commons Validator. Jako příklad takovéto Struts 1 akce zde bude uvedena třída ExampleAction, která bude používat ActionForm, jenž bude představovat třída ExampleForm. Tato třída navíc bude validována pomocí validátoru Commons Validator, jehož konfigurační soubory budou uloženy v adresáři WEB-INF. K tomuto účelu je třeba v konfiguračním souboru struts.xml upravit definici interceptorů, a to následovně:

```
<interceptors>
    <interceptor name="exampleForm"
class="com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor">
       <param name="className">struts1.ExampleForm</param>
       <param name="name">exampleForm</param>
    </interceptor>
```

```
    <interceptor name="exampleValidation"
     class="org.apache.struts2.s1.ActionFormValidationInterceptor">
      <param name="pathnames">/WEB-INF/validator-rules.xml,
                              /WEB-INF/validation.xml</param>
    </interceptor>
    <interceptor-stack name="S1FormValidStack">
      <interceptor-ref name="staticParams"/>
      <interceptor-ref name="exampleForm"/>
      <interceptor-ref name="modelDriven"/>
      <interceptor-ref name="actionForm-reset"/>
      <interceptor-ref name="basicStack"/>
      <interceptor-ref name="exampleValidation"/>
      <interceptor-ref name="workflow"/>
    </interceptor-stack>
  </interceptors>
  <default-interceptor-ref name="S1FormValidStack"/>
</interceptors>
```

Jak je vidět, definice interceptorů se potom stává o dost složitější. Aby byla definice interceptorů o něco srozumitelnější, nejdůležitější z nich zde budou krátce vysvětleny. Interceptory exampleForm a modelDriven slouží k naplnění dat do třídy ExampleForm. Interceptor actionForm-reset pak umožňuje spouštění metody reset ve třídě ExampleForm. Interceptor basicStack je součástí standardního Struts 2. Validaci dat pak zajišťuje interceptor exampleValidation, který zde k Validaci využívá framework Commons Validator. Aby tento interceptor správně fungoval, je nutné k aplikaci přidat další zásuvný modul z distribuce Struts 2, a sice soubor se jménem commons-validator-1.3.0.jar. Pokud není potřeba používat k validaci framework Commons Validator a validace má být prováděna pomocí metody validate ve třídě ExampleForm, pak stačí z definice interceptoru exampleValidation vypustit značku param i s celým jejím obsahem.

Nyní již lze nadefinovat akci, která bude používat třídy napsané pro framework Struts 1. Konkrétně třídu ExampleAction, která bude používat action form ExampleForm a ten validovat pomocí frameworku Commons Validator, navíc se ve třídě ExampleForm bude spouštět metoda reset, tak jak je tomu zvykem ve Struts 1. Formulář pak bude umístěn na stránce exampleForm.jsp, tato JSP stránka však musí používat značky frameworku Struts 2, není možné tedy použít beze změn původní JSP stránku. Po úspěšném vyplnění formuláře se zobrazí stránka result.jsp. Definice této akce pak vypadá následovně:

```
<action name="ExampleAction" class="org.apache.struts2.s1.Struts1Action">
  <param name="className">struts1.ExampleAction</param>
  <param name="validate">true</param>
  <result name="input">exampleForm.jsp</result>
  <result>result.jsp</result>
</action>
```

Značka ve Struts 1	**Odpovídající značka ve Struts 2**
html:base	Není součástí Struts 2
html:button	s:submit s parametrem type="button"
html:cancel	s:submit
html:errors	s:actionerror popřípadě s:fielderror
html:file	s:file
html:form	s:form
html:frame	Není součástí Struts 2
html:hidden	s:hidden
html:html	Není součástí Struts 2
html:image	s:submit s parametrem type="image"
html:img	Není součástí Struts 2
html:link	s:a popřípadě zkombinovaná se značkou s:url
html:multibox	s:checkboxlist
html:option	s:select
html:options	s:select
html:optionsCollection	s:select
html:param	s:param
html:password	s:password
html:radio	s:radio
html:reset	s:reset
html:rewrite	pomocí s:url
html:select	s:select
html:submit	s:submit
html:text	s:textfield
html:textarea	s:textarea
html:xhtml	Není součástí Struts 2

Tabulka 5.3: Převod značek frameworku Struts 1 do frameworku Struts 2

Kapitola 6

Případová studie modernizace frameworku Struts

V této kapitole je uvedena případová studie modernizace frameworku Struts 1. Tato studie je realizována na aplikaci Kalkulátor energetické náročnosti technologií pro čistění kontaminovaných médií (Waste Site Energy Calculator) a modernizace bude provedena do frameworku Struts 2. Poměrně podrobné srovnání frameworků Struts 1 a Struts 2 bylo již zmíněno v předchozím textu. Modernizace bude probíhat tím způsobem, že se do aplikace přidá framework Struts 2 a poté se budou převádět jednotlivé akce z frameworku Struts 1 do frameworku Struts 2.

Je vhodné upozornit, že mnohé poznatky získané převodem této aplikace byly zobecněny a použity v kapitole Modernizace aplikace ze Struts 1 na Struts 2. Z toho důvodu již není nutné je znovu uvádět.

6.1 Popis aplikace

Následující popis aplikace je částečně přejat z [5].

Projekt byl zahájen přibližně na počátku roku 2004 na základě dohody mezi Fakultou informatiky Masarykovy univerzity a americkými organizacemi US EPA a PPRC[1], které se zabývají ochranou životního prostředí. Účelem projektu je vytvořit aplikaci pro provozovatele čistících areálů určených k odstraňování různých druhů kontaminantů (pohonné hmoty, těžké kovy, těkavé látky, kyanidy,. . .) z různých médií (půda, vzduch, voda,. . .). Systém umožňuje na základě vstupů zadaných uživatelem stanovit energetickou náročnost rozličných technologií pro čištění těchto kontaminovaných medií (tzv. sanačních technologií) a v konečném důsledku pak umožnit vybrat tu technologii, která nejméně zatěžuje životní prostředí. Úzce s touto problematikou souvisí publikace [6].

Technicky je celý systém realizován jako webová aplikace na platformě Java 2 Enterprise Edition. Jako webový server a J2EE kontejner je použit Jakarta Tomcat. To, co je pro tuto případovou studii nejdůležitější, je, že tato aplikace používá MVC framework Struts 1. Pro automatizovaný překlad a činnosti související s nasazením systému je použit javový nástroj Ant.

1. U.S. Environmental Protection Agency, The Northwest Pollution Prevention Resource Center

6.2 Přidání nového frameworku a základní konfigurace

Přidání frameworku Struts 2 do stávající aplikace obnáší dva základní kroky, editaci souboru web.xml a přidání knihoven frameworku Struts 2.

Editace souboru web.xml vyžaduje přidání následujících řádek, které přidají filtr frameworku Struts 2.

```
<filter>
  <filter-name>struts2</filter-name>
  <filter-class
   >org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
</filter>
<filter-mapping>
  <filter-name>struts2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Dalším krokem je přidání knihoven frameworku Struts 2. Konkrétně se jedná o knihovny freemarker-2.3.8.jar, ognl-2.6.11.jar, struts2-core-2.0.11.jar a xwork-2.0.4.jar. Tyto knihovny jsou pak při sestavení aplikace nakopírovány do adresáře WEB-INF/lib, což je zajištěno úpravou souboru build.xml, který potom při sestavování využívá program Ant. Více zde tato problematika rozebírána nebude, neboť případné podrobnosti lze zjistit z dokumentace programu Ant.

Tím byl framework Struts 2 přidán do aplikace. Dalším krokem bylo vytvoření souboru struts.xml, který je umístěn v classpath aplikace, a nastavení zdroje zpráv pro framework Struts 2. Původní Struts 1 používal pro zobrazování zpráv soubor CalculatorResources, který byl umístěn v balíku calculator.system. To bylo možné vyčíst ze souboru struts-config.xml, který obsahoval řádek:

```
<message-resources parameter="calculator.system.CalculatorResources" />
```

Struts 2 byl rovněž nastaven na používání stejného konfiguračního souboru, a sice tím, že do jeho konfiguračního souboru struts.xml byl přidán následující kód:

```
<constant name="struts.custom.i18n.resources"
          value="calculator.system.CalculatorResources" />
```

Tímto byl do aplikace přidán framework Struts 2 a zároveň byla provedena jeho základní konfigurace.

6.3 Převedení akce basic

Nyní bude demonstrován převod akce basic. Je nutné upozornit, že zde demonstrovaný převod je značně zjednodušen a že uvedené výpisy kódu jsou maximální možnou měrou zkráceny, aby je zde bylo možné uvést. Pro naznačení dalšího obsahu, který byl ve výpisu vypuštěn, jsou ve zdrojovém kódu použity tři tečky.

6.3.1 Konfigurace akce

V souboru struts-config.xml vypadal zápis týkající se akce basic následujícím způsobem:

```
...
<form-bean name="BasicForm" type="calculator.web.actionforms.BasicForm"/>
...
<action path="/basic" type="calculator.web.actions.BasicAction"
        name="BasicForm" validate="true" input="basic" scope="request">
 <forward name="basic" path="/basic.jsp" redirect="false" />
 <forward name="technologies" path="/technologies.jsp" redirect="false"/>
</action>...
```

Po převedení do Struts 2 pak obsah souboru struts.xml vypadal následovně:

```
...
<package name="default" extends="struts-default" >
  <action name="basic" class="calculator.web.actions.BasicAction" >
    <result name="basic">/basic.jsp</result>
    <result type="redirect-action"
            name="technologies">technologies</result>
    <result name="input">/basic.jsp</result>
  </action>
</package>
...
```

Jak provádět převod souboru struts-config.xml na struts.xml je podrobně popsáno v minulé kapitole. Proto zde uvedu pouze stručný přehled nejdůležitějších změn provedených v tomto konkrétním příkladu.

První změnou je, že byl zcela odstraněn form bean BasicForm. Dále se pak, dle konvencí Struts 2, musela stát akce součástí některého balíku (package), v tomto případě je to balík se jménem default. Dále byly změněny názvy parametrů u značky action tak, aby odpovídaly značení Struts 2. Poslední úpravou byl pak převod značek forward na značky result.

6.3.2 Převedení třídy akce

Ve Struts 1 původně třída BasicAction vypadala následovně:

```
...
public class BasicAction extends Action {
  public ActionForward execute(ActionMapping mapping, ActionForm form,
           HttpServletRequest request, HttpServletResponse response)
           throws Exception {
    ActionForward goTo = mapping.findForward("basic");
    HttpSession session = request.getSession();
    SessionData sessionData =
      (SessionData) session.getAttribute(SessionData.CLASS_SESSION_ID);
    String next = request.getParameter("next");
```

```
    ...
    if ((next != null) && form.validate(mapping, request).isEmpty()) {
      Logger.info("Reinicializing datafields for choosen unitsystem.");
      ...
      goTo = mapping.findForward("technologies");
    }
    return goTo;
  }
}
```

Tato třída využívala action form BasicForm. Ten pak zkráceně vypadal takto:

```
...
public class BasicForm extends ActionForm {
  private String name;
  ...
  public void setName(String name) { this.name = name; }
  public String getName() { return name; }
  ...
  public void reset(ActionMapping mapping, HttpServletRequest request) {
    ...
  }
  ...
  public ActionErrors validate(ActionMapping mapping,
        HttpServletRequest request) {
    ActionErrors errors = new ActionErrors();
    ...
    if (concentration < cleanupGoal) {
     errors.add("concentration",new ActionMessage("error.concentration"));
    }
    return errors;
  }
}
```

Při převodu do Struts 2 byl zvolen typický způsob, kdy se action form stane součástí třídy akce a obě třídy se tak sloučí v jednu.

```
...
public class BasicAction extends ActionSupport implements
        ServletRequestAware {
  public String execute() throws Exception {
    String goTo = "basic";
    HttpSession session = request.getSession();
    SessionData sessionData = (SessionData)
      session.getAttribute(SessionData.CLASS_SESSION_ID);
    String next = request.getParameter("next");
    ...
    if ((next != null) && !hasErrors() ) {
      Logger.info("Reinicializing datafields for choosen unitsystem.");
```

```
      ...
      goTo = "technologies";
    }
    return goTo;
  }
  //ServletRequestAware implementation
  private HttpServletRequest request;
  public void setServletRequest(HttpServletRequest request) {
      this.request = request;
  }
  //BasicForm
  private String name;
  public void setName(String name) { this.name = name; }
  public String getName() {return name; }
  ...
  public void reset() {
    ...
  }
  ...
  public void validate() {
    ...
    if (concentration < cleanupGoal) {
      addFieldError("concentration",getText("error.concentration"));
    }
  }
}
```

Většina změn potřebných pro převod této třídy byla již popsána a vysvětlena v předešlé kapitole. Proto zde budou nejdůležitější změny provedené při převodu popsány pouze stručnou formou v následujících bodech:

- Třída BasicAction nerozšiřuje třídu Action, ale třídu ActionSupport, která je součástí frameworku Struts 2.
- Metoda execute byla upravena do tvaru odpovídajícímu Struts 2. To znamená, že nevrací typ ActionForward, ale String a navíc byly odstraněny všechny její parametry.
- Třída BasicAction implementuje rozhraní ServletRequestAware, aby bylo možné nadále pracovat s objektem request.
- Typ proměnné goTo byl změněn z ActionForward na String. Společně s tím byly upraveny i všechny výrazy, které do této proměnné přiřazovaly nějakou hodnotu. Jak je vidět z uvedeného kódu, výraz mapping.findForward("technologies") byl změněn na "technologies".
- Výraz form.validate(mapping, request).isEmpty() zapříčiňoval nesmyslné několikanásobné volání metody validate. Při přepisu do Struts 2 byl proto tento nahrazen výrazem !hasErrors(), který pouze zjistí, zda nedošlo k chybám a nespouští již znovu validaci.

- Dále pak obsah třídy BasicForm se stal součástí třídy BasicAction.
- U metody reset byly odstraněny parametry, protože ty již nadále ve Struts 2 nejsou potřebné.
- Rovněž u metody validate byly odstraněny parametry a navíc byl u této metody změněn návratový typ ActionErrors na void.
- Dále pak v metodě validate byla zcela odstraněna proměnná errors. A místa, kde se do této proměnné přidávala chybová hlášení, byla nahrazena metodami frameworku Struts 2. Například výraz

  ```
  errors.add("concentration", new ActionMessage("error.concentration"))
  ```

  byl nahrazen voláním metody

  ```
  addFieldError("concentration",getText("error.concentration"))
  ```

6.3.3 Převedení JSP stránky

Posledním krokem při převodu akce basic je převést JSP stránky používající tuto akci. V tomto konkrétním případě to znamenalo především upravit stránku basic.jsp, která ve velmi zkrácené podobě vypadala takto:

```
...
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib prefix="myhtml" tagdir="/WEB-INF/tags" %>
...
<html:form action="/basic" ... >
 <label class="grcolor" for="name"><bean:message key="site.name"/>
   <myhtml:help name="name"/></label>
 <html:text property="name" styleClass="textinput"/><br/>
 ...
 <span class="site_size"><bean:message key="site.volume"/></span>
 <html:radio property="size_type" styleId="size_type_volume"
     value="volume" style="margin-bottom: 0px;" />
 <span class="site_size"><bean:message key="site.area"/></span>
 <html:radio property="size_type" styleId="size_type_area"
     value="area" style="margin-bottom: 0px;" />
 ...
 <myhtml:select property="contaminant" css_class="textinput"
  dataFieldNamespace="site" dataFieldObject="<%=...%>" />
 ...
</html:form>
```

Po převedení do Struts 2 pak stránka vypadala následovně:

```
...
<%@ taglib prefix="s" uri="/struts-tags" %>
<%@ taglib prefix="myhtml" tagdir="/WEB-INF/tags" %>
...
<s:form action="basic" ... theme="simple">
 <label class="grcolor" for="name"><s:text name="site.name"/>
   <myhtml:help name="name"/></label>
 <s:textfield name="name" cssClass="textinput" /><br/>
 ...
 <s:set name="volumeTypeMap" value="#{'volume':getText('site.volume'),
                                      'area':getText('site.area')}" />
 <s:radio name="size_type" id="size_type_"
  cssStyle="margin-bottom: 0px;" list="volumeTypeMap" theme="mytheme"/>
 ...
 <myhtml:selects2 property="contaminant" css_class="textinput"
  dataFieldNamespace="site" dataFieldObject="<%=...%>" />
 ...
</s:form>
```

Změny, které zde byly provedeny, jsou následující:

- Knihovny značek frameworku Struts 1 (bean a html) byly nahrazené knihovnou frameworku Struts 2.
- Značka html:form byla nahrazena za značku s:form. Navíc bylo pro tento formulář nastaveno téma simple, díky kterému je výsledný kód vygenerovaný značkami frameworku Struts 2 velmi podobný kódu, který generují značky frameworku Struts 1. Díky tomu nejsou nutné velké změny v kaskádových stylech použitých pro tuto stránku.
- K výpisu lokalizovaných zpráv se místo značky bean:message použila značka s:text. Při přidávání frameworku Struts 2 bylo nastaveno, aby používal stejný lokalizační soubor, jako používal Struts 1. Díky tomu budou nově vypisované zprávy shodné s původními.
- Značka html:text byla převedena na značku s:textfield. Převod této značky je podrobně rozebrán v předešlé kapitole.
- Dvojice značek html:radio a jejich popisků byla nahrazena jedinou značkou s:radio. Pro větší přehlednost byl nejprve definován list hodnot a popisků s názvem volumeTypeMap, který byl potom ve značce s:radio použit. Id každého radiobuttonu je ve Struts 2 nastaveno na hodnotu parametru id u značky radio zřetězenou s hodnotou konkrétního radiobuttonu. Díky tomu id ve výsledném kódu zůstane stejné, jako tomu bylo ve Struts 1. Navíc je u této značky použito vlastní téma mytheme, které upravuje téma simple tak, aby výsledné radiobuttony byly na stránce zobrazeny pod sebou. Popis toho, jak lze vytvořit vlastní téma, je uveden v předešlé kapitole, proto zde již nebude rozebírán.

- Značka myhtml:select je speciální značka vytvořená přímo pro účely této aplikace. Tato značka však používá třídy závislé na frameworku Struts 1 a navíc generuje značky tohoto frameworku. Z toho důvodu bylo nutné tuto značku rovněž upravit. Protože je však tato značka využívána i na jiných JSP stránkách, které dosud používají framework Struts 1, je nutné ji prozatím ponechat a použít místo ní značku myhtml:select2, která je vytvořena pro účely frameworku Struts 2. Úprava této značky zde však rozebrána nebude, neboť je až příliš závislá na této aplikaci a problematiky modernizace MVC frameworku se dotýká jen okrajově.

Tímto byla kompletně převedena akce basic do frameworku Struts 2.

6.4 Některé další problémy

6.4.1 Přístup k proměnné definované uvnitř JSP stránky

Při převodu této aplikace ze Struts 1 na Struts 2 bylo třeba převést některé JSP stránky, které obsahovaly zhruba následující kód.

```
<%
Technology technology = ...//složitý výpočet
%>
<bean:message key="<%=technology.getLabel()%>"/>
```

Převod této značky bean:message je na první pohled jednoduchý a vede k tomu, že tato značka bude nahrazena značkou s:text. Problémem převodu tohoto kódu však je, že uvnitř značek frameworku Struts 2 nelze používat žádné skriplety. A díky tomu se nelze dostat na proměnnou technology, která byla nadefinována přímo na JSP stránce.

Možné řešení tohoto problému je umístit proměnnou na ValueStack frameworku Struts 2 a potom k získání jejího obsahu použít běžný OGNL výraz. Převést tuto část JSP stránky lze tedy například následujícím způsobem:

```
<%
Technology technology = ...//složitý výpočet
pageContext.setAttribute("technologyLabel",technology.getLabel());
%>
<s:text name="%{#attr.technologyLabel}" />
```

6.4.2 Části používané z více míst aplikace

Některé JSP stránky (nebo i jiné části aplikace) nemohly být rovnou přepsány do nového frameworku, protože byly stále používány jinými částmi aplikace, které ještě nebyly převedeny. Pokud by taková JSP stránka byla převedena do Struts 2, dosud nepřevedená část aplikace by ji pak nedokázala používat.

Z toho důvodu bylo nutné vytvořit kopii takovýchto JSP stránek, která pak byla převedena do Struts 2. Struts 1 pak používal původní JSP stránku a části převedené do Struts 2

používaly převedenou kopii této stránky. Po převedení celé aplikace pak byly originální JSP stránky nahrazeny kopiemi a framework Struts 2 byl překonfigurován tak, aby používal správné JSP stránky. Překonfigurování pak většinou obnášelo pouze editaci souboru struts.xml.

Stejný způsob pak musel být použit i při převodu některých akcí a jiných částí aplikace.

6.5 Další postup

Další akce pak byly převedeny obdobným způsobem jako již popsaná akce basic. Po dokončení převodu všech akcí pak byly přejmenovány dočasné a zkopírované soubory a celá aplikace tak byla převedena.

Kapitola 7

Závěr

Použití MVC frameworku urychluje vývoj webové aplikace a zároveň usnadňuje její další údržbu. Díky tomu se MVC frameworky staly v podstatě standardní součástí webových aplikací. Zároveň byly vyvíjeny i nové frameworky, které vývoj a správu webové aplikace usnadňují ještě více. Mnoho aplikací však stále používá některý ze starších frameworků, proto bylo cílem práce prozkoumat možnosti modernizace MVC frameworku.

I přes nesporné výhody novějších frameworků se ve většině aplikací kompletní přechod na nový framework pravděpodobně nevyplatí, a to z toho důvodu, že modernizace je zpravidla poměrně náročná a vyžaduje nemalé změny v kódu aplikace. Přechod na nový framework se tedy může vyplatit jedině v dlouhodobém horizontu. Pro mnoho aplikací by však mohlo být vhodným řešením použití dvou frameworků současně, kdy nová funkcionalita aplikace je programována v novějším frameworku a stávající část aplikace používá stále starý framework. Při tomto způsobu řešení lze navíc dle potřeby postupně přepisovat části starého frameworku do nového a popřípadě tak postupně provést kompletní modernizaci. Součástí práce je rovněž navržená obecná strategie, kterou lze postupovat při modernizaci požadavkově orientovaného MVC frameworku.

Při modernizaci vždy hraje největší roli, z kterého konkrétního frameworku do kterého je modernizace prováděna. Z toho důvodu je druhá část práce věnována velmi stručnému představení nejpoužívanějších frameworků a především se tato část zabývá studií jednotlivých případů, které pak napomáhají při převodu aplikace používající framework Struts 1 na aplikaci používající framework Struts 2. Jsou zde rovněž zmíněny možnosti, kterými lze převod zjednodušit nebo jak při převodu využít stávající kód.

Jako praktická část práce byl realizován převod aplikace Waste Site Energy Calculator, která slouží k výpočtu energetické náročnosti dekontaminačních technologií.

K problematice modernizace MVC frameworků neexistuje žádná literatura, která by se jí uceleně zabývala. Proto je přínosem této práce právě shrnutí možností, kterými lze modernizaci provést. Dalším přínosem je rozebíraná problematika převodu existujícího Struts 1 kódu do Struts 2. O této problematice sice existuje několik materiálů, ale ty se omezují pouze na základní informace a nevěnují se této tématice v takovém rozsahu, v jakém se jí věnuje tato práce.

Literatura

[1] Sun Microsystems: *Java BluePrints – J2EE Patterns: Model–View–Controller*, <http://java.sun.com/blueprints/patterns/MVC-detailed.html>.

[2] Inderjeet Singh: Beth Stearns: Mark Johnson: Greg Murray: *Designing Enterprise Applications with the J2EE Platform, Second Edition*, Duben 2002, ISBN 0-201-78790-3, <http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/titlepage.html>. 1

[3] Michael Nash: *The Java Framework Landscape*, <http://www.developer.com/design/article.php/10925_3617156_1>. 3.5

[4] *Model View Controller History*, http://c2.com/cgi/wiki?ModelViewControllerHistory <http://folk.uio.no/trygver/themes/mvc/mvc-index.html>.

[5] Marek Cikánek: *Reinženýrství a modernizace existujících softwarových systémů*, [Diplomová práce], Masarykova univerzita, <https://is.muni.cz/auth/th/39276/fi_m/dp.pdf>. 6.1

[6] Jan Pavlovič: Katarina Mahutová: *Energy Management at Waste Clean up Sites, Avoiding Secondary Air Impacts to Human Health. In Environmental Health in Central and Eastern Europe.*, 2006, ISBN 1-4020-4844-0. 6.1

[7] Matt Raible: *Migrating from Struts 1 to Struts 2*, říjen 2006, <http://www.softwaresummit.com/2006/speakers/RaibleMigratingStruts.pdf>. 1, 5.1

[8] Matt Raible: *Comparing Java Web Frameworks*, <http://static.raibledesigns.com/repository/presentations/ComparingJavaWebFrameworks-ApacheConUS2007.pdf>. 1, 4.1

[9] Ian Roughley: *Starting Struts 2*, 2006, ISBN 978-1-4303-2033-3, <http://www.infoq.com/resource/minibooks/starting-struts2/en/pdf/StartingStruts2online2.pdf>. 1

[10] *Jakarta Struts Tutorial*, <http://www.roseindia.net/struts/>. 1

[11] *Struts 2 Tutorial*, <http://www.roseindia.net/struts/struts2/>. 1

[12] Ken Paulsen: Staff Engineer: *Struts and JavaServer Faces*, <http://www.pjug.org/JavaServerFaces.pdf>.

[13] *Java Web Framework Sweet Spots*, 25. dubna 2006, <http://www.virtuas.com/files/JavaWebFrameworkSweetSpots.pdf>. 1

[14] Jan Tichý: *Programová podpora tvorby webových aplikací*, [Diplomová práce], Vysoká škola ekonomická v Praze, <http://www.jantichy.cz/diplomka>. 2.1

[15] Trygve M. H. Reenskaug: *MVC XEROX PARC 1978-79*, <http://folk.uio.no/trygver/themes/mvc/mvc-index.html>. 1

[16] Neal Ford: *Comparison of Java Web Frameworks*, <http://www.coadletter.com/article/borcon/files/6000/paper/6000.html>. 1

[17] The Apache Software Foundation: *Struts 1*, <http://struts.apache.org/1.x/>. 1

Příloha A

Typická akce zajišťující lokalizaci frameworku Struts 1

```
public final class LocaleAction extends Action {
  private static final String LANGUAGE = "language";
  private static final String COUNTRY = "country";
  private static final String SUCCESS = "success";
  public ActionForward execute(ActionMapping mapping,
    ActionForm form, HttpServletRequest request,
    HttpServletResponse response) throws Exception {
    String language = request.getParameter(LANGUAGE);
    String country = request.getParameter(COUNTRY);
    Locale locale = getLocale(request);
    if ((language != null && language.length() > 0) &&
      (country != null && country.length() > 0)) {
      locale = new java.util.Locale(language, country);
    }else if (language != null && language.length() > 0){
      locale = new java.util.Locale(language, "");
    }
    setLocale(request, locale);
    return mapping.findForward(SUCCESS);
  }
}
```