VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

Fakulta informatiky a statistiky

Katedra informačních technologií





Vývoj webových aplikací pomocí frameworku JavaServer Faces

Bakalářská práce

David Hanel

Vedoucí práce: Ing. Luboš Pavlíček

prosinec 2009

Anotace

Bakalářská práce se zabývá technologií JavaServer Faces (JSF). Jedná se o serverový framework, jenž by měl usnadňovat tvorbu webových aplikací fungujících na Java platformě.

Po krátkém úvodu do architektury Java EE platformy jsou v jednotlivých kapitolách analyzovány možnosti, které JSF vývojářům přináší. Celý text je podán ve formě srozumitelné pro ty, kteří mají zkušenosti s vývojem webových aplikací pomocí technologie JavaServer Pages (JSP), avšak s tímto frameworkem se setkávají poprvé. Postupně budou představeny základní prvky JSF aplikací a jejich funkcionalita, jako např. komponenty uživatelského rozhraní, navigace mezi stránkami, konverze a validace dat, zpracování událostí, životní cyklus aplikace atd.

Práce může sloužit jako výukový materiál. Z toho důvodu jsou na konci kapitol tři až sedm vložena také zadání příkladů k procvičení, jejichž možná řešení lze nalézt v přiložených souborech.

Klíčová slova

JavaServer Faces (JSF), Java Enterprise Edition (Java EE), webový framework, managed bean, JSF konverze, JSF validace, JSF životní cyklus

Annotation

This bachelor thesis deals with a JavaServer Faces technology (JSF). It is a server-side framework which should facilitate a development of Java technology-based web applications.

Firstly there is a brief introduction of the Java EE platform architecture. Then the possibilities and advantages of JSF are analyzed subsequently. The whole text is adjusted to those who are experienced in JavaServer Pages (JSP) but have never used the JSF framework. In individual chapters are introduced the basic aspects of JSF applications and their functionality, like for example user interface components, page navigation, data conversion and validation, event handling, application life cycle etc.

The thesis may be used as an educational material. Therefore there are also some practising tasks at the end of the chapters three till seven. Their possible solutions may be found in the attached files.

Keywords

JavaServer Faces (JSF), Java Enterprise Edition (Java EE), web framework, managed bean, JSF conversion, JSF validation, JSF life cycle

Poděkování

Rád bych na tomto místě poděkoval všem lidem, kteří měli větší či menší podíl na vzniku této práce.

Velký dík patří především vedoucímu mé bakalářské práce panu Ing. Luboši Pavlíčkovi, který mi poskytoval kritickou a velmi přínosnou zpětnou vazbu a ochotně se mnou konzultoval různé nejasnosti.

Dále bych rád poděkoval svým spolubydlícím Bc. Františku Odehnalovi, Bc. Václavu Pospíšilovi a Jiřímu Štenckovi, kteří trpělivě snášeli mé večery prosezené u počítače. Práci si navíc pročetli a poskytli mi řadu užitečných připomínek.

Poslední osobou, které chci vyjádřit svůj dík, je má přítelkyně slečna Ivona Marešová. Po celou dobu mě přísně kontrolovala a kdykoliv jsem jen na okamžik polevil v úsilí, viděl jsem její zdvižený ukazováček, který mi připomínal blížící se termín odevzdání.

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze kterých jsem čerpal.

V Praze dne 9. prosince 2009 .....

Podpis

Obsah

1. Úvod 8
2. Základní principy Java *Enterprise Edition* 10
    2.1. Java EE platforma 10
    2.2. Aplikační server 11
        2.2.1. Kontejnery 12
    2.3. Vícevrstvá architektura 12
        2.3.1. Klientská vrstva 13
        2.3.2. Webová vrstva 14
        2.3.3. *Business* vrstva 14
        2.3.4. *Enterprise Information System* (EIS) vrstva 14
3. Úvod do *JavaServer Faces* 15
    3.1. Co je to JSF 15
    3.2. Hlavní výhody JSF 16
    3.3. První JSF aplikace 18
        3.3.1. Z čeho se JSF aplikace skládá 18
        3.3.2. Postup pří vývoji JSF aplikace 19
    3.4. Příklad k procvičení 31
4. UI komponenty 33
    4.1. Základní principy UI komponent modelu 33
    4.2. Renderování UI komponent 35
    4.3. Použití UI komponent na stránce 36
        4.3.1. Základní atributy tagů 36
        4.3.2. Formuláře 37
        4.3.3. Textové prvky 38
        4.3.4. Tlačítka a odkazy 40
        4.3.5. Selekce 41
        4.3.6. Zprávy 43
        4.3.7. Panely 44
        4.3.8. Ostatní 45
    4.4. Příklad k procvičení 45
5. Konverze a validace 47
    5.1. Konvertory 47
        5.1.1. Použití konvertorů na stránce 48
    5.2. Validátory 50
        5.2.1. Použití validátorů na stránce 50
    5.3. Příklad k procvičení 51
6. Zpracování událostí 53

6.1. Zpracování událostí metodou ... 54
6.2. Zpracování událostí listenerem ... 55
6.3. Příklad k procvičení ... 56
7. Životní cyklus JSF aplikace ... 58
7.1. Fáze *Restor View* ... 60
7.2. Fáze *Apply Request Values* ... 61
7.3. Fáze *Process Validations* ... 61
7.4. Fáze *Update Model Values* ... 61
7.5. Fáze *Invoke Application* ... 62
7.6. Fáze *Render Response* ... 62
7.7. Události v životním cyklu aplikace ... 62
7.7.1. Ovlivňování životního cyklu pomocí atributu `immediate` ... 63
7.8. Příklad k procvičení ... 64
8. Závěr ... 65
A. Výpis souborů první JSF aplikace ... 68
A.1. **`index.jsp`** ... 68
A.2. **`vitejte.jsp`** ... 69
A.3. **`Uzivatel.java`** ... 70
A.4. **`faces-config.xml`** ... 70
A.5. **`web.xml`** ... 71
B. Knihovna *core tag library* ... 72
C. Knihovna *HTML component tag library* ... 74
D. Třídy UI komponent a jejich renderery ... 76
E. Postup nasazení aplikace na aplikační server GlassFish ... 79
E.1. Spuštění aplikačního serveru ... 79
E.2. Přihlášení se k administrátorské konzoli ... 80
E.3. Nasazení aplikace na server ... 81
F. Příklady k procvičení ... 83
F.1. Příklad z kapitoly 3 – *„Úvod do JavaServer Faces“* ... 83
F.2. Příklad z kapitoly 4 – *„UI komponenty“* ... 83
F.3. Příklad z kapitoly 5 – *„Konverze a validace“* ... 84
F.4. Příklad z kapitoly 6 – *„Zpracování událostí“* ... 84
F.5. Příklad z kapitoly 7 – *„Životní cyklus JSF aplikace“* ... 85
Terminologický slovník ... 86
Literatura ... 88

Kapitola 1

Úvod

Práce analyzuje možnosti technologie JavaServer Faces (JSF), frameworku, jenž je vyvíjen od roku 2004 a snaží se usnadnit tvorbu uživatelského rozhraní webových aplikací postavených na platformě Java. O tomto tématu bylo dosud napsáno poměrně málo literatury v češtině. Snažím se tedy koncipovat text tak, aby mohl sloužit také jako výukový materiál a to především těm, kteří se s JSF setkávají poprvé. Mým dalším cílem je zpřístupnit tuto práci také na doprovodných webových stránkách.

Pro plné porozumění obsahu je předpokládaná znalost některých technologií, na kterých je JSF založeno. Jedná se zejména o Java Standard Edition (Java SE)[1] a základní principy Java Enterprise Edition (Java EE).[2] Dále je potřebná znalost technologie JavaServer Pages (JSP)[3] a také značkovacího jazyka Hypertext Markup Language (HTML).[4]

Co se obsahu týče, budou postupně představeny základní principy a možnosti JSF frameworku. Začíná se stručným popisem Java EE platformy. Poté následuje úvod do samotného JSF a napsání první jednoduché aplikace. V dalších kapitolách jsou postupně probrány UI komponenty a jejich renderovací model, konverze a validace dat, události a jejich zpracovávání a nakonec je popsán průběh životního cyklu JSF aplikace. Na konci jednotlivých kapitol naleznete také cvičná zadání, která by měla sloužit k procvičení právě probrané problematiky. K práci jsou přiloženy soubory, ve kterých jsou tyto příklady vypracovány. Pro vypracování ukázkových aplikací bylo využito JSF 1.2, což je v době psaní této práce (říjen 2009) aktuální verze.

K tomu abyste byli schopni vyvíjet a spouštět JSF aplikace, budete potřebovat nějaké softwarové vybavení. Předně je nezbytné mít nainstalován Java Development Kit (JDK).[5] Dále budete potřebovat aplikační server, na kterém budete vaše weby testovat. Můžete

[1] http://java.sun.com/javase
[2] http://java.sun.com/javaee
[3] http://java.sun.com/products/jsp
[4] http://www.w3.org/html
[5] http://java.sun.com/javase/downloads

využít např. zdarma dostupný GlassFish,[6] jehož verze 2.1 byla využita také v této práci. Pro usnadnění práce se také hodí některé z vývojových prostředí. Mezi nejznámější patrně patří Eclipse,[7] IntelliJ IDEA[8] a NetBeans,[9] v němž jsou vytvořeny ukázkové příklady. Můžete však použít jakýkoliv jiný software. Informace o spolupráci vašeho vývojového prostředí s konkrétním aplikačním serverem hledejte v příslušné dokumentaci.

V textu jsou použity následující konvence:

- Důležité poprvé použité termíny nebo pojmy, které chci zvýraznit, jsou vysazeny **tučně**.
- U odborných výrazů z angličtiny se nesnažím o překlad do češtiny, nýbrž používám jejich anglického tvaru. Takovéto výrazy jsou vysazeny *kurzívou*. Akronymy anglických výrazů (např. JSF, API atd.) však píši standardním fontem.
- Názvy tříd a programový kód jsou vysazeny `neproporcionálním písmem`.
- URL adresy a názvy souborů jsou vysazeny **`tučným neproporcionálním písmem`**.



Texty zadání příkladů jsou celé vysazeny kurzívou a uvedeny obrázkem.

Výše uvedené pravidla mohou být také kombinována. Např. důležitý odborný výraz z angličtiny bude vysazen ***tučnou kurzívou***. V poznámkách pod čarou se tyto konvence neuplatňují.

[6] http://java.sun.com/javaee/downloads
[7] http://www.eclipse.org
[8] http://www.jetbrains.com/idea
[9] http://netbeans.org

Kapitola 2

Základní principy Java *Enterprise Edition*

Základem Java *Enterprise Edition* (Java EE) je Java *Standard Edition* (Java SE). Java EE však navíc poskytuje knihovny, které by měly usnadňovat vývoj komplexnějších *enterprise* aplikací. V této práci se chci primárně zabývat pouze jedinou z těchto knihoven, a sice ***JavaServer Faces*** (JSF). Nejprve je však třeba seznámit se s obecnými principy Java EE platformy. K tomu slouží tato kapitola.

Nejedná se však o vyčerpávající výklad, který by architekturu Java EE popisoval kompletně, nýbrž pouze o jakési nutné minimum. V dalším textu pravděpodobně narazíme na oblasti, na které v téhle kapitole není místo. V takových případech budete odkázáni na externí zdroje.

2.1. Java EE platforma

Termín Java označuje kromě programovacího jazyka také platformu. Java platforma se skládá z **virtuálního stroje** a příslušného **API**. Virtuální stroj je prostředí, ve kterém běží naše aplikace respektive naše třídy zkompilované do *byte* kódu. Je navržen pro konkrétní hardware a operační systém. API je sada předprogramovaných tříd, které nám jsou k dispozici při psaní našich vlastních tříd a umožňují nám přístup k funkcionalitě virtuálního stroje. Právě v obsahu API se od sebe jednotlivé platformy liší.

Existují tři Java platformy - Java SE, Java ME a Java EE. Java SE a Java ME nejsou předmětem této práce a tudíž se jimi nebudu dále zaobírat.[1]

Již v úvodu jsem zmínil že, Java EE je nadstavbou nad Java SE. To znamená, že pokud chceme na nějakém počítači provozovat Java EE aplikaci, musí na něm být nainstalováno také standardní API, které je *enterprise* aplikacemi využíváno. Úlohu Java SE v Java EE aplikacích můžete vidět na obrázku 2.1.

[1] Pro více informací o Java SE a Java ME viz http://java.sun.com.

Java EE poskytuje navíc oproti Java SE okolo dvou desítek API, z nichž každé usnadňuje vývoj v určité oblasti. Jako příklad můžeme uvést *Java Transaction API* (JTA), které usnadňuje správu transakcí, *Java API for XML Web Services* (JAX-WS), které poskytuje podporu pro webové služby, či *JavaServer Faces* (více příkladů API můžete vidět na obrázku 2.1).[2] JSF, kterému je věnována tato práce, je tedy jedna z mnoha Java EE knihoven a je zaměřena na podporu tvorby webových aplikací, především jejich uživatelského rozhraní.[3]

Ve své prvotní podobě jsou však tato API pouhou **specifikací** Sunu. Tzn. jakési minimální požadavky na funkcionalitu Java EE platformy. Nejedná se tedy o konkrétní implementaci. Tu poskytují po uveřejnění této specifikace buď třetí osoby, které mohou nad rámec minimálních funkčních požadavků přidat také něco navíc, nebo samotný Sun, který zpravidla nic navíc nepřidává a jeho implementace se označuje jako **referenční**.

2.2. Aplikační server

Serverovou aplikaci zastřešující všechny knihovny, které dle specifikací Java EE platformy zajišťují požadovanou funkcionalitu, označujeme pojmem **aplikační server**. Tyto knihovny implementují veškerá API obsažená v Java EE. Kromě toho poskytuje aplikační server další klasické služby jako např. administrátorskou konzoli, logování atp. Ze známých implementací Java EE platformy můžeme zmínit např. JBoss od firmy Red Hat[4] či JRun od firmy Adobe Systems.[5] Referenční implementace Sunu je šířena pod hlavičkou projektu GlassFish.[6]

Na aplikační server se nasazují tzv. komponenty, což jsou základní jednotky, ze kterých je složena výsledná aplikace. Komponent existuje několik druhů, z nichž pro nás jsou důležité pouze dva:

- **Webové komponenty** - servlety, JSP soubory a JSF soubory.
- ***Enterprise JavaBeans* (EJB) komponenty** - javovské třídy, které tvoří logiku aplikace a manipulují s daty (pro více informací o EJB viz [8]).

[2] Pro kompletní přehled všech API, která jsou v Java EE k dispozici viz [8].
[3] Pod pojem JSF patří kromě knihovny tříd také knihovna tagů pro použití na JSP stránkách. Více se JSF budeme zabývat v následujících kapitolách.
[4] http://jboss.org/jbossas
[5] http://www.adobe.com/products/jrun
[6] https://glassfish.dev.java.net

Existují ještě další typy komponent, ale pouze tyto dvě se nasazují na aplikační server. Ostatní typy běží na klientských strojích, a proto pro nás nejsou v tuto chvíli důležité.

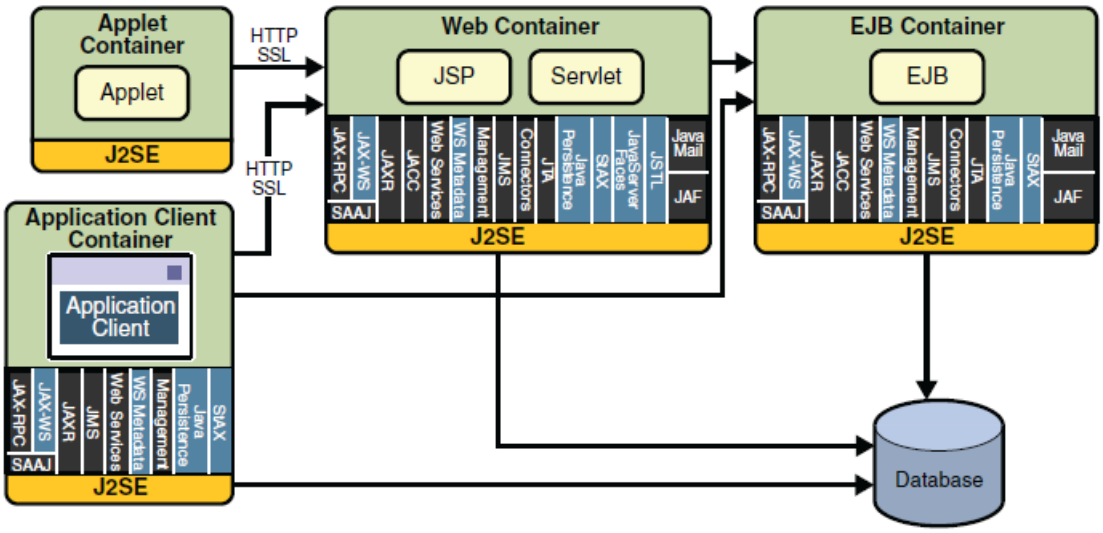
Předtím než může být komponenta spuštěna, musí být přiřazena do tzv. **kontejneru**.

2.2.1. Kontejnery

Java EE server v sobě zahrnuje tzv. kontejnery, které spravují komponenty určitého typu a poskytují jim svou funkcionalitu. Funkcionalita jednotlivých kontejnerů je dána specifikací Java EE. Druhy kontejnerů jsou analogické s druhy komponent. Opět jsou pro nás důležité pouze dva z nich:

- **Webový kontejner** - má na starosti správu webových komponent.
- **EJB kontejner** - má na starosti správu EJB komponent.

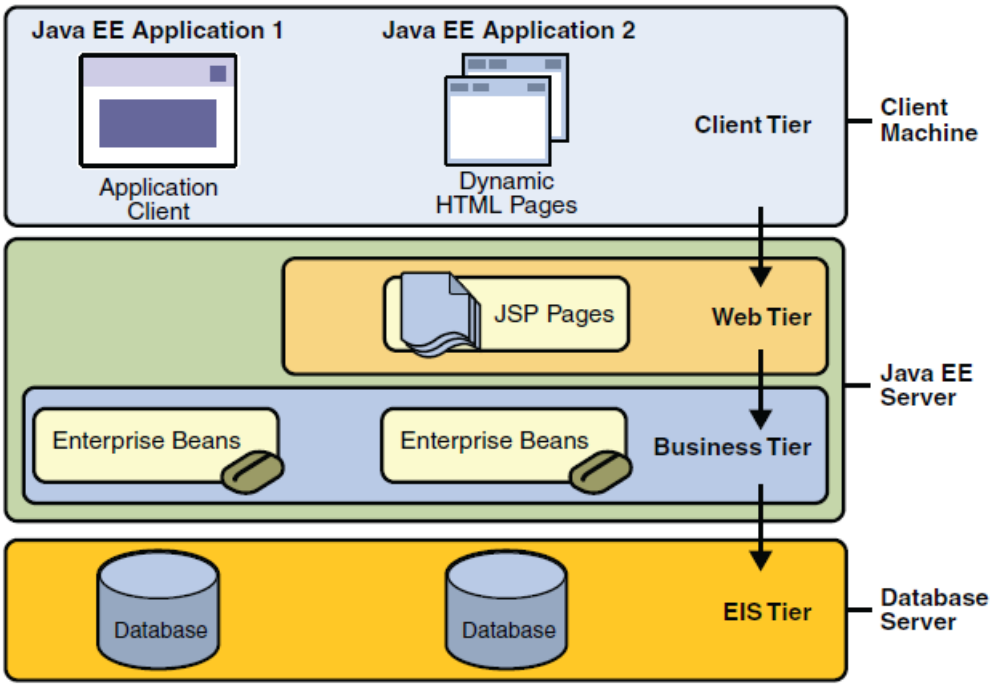
Existují celkem čtyři typy kontejnerů, ale pouze tyto dva běží na aplikačním serveru. Ostatní běží na klientských strojích, a proto pro nás nejsou v tuto chvíli důležité. Kooperaci jednotlivých kontejnerů můžete vidět na obrázku 2.1.



Obrázek 2.1. Kooperace Java EE kontejnerů (zdroj: [8])

2.3. Vícevrstvá architektura

Při použití vícevrstvé architektury jsou komponenty rozděleny podle svého typu do oblastí zvané **vrstvy**. Tyto vrstvy bývají instalovány na různých počítačích. Java EE používá tzv. **čtyřvrstvou architekturu**, jejíž jednotlivé vrstvy jsou vidět na obrázku 2.2.



Obrázek 2.2. Vrstvy v Java EE (zdroj: [8])

Java EE platforma se zaměřuje na vývoj ve webové a *business* vrstvě, které běží na aplikačním serveru. Jelikož tyto dvě vrstvy běží na stejném stroji, jsou někdy dohromady označovány jako **střední vrstva**. Pokud porovnáte obrázky 2.1 a 2.2, zjistíte, že v klientské, webové a *business* vrstvě běží také příslušné kontejnery (o kontejnerech jsme mluvili v sekci 2.2.1 – „Kontejnery“).

2.3.1. Klientská vrstva

Klientskou vrstvu představuje aplikace, která ani nemusí být napsána v Javě. Dva základní typy klientů jsou **webový prohlížeč** a **klientská aplikace**. Hlavní úlohou klienta je zasílat požadavky střední vrstvě, přijímat od ní odpovědi a prezentovat je uživateli.

Každý z těchto dvou typů klientů to však dělá trošku jiným způsobem. Webový prohlížeč komunikuje s webovou vrstvou pomocí HTTP protokolu. Webová vrstva následně interpretuje klientské požadavky *business* vrstvě, která je obsluhuje. Odpověď pak putuje zpět po stejném řetězci.

Klientská aplikace může rovněž komunikovat s webovou vrstvou. Na rozdíl od webového prohlížeče ji však může také přeskočit a komunikovat přímo s EJB komponentami v *business* vrstvě.

2.3.2. Webová vrstva

Webová vrstva je tvořena servlety, JSP a JSF soubory a případně také *JavaBeans*[7] komponentami. Tyto komponenty zpracovávají klientské požadavky a generují odpověď, která je následně zaslána do webového prohlížeče klienta.[8] Při zpracovávání odpovědi může webová vrstva komunikovat s *business* vrstvou a získávat od ní určité informace.

2.3.3. *Business* vrstva

Zde leží jádro celé aplikace. Veškerá logika a funkcionalita by měla být uložena v *business* vrstvě, konkrétně v EJB komponentách. Ty přijímají požadavky od klientské a webové vrstvy a na jejich základě pracují se zdroji z EIS vrstvy. Následně zasílají odpověď zpět klientské respektive webové vrstvě.

2.3.4. *Enterprise Information System* (EIS) vrstva

Tato vrstva představuje veškeré externí systémy, jejichž funkcionalitu či data *enterprise* aplikace využívá. Může se jednat například o ERP systém, databázový systém atp. [7]. Komunikaci s EIS zpravidla zajišťuje *business* vrstva.

[7] Pro více informací o JavaBeans komponentách viz http://java.sun.com/docs/books/tutorial/javabeans.

[8] V naprosté většině případů se jedná o HTTP komunikaci. Java servlety jsou však schopny komunikovat pomocí jakéhokoliv protokolu, který je založen na principu request-response. Stejně tak JSF soubory umí generovat i jiný výstup než jen HTML kód.

Kapitola 3

Úvod do *JavaServer Faces*

V této kapitole si poprvé představíme JSF. Nastíníme hlavní aspekty a výhody tohoto frameworku a napíšeme si první velice jednoduchou aplikaci. Na ní budeme demonstrovat základní možnosti JSF, které budeme podrobněji rozebírat v následujících kapitolách. Dále si čtenář udělá představu o součástech, ze kterých se JSF aplikace skládá a o základních krocích, které je třeba při jejím vývoji provést.

3.1. Co je to JSF

Základem JSF je technologie *JavaServer Pages* (JSP), ve které funguje systém knihoven tagů[1] a JSF obsahuje právě dvě takovéto knihovny (těmito knihovnami se budeme blíže zaobírat v následujících sekcích). Díky tagům z těchto knihoven může vývojář vkládat na stránku komponenty uživatelského rozhraní (UI komponenty).[2]

Dle [3] existují dva různé přístupy k tvorbě webových aplikací:

- *Rapid development*, při kterém programátor využívá vizuální vývojové prostředí, jenž mu umožní uživatelské rozhraní "naklikat". Tzn. pouze rozmístit předpřipravené komponenty a doprogramovat jejich funkcionalitu. Příkladem může být ASP.NET od Microsoftu.

- *Hard-core coding*, při kterém programátor píše velké množství kódu manuálně a uživatelské rozhraní se většinou taktéž složitě programuje. Zde bychom mohli jako příklad uvést Java EE platformu.

[1] Znalost technologie JSP a knihoven tagů je předpokladem pro četbu této práce. Pro více informací o těchto technologiích viz [8].

[2] UI komponentami se zde myslí klasické prvky uživatelského rozhraní, na které jste zvyklí z jiných aplikací. Jsou to např. tlačítka, textová pole, různé druhy zaškrtávacích políček (checkboxy, radiobuttony), různé druhy seznamů (listboxy, comboboxy) atd.

JSF framework se snaží především o usnadnění tvorby uživatelského rozhraní. Můžeme ho tedy chápat jako jakýsi *Swing*[3] pro webové aplikace na Java EE platformě. Při využití JSF vývojář umísťuje na stránku předpřipravené komponenty a mapuje je na příslušné třídy zajišťující funkcionalitu.

JSF se skládá ze dvou hlavních součástí:

- **API** [6] zastřešující komponenty uživatelského rozhraní. Umožňuje správu těchto komponent, konverzi a validaci uživatelských vstupů (konverzí a validací se budeme zabývat v sekci 5 – „*Konverze a validace*“), zpracování událostí (zpracováním událostí se budeme zabývat v sekci 6 – „*Zpracování událostí*“), řízení navigace mezi stránkami (navigací mezi stránkami se budeme zabývat v sekci 3.3.2.5 – „Namapování požadavků na instanci `FacesServlet`“) atd. Třídy zajišťující tuto funkcionalitu lze samozřejmě v případě potřeby rozšířit.

- **JSP knihovny tagů** [10] zajišťující přístup k JSF třídám z JSP stránky. Pomocí těchto tagů a jejich atributů umísťujeme UI komponenty na stránku a editujeme jejich chování. Např. je mapujeme na naše třídy, přiřazujeme jim validátory a *listenery* atp. (využitím UI komponent na stránce se budeme zabývat v sekci 4.3 – „Použití UI komponent na stránce“).

3.2. Hlavní výhody JSF

Na následujících řádcích se pokusím nastínit hlavní přínosy, jenž Vám JSF nabídne. Nebudu zde porovnávat JSF s jinými webovými frameworky pracujícími nad Javou,[4] nýbrž spíše upozorním na rozdíly ve vývoji Java EE aplikace s využitím frameworku JSF a bez něj.

Hlavní výhodou JSF je zautomatizování některých rutinních úkonů, které by samotným JSP šly řešit buď velmi těžko, nebo vůbec. Následuje výčet některých z rutin, jejichž vývoj JSF usnadňuje [8].

- Vkládání komponent uživatelského rozhraní na stránku pomocí dostupných knihoven tagů.

[3] Pri více informací o Swingu viz http://java.sun.com/docs/books/tutorial/uiswing.

[4] Webových frameworků pracujících nad Javou je celá řada. Mezi ty populárnější patří např. Struts [http://struts.apache.org], Stripes [http://www.stripesframework.org], Tapestry [http://tapestry.apache.org], Wicket [http://wicket.apache.org] či Spring MVC [http://www.springsource.org]. Jejich srovnáním se zabývá mnoho prací, např. prezentace Matta Raibla [http://static.raibledesigns.com/repository/presentations/ComparingJavaWebFrameworks-ApacheConUS2007.pdf].

- Mapování komponent, které generují události, na serverový kód, jenž tyto události zpracovává.
- Mapování stavu komponent na serverová data.
- Validace a konverze uživatelských vstupů.
- Tvorba znovupoužitelných komponent.
- Ukládání a načítání stavu komponent mezi požadavky.

Každá komponenta uživatelského rozhraní je definována třídou, která determinuje její funkcionalitu, a tzv. **rendererem**, který určuje její výslednou podobu na výstupním zařízení. JSF tedy není vázáno na žádný značkovací jazyk. Výslednou podobou může být HTML, WML či jakýkoli jiný jazyk, podle toho, jak jsou naprogramovány renderovací třídy (pro více informací o renderování komponent viz 4.2 – „Renderování UI komponent“). Jelikož komponenty mohou mít více rendererů, může být jedna komponenta zobrazena jinak na monitoru a jinak třeba na mobilním telefonu. Základní implementace JSF převádí komponenty do HTML kódu.

JSF aplikuje architekturu *Model-View-Controller* (MVC), což ve zkratce znamená striktní oddělení logiky od prezentace.[5] Toho lze za použití čistého JSP dosáhnout pouze z části. V řešitelském týmu, jehož členové mají různé znalosti, se dá díky dodržování MVC architektury dosáhnout efektivnější dělby práce. JSF specifikace definuje pět rolí, které se při vývoji aplikace uplatní [5]. Pro nás jsou však důležité pouze tři z nich:

- ***Page author*** - zodpovídá za tvorbu uživatelského rozhraní. Má znalosti značkovacích jazyků a JSP. Podle specifikace nemusí mít znalosti programování.
- ***Application developer*** - má na starosti funkcionalitu aplikace, tzn. například tvorbu EJB komponent.
- ***Component writer*** - tvoří znovupoužitelné komponenty, které jsou pak na stránku vkládány *page authorem*.

Počet vývojových rolí samozřejmě neříká nic o tom, kolik lidí na projektu pracuje. V jedné vývojové roli může vystupovat mnoho členů týmů a zároveň jedna osoba může zastávat více rolí. Dle [5] se například v případě rolí *page author* a *application developer* jedná většinou o stejnou osobu.

[5] Pro více informací o architektuře MVC viz např. [2].

3.3. První JSF aplikace

Nyní si konečně vytvoříme první velice jednoduchou aplikaci. Uživatel, který navštíví stránku pouze zadá do vstupního pole své jméno a potvrdí ho, načež se mu zobrazí uvítací zpráva se zadaným jménem. Na příkladu budeme demonstrovat, z čeho se JSF aplikace skládá a základní kroky při vývoji. Programový kód a obsah veškerých souborů k této ukázkové aplikace najdete v příloze A. Snímky obou oken aplikace můžete vidět na konci kapitoly na obrázcích 3.5 a 3.6.

Kód první aplikace je umístěn celý v příloze proto, aby bylo možno utvořit si ucelenou představu o koncepci a architektuře JSF. Ve zbytku textu najdete v kapitolách pouze fragmenty kódu, aby byl čtenář nucen psát si další aplikace sám.

3.3.1. Z čeho se JSF aplikace skládá

JSF aplikace je z velké části podobná klasickým webovým aplikacím, má však i svá specifika [8]. Měla by se skládat z následujících komponent (souborů):

- Klasické JSP soubory.
- Množina tzv. ***managed beanů***, což jsou třídy, které dodržují konvence *JavaBeans* modelu[6] a jsou spravovány JSF frameworkem. Jejich konfigurace se provádí v souboru `faces-config.xml` (viz sekce 3.3.2.2 – „Konfigurace *backing beanů*“). Dle funkcionality, kterou *managed beany* zajišťují, rozlišujeme několik druhů. Pro nás budou nejdůležitější tzv. ***backing beany***, jenž zajišťují funkcionalitu UI komponent na stránce. Mohou například udržovat informace o jejich stavu, provádět validaci dat, zpracovávat události atd.[7]
- **Konfigurační soubor JSF aplikace**, ve kterém se definují pravidla pro navigaci mezi stránkami a konfigurují *managed beany*. Jedná se o XML soubor, který má zpravidla název `faces-config.xml`.
- ***Deployment descriptor***, což je také XML soubor, který musí být přítomen v každé webové aplikaci. Nabízí poměrně široké možnosti konfigurace, např. nastavení uvítacích stránek, mapování servletů, filtry, kontextové parametry aj. Obvyklý název souboru je `web.xml`.[8]

[6] Pro více informací o modelu JavaBeans viz http://java.sun.com/docs/books/tutorial/javabeans.

[7] Je důležité neplést si managed beany, které se nacházejí ve webové vrstvě, s enterprise Java beany z business vrstvy. Ty totiž nezajišťují funkcionalitu webových stránek, nýbrž stěžejní funkcionalitu celé enterprise aplikace a navíc se nacházejí v business vrstvě. Pro více informací o EJB viz [8].

[8] Pro více informací o možnostech deployment descritporu viz [8].

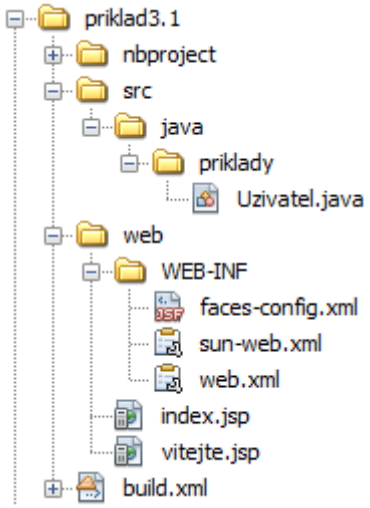
Aplikace může dále obsahovat komponenty vytvořené vývojářem. Může se jednat např. o vlastní UI komponenty, vlastní konvertory či validátory atp. Vývoj těchto komponent však přesahuje rozsah práce.

3.3.2. Postup pří vývoji JSF aplikace

Při tvorbě JSF aplikace musíme postupně vytvořit a naprogramovat všechny výše popsané komponenty. My budeme postupovat následovně:

1. Tvorba *backing beanů*.
2. Konfigurace *backing beanů* v souboru **`faces-config.xml`**.
3. Tvorba JSP stránek za použití tagů JSF.
4. Definování pravidel pro navigaci mezi stránkami v souboru **`faces-config.xml`**.
5. Namapování požadavků na instanci `FacesServlet`.[9]
6. Nasazení aplikace na aplikační server.

Všechny tyto kroky si nyní postupně projdeme a dopracujeme se tak k naší první JSF aplikaci. Pro představu uvádím, jak by měla vypadat její finální adresářová struktura.



Obrázek 3.1. Finální adresářová struktura první JSF aplikace (zdroj: NetBeans)

[9] FacesServlet je třída, jejíž instance řídí celý životní cyklus JSF aplikace. Přes tuto instanci musí projít veškeré requesty. Více se touto třídou budeme zabývat v sekci 3.3.2.5 – „Namapování požadavků na instanci `FacesServlet`“.

3.3.2.1. Tvorba *backing beanů*

Backing bean je třída napsána v jazyce Java, která je provázána s komponentami uživatelského rozhraní na stránce. Zodpovědnost za *backing beany* má osoba, která je znalá v programování v Javě. Dle Java EE specifikací se jedná o *application developera*. *Page author* poté jen páruje komponenty uživatelského rozhraní s příslušnými *backing beany* pomocí speciálního jazyka zvaného ***Unified Expression Language* (EL)**.[10]

V našem příkladu používáme jeden velice jednoduchý *backing bean*, třídu `Uzivatel`. Naleznete v ní jeden ze základních prvků *JavaBeans* modelu, a sice že každá proměnná musí mít svůj *getter* nebo *setter* nebo obojí, podle toho, zda se jedná o proměnnou určenou pro čtení nebo pro zápis.

Příklad 3.1. ***Backing bean*** **`Uzivatel.java`**

```
package priklady;

public class Uzivatel {
    private String jmeno;

    public String getJmeno() {
        return jmeno;
    }

    public void setJmeno(String jmeno) {
        this.jmeno = jmeno;
    }
}
```

3.3.2.2. Konfigurace *backing beanů*

Aby mohly být *backing beany* používány v aplikaci, nestačí je pouze naprogramovat. Je také třeba zadat o nich určité informace do konfiguračního souboru **`faces-config.xml`**. JSF aplikace potřebuje o *beanu* vědět, jak ho pojmenovat, z jaké třídy se má vytvořit a v jakém rozsahu působnosti ho vytvořit. Rozsah působnosti (tzv. *scope*) aplikaci říká, odkud bude instance dostupná a jak dlouho má existovat. Může nabývat čtyř hodnot:

- `application` - přístupnost a životnost v rámci celé aplikace
- `session` - přístupnost a životnost v rámci jedné uživatelské relace

[10] Pro více informací o jazyce Unified Expression Language viz [8].

- `request` - přístupnost a životnost v rámci jediného *requestu*
- `none` - instance takového *beanu* se nikde neukládají, při každém přístupu je vytvořena instance nová

V naší aplikaci jsme zvolili rozsah působnosti `session`.

Příklad 3.2. Konfigurace backing beanu

```
<managed-bean>
    <managed-bean-name>uzivatel</managed-bean-name>
    <managed-bean-class>priklady.Uzivatel</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Tag `managed-bean` ohraničuje konfiguraci jednoho *managed beanu*. Tag `managed-bean-name` udává identifikátor, kterým se budu na *bean* v celé aplikaci odkazovat, `managed-bean-class` říká z jaké třídy *bean* vytvořit a `managed-bean-scope` definuje rozsah působnosti *beanu*. V souboru můžeme také definovat, s jakou hodnotou se mají inicializovat jednotlivé proměnné. Tuto možnost však v naší aplikaci nevyužíváme.

3.3.2.3. Tvorba JSP stránek za použití tagů JSF

JSF soubory mohou samozřejmě obsahovat klasické JSP tagy, HTML tagy a další objekty, které znáte z různých webů, např. klientské skripty. Obsahují však také některé specifické elementy, které v jiných typech souborů nenaleznete. V souboru **`index.jsp`** z naší aplikace se nachází několik základních JSF elementů. Ty rozeznáte na stránce podle toho, že jsou uvedeny předponou, kterou si *page author* zvolí při deklaraci JSF knihoven tagů. Tyto knihovny si nadeklarujeme níže (více se budeme jednotlivými JSF elementy zabývat v sekci 4.3 – „Použití UI komponent na stránce“).

Soubor **`index.jsp`** si nyní postupně projdeme a elementy si vysvětlíme, přičemž se nebudeme zabývat objekty, které nejsou specifické pro JSF, jako např. HTML a JSP tagy. Nejsou totiž předmětem této práce a předpokládá se zde jejich znalost.

Deklarace knihoven JSF tagů:

Pokud chceme na stránce používat JSF komponenty, je na ní třeba nadeklarovat dvě knihovny.[11] Ta první má název ***core tag library*** a obsahuje 20 tagů. Najdeme zde například tag pro ohraničení veškerých JSF komponent na stránce, dále tagy

[11] Problematiku knihoven tagů, byste měli znát z JSP technologie. Pro více informací o knihovnách tagů viz např. [1].

pro validátory, listenery atp. Tagy z této knihovny nejsou závislé na použité renderovací technologii.[12] Výpis tagů z knihovny *core tag library* najdete v příloze B.

Druhá knihovna má název ***HTML component tag library*** a obsahuje zejména tagy pro vkládání UI komponent na stránku, např. tlačítka, formulářová pole atp. Tyto komponenty jsou závislé na použité renderovací technologii. Výpis všech 25 tagů z knihovny *HTML component tag library* najdete v příloze C.[13]

Příklad 3.3. Deklarace knihoven JSF tagů

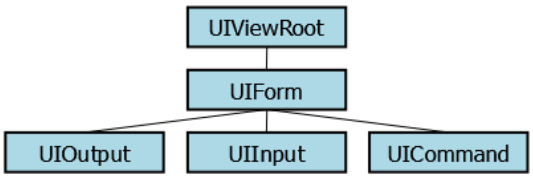
```
<%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>
```

Ve výpise můžete vidět jedinečné URI, pomocí kterých se musíte na tyto knihovny vždy odkazovat, a také prefixy `f` a `h`, pomocí kterých budete uvozovat všechny tagy příslušné knihovny. Přestože si můžete prefixy zvolit libovolně, doporučuji držet se zavedené konvence a používat `f` a `h`.

Tagy `view` a `form`:

Všechny JSF tagy na stránce musí být uzavřeny v tagu `view`. Tento tag se nachází v *core tag library*. `FacesServlet` si poté ze všech tagů uvnitř `view` vytvoří jakýsi strom komponent, pomocí kterého k nim přistupuje.

```
...
<f:view>
   <h:form>
      <h:outputText value="Zadejte sumu: "/>
      <h:inputText value="#{bean.suma}"/>
      <h:commandButton value="Potvrdit" action="success"/>
   </h:form>
</f:view>
...
```



Obrázek 3.2. Strom vytvořený z JSF tagů (zdroj: vlastní; inspirován [5])

Tag `form` je obdoba stejnojmenného HTML tagu. Tento tag patří do *HTML component tag library*. Veškeré JSF komponenty, které umožňují uživateli zadat nějaká

[12] Renderovací technologie představuje souhrn pravidel, která determinují, jak mají být JSF tagy prezentovány na koncovém zobrazovacím zařízení. Jako příklad můžeme uvést tag commandButton, který představuje tlačítko a standardní renderovací třídy JSF ho na koncovém zařízení prezentují pomocí HTML tagu input třeba takto: <input type="submit" value="Potvrdit"/>. Více se renderováním UI komponent budeme zabývat v sekci 4.2 – „Renderování UI komponent“.

[13] Pro více informací o těchto knihovnách a tazích, které obsahují, viz [10].

vstupní data a odeslat je na server musí být umístěny uvnitř tohoto tagu. Mezi takové komponenty patří např. `inputText` či `commandButton`.

Příklad 3.4. Tagy `view` a `form`

```
<f:view>
    <html>
        ...
        <body>
            ...
            <h:form>
                <p>Zadejte vaše jméno: <h:inputText ▶
value="#{uzivatel.jmeno}"/></p>
                <h:commandButton action="ok" value="Potvrdit"/>
            </h:form>
        </body>
    </html>
</f:view>
```

Tagy `outputText`, `inputText` a `commandButton`:

Tyto tři tagy, jenž se nacházejí v naší aplikaci, jsou jedny z mnoha tagů, které můžeme použít pro tvorbu uživatelského rozhraní. Všechny tři patří do *HTML component tag library*.

Tag `outputText` vypisuje na obrazovku text, který je dán atributem `value`. Tento atribut přijímá obyčejný řetězec, ale pomocí jazyka EL do něj můžeme zadat také název proměnné či metody *backing beanu*, ze které řetězec získáme.

Tag `inputText` reprezentuje vstupní pole, do kterého může uživatel zadat textový řetězec. Atributem `value` určujeme hodnotu vstupního pole. V naší aplikaci tento atribut obsahuje výraz jazyka EL, pomocí kterého spojujeme hodnotu vstupního pole s hodnotou proměnné `jmeno` třídy `Uzivatel`. Výrazem před tečkou tedy definujeme třídu a výrazem za tečkou proměnnou. Výraz `uzivatel` se zde však neodkazuje přímo na název třídy, nýbrž na název, který jsme si u příslušného *backing beanu* nastavili v konfiguračním souboru **`faces-config.xml`** (viz sekce 3.3.2.2 – „Konfigurace *backing beanů*“).

Pomocí tagu `commandButton` umístíme na stránku tlačítko, které odesílá data z formulářových polí na server. Atributem `value` zde určujeme popisek, který se na tlačítku objeví, a atributem `action` definujeme akci, kterou stisk tlačítka vyvolá. Opět lze vložit buď prostý řetězec, jako v naší aplikaci, nebo můžeme volat funkci,

která na základě určité operace vrátí řetězec akce. Tento řetězec se pak využívá pro definici pravidel pro navigaci mezi stránkami, která se provádí v konfiguračním souboru **`faces-config.xml`** (definicí pravidel pro navigaci mezi stránkami provedeme v sekci 3.3.2.4 – „Definování pravidel pro navigaci mezi stránkami“).

Příklad 3.5. Tagy `outputText`, `inputText` a `commandButton`

```
<h1><h:outputText value="První JSF aplikace"/></h1>
<h:form>
    <p>Zadejte vaše jméno: <h:inputText value="#{uzivatel.jmeno}"/></p>
     <h:commandButton action="ok" value="Potvrdit"/>
</h:form>
```

V druhém souboru naší aplikace **`vitejte.jsp`** se nenachází žádný JSF tag, který bychom nyní neprobrali. Není proto třeba se tímto souborem zabývat. Snad jen poznamenám, že atribut `value` u tagu `outputText` se odkazuje na tu stejnou proměnnu, do které uživatel na předchozí stránce vložil text. Hodnotu této proměnné, tedy uživatelské jméno, nyní z této proměnné načítáme.

Více se jednotlivými tagy budeme zabývat v sekci 4.3 – „Použití UI komponent na stránce“.

3.3.2.4. Definování pravidel pro navigaci mezi stránkami

Definování pravidel pro navigaci mezi stránkami znamená stanovit pravidla, podle kterých `FacesServlet` jednoznačně pozná, na kterou stránku přejít po tom, co uživatel klikne na tlačítko či URL odkaz na aktuální stránce. V JSF se pro stanovení těchto pravidel používá odlišný způsob, než na jaký jste zvyklí z JSP. Místo psaní klasických URL adres se u každého tlačítka respektive odkazu stanoví pomocí atributu `action` logický výstup operace. Ten má podobu textového řetězce a můžeme ho buď zapsat napevno, nebo vygenerovat nějakou metodou. Stanovení logického výstupu napevno jsme použili v naší aplikaci.

Příklad 3.6. Stanovení logického výstupu operace staticky

```
<h:commandButton action="ok" value="Potvrdit"/>
```

Kdybychom chtěli výstup generovat dynamicky, volaná metoda musí vracet typ `Object` či jakéhokoli potomka. Pro určení řetězce se následně použije metoda `toString()`. Běžnou praxí je vracet rovnou řetězec typu `String`. Tuto metodu pak voláme pomocí EL výrazu. Metoda a její volání ze stránky by mohlo vypadat následovně.

Příklad 3.7. Stanovení logického výstupu operace dynamicky

```
public String vystup() {
    if (objednavka.suma() > 1000) {
        return "sleva";
    } else {
        return "bezSlevy";
    }
}

<h:commandButton action="#{objednavka.vystup}" value="Potvrdit"/>
```

Jak naložit s tímto logickým výstupem se `FacesServlet` dozví z konfiguračního souboru **`faces-config.xml`**. Tam se stanovují pravidla, pomocí kterých lze jednoznačně určit na jakou stránku pokračovat. V konfiguračním souboru naší aplikace vypadá stanovení jediného pravidla následovně:

Příklad 3.8. Definování pravidel pro navigaci mezi stránkami

```
<navigation-rule>
    <from-view-id>/index.jsp</from-view-id>
    <navigation-case>
        <from-outcome>ok</from-outcome>
        <to-view-id>/vitejte.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

Tag `navigation-rule` ohraničuje pravidla pro navigaci z jedné stránky. V našem případě ze stránky **`index.jsp`**. Pokud bychom chtěli přidat pravidla pro další stránku, přidali bychom další tag `navigation-rule`. Tag `navigation-case` zase konfiguruje jeden logický výstup z dané stránky. Tento výstup se nachází v tagu `from-outcome` a následný tag `to-view-id` říká, na jakou stránku v případě daného výstupu pokračovat. Konfigurace více stránek s více možnými výstupy by mohla vypadat následovně:

Příklad 3.9. Definování pravidel pro navigaci mezi stránkami II

```
<navigation-rule>
    <from-view-id>/index.jsp</from-view-id>
    <navigation-case>
```

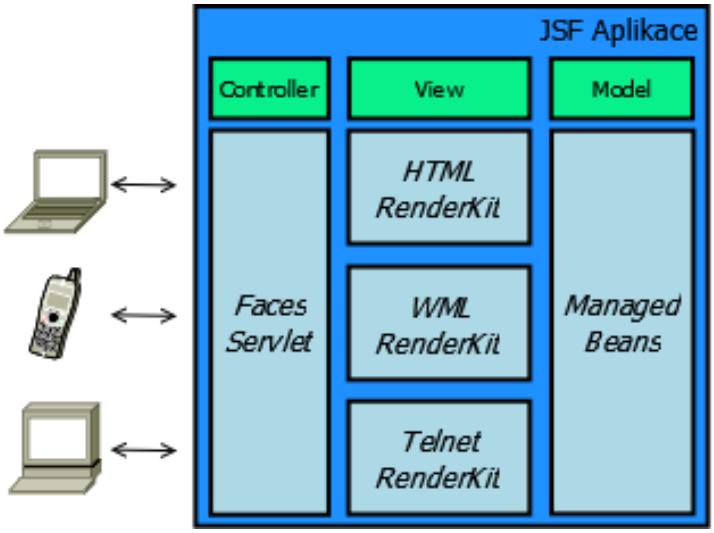
```
            <from-outcome>ok</from-outcome>
            <to-view-id>/vitejte.jsp</to-view-id>
        </navigation-case>
        <navigation-case>
            <from-outcome>chyba</from-outcome>
            <to-view-id>/chyba.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>
    <navigation-rule>
        <from-view-id>/vitejte.jsp</from-view-id>
        <navigation-case>
            <from-outcome>ok</from-outcome>
            <to-view-id>/katalog.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>
```

3.3.2.5. Namapování požadavků na instanci `FacesServlet`

Aby jakákoliv JSF aplikace mohla fungovat, musí splňovat jednu základní podmínku - veškeré požadavky, které na ni směřují, musí jít přes servlet jménem `FacesServlet`.[14] Ten je součástí JSF knihovny umístěné na aplikačním serveru a nachází se v balíčku `javax.faces.webapp`. Instance tohoto servletu, která je vytvořena webovým kontejnerem při prvním přístupu, řídí celý životní cyklus a inicializuje veškeré potřebné zdroje. Z pohledu MVC[15] architektury vystupuje `FacesServlet` v roli *controlleru*.

[14] Pro více informací o třídě FacesServlet viz [6].

[15] V tomto kontextu mám na mysli MVC architekturu z pohledu izolované webové aplikace. Pokud bychom se na enterprise aplikaci o čtyřech vrstvách podívali z perspektivy, MVC model by vypadal jinak.



Obrázek 3.3. MVC model JSF aplikace[16] (zdroj: vlastní; inspirován [5])

Z uvedených důvodů je potřeba namapovat `FacesServlet` na určité URL vzory.[17] Mapování servletů se provádí v souboru *deployment descriptoru* **`web.xml`** a existují dva způsoby, jak ho realizovat - tzv. ***prefix mapping*** a ***extension mapping***.

Prefix mapping (mapování pomocí předpony):

Základem této metody je namapovat v *deployment descriptoru* `FacesServlet` na předponu, kterou si libovolně zvolíme. Tagy pro mapování vypadají následovně:

Příklad 3.10. Mapování `FacesServletu` metodou prefix mapping

```
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

[16] View JSF aplikace je tvořeno tzv. RenderKity, což je sada tříd, které mají na starost renderování UI komponent na výstupní zařízení. Více se budeme renderováním UI komponent zabývat v sekci 4.2 – „Renderování UI komponent“.

[17] URL vzor je řetězec složený ze znaků, které mohou být obsaženy v URL, a zástupných znaků (např. *), které představují libovolný znak. Takovému vzoru pak odpovídá určitá množina URL adres. Pro více informací o URL vzorech viz [8].

Tag `servlet` definuje základní vlastnosti servletu. Důležitější je pro nás v tuto chvíli tag `servlet-mapping`, kde je nakonfigurováno mapování. Nejprve je uvedeno jméno servletu a následně mu přiřazujeme URL vzor s předponou. Jak vidíte, v našem příkladu jsme zvolili předponu `faces`. To znamená, že jakákoliv URL adresa ve tvaru **`http://adresaserveru:port/contextPath/faces/soubor.jsp`**[18] povede na `FacesServlet`. Jako konkrétní příklad můžeme uvést adresu, kterou používám na *localhostu* pro testování naší první aplikace: **`http://localhost:8080/priklad3.1/faces/index.jsp`**. Jelikož je v URL vzoru za příponou `faces` hvězdička, tzn. zástupný znak, mohli bychom soubor **`index.jsp`** nahradit jakýmkoliv jiným existujícím JSP souborem.

Extension mapping (mapování pomocí přípony):
Jak z názvu vyplývá, v případě *extension mappingu* si nevolíme předponu, nýbrž příponu, kterou použijeme na mapování. Pokud bychom zůstali u slůvka `faces`, vypadalo by mapování v souboru **`web.xml`** následovně:

Příklad 3.11. Mapování servletu `FacesServlet` metodou extension mapping

```
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.faces</url-pattern>
</servlet-mapping>
```

Jediná změna oproti příkladu 3.10 nastala v tagu `url-pattern`, kde nyní nepoužíváme *prefix mapping*, nýbrž *extension mapping*.

Webový kontejner pošle v tomto případě jakýkoliv požadavek na soubor s příponou .faces na `FacesServlet`. Ten pak zajistí záměnu naší přípony za příponu .jsp a bude hledat odpovídající soubor. To znamená, že například požadavek **`http://localhost:8080/priklad3.1/vitejte.faces`** bude automaticky přesměrován na `FacesServlet` a ten následně zamění příponu a načte soubor **`vitejte.jsp`**.

[18] Context path je adresář, který identifikuje aplikaci v rámci aplikačního serveru. Pro více informací o context path viz [8].

Nyní, když máme `FacesServlet` jedním ze zmíněných způsobů namapován, máme tři možnosti, jak aplikaci správně spustit:

- Prvním a nejjednodušším způsobem je napsat správnou URL adresu do adresního řádku našeho prohlížeče. Jako příklad mohou posloužit URL adresy, které jsem uvedl u obou způsobů mapování. Nevýhodou tohoto řešení je požadavek na uživatele, který musí znát správnou předponu respektive příponu, kterou vývojář zvolil.

- Druhá možnost je použít jako uvítací stránku[19] JSP soubor, který bude ve svém těle obsahovat pouze tag pro přesměrování na správnou URL adresu, která spouští `FacesServlet`. Tento tag by mohl v případě *prefix mappingu* vypadat takto: `<jsp:forward page="faces/welcome.jsp"/>`. V případě *extension mappingu* zase takto: `<jsp:forward page="index.faces"/>`. Uživatel v tomto případě nepotřebuje znát předponu respektive příponu. Když opět použijeme náš příklad na *localhostu*, tak stačí, aby uživatel zadal **`http://localhost:8080/priklad3.1/`**. Automaticky se tak načte nastavená uvítací stránka, které si však ani nevšimneme, neboť nás ihned přesměruje na JSF stránku přes `FacesServlet`.

- Poslední možností je nakonfigurovat v *deployment deskriptoru* URL uvítací stránky již s předponou. Konfigurace v souboru **`web.xml`** by vypadala následovně:

Příklad 3.12. Nastavení uvítací stránky

```
<welcome-file-list>
    <welcome-file>faces/index.jsp</welcome-file>
</welcome-file-list>
```

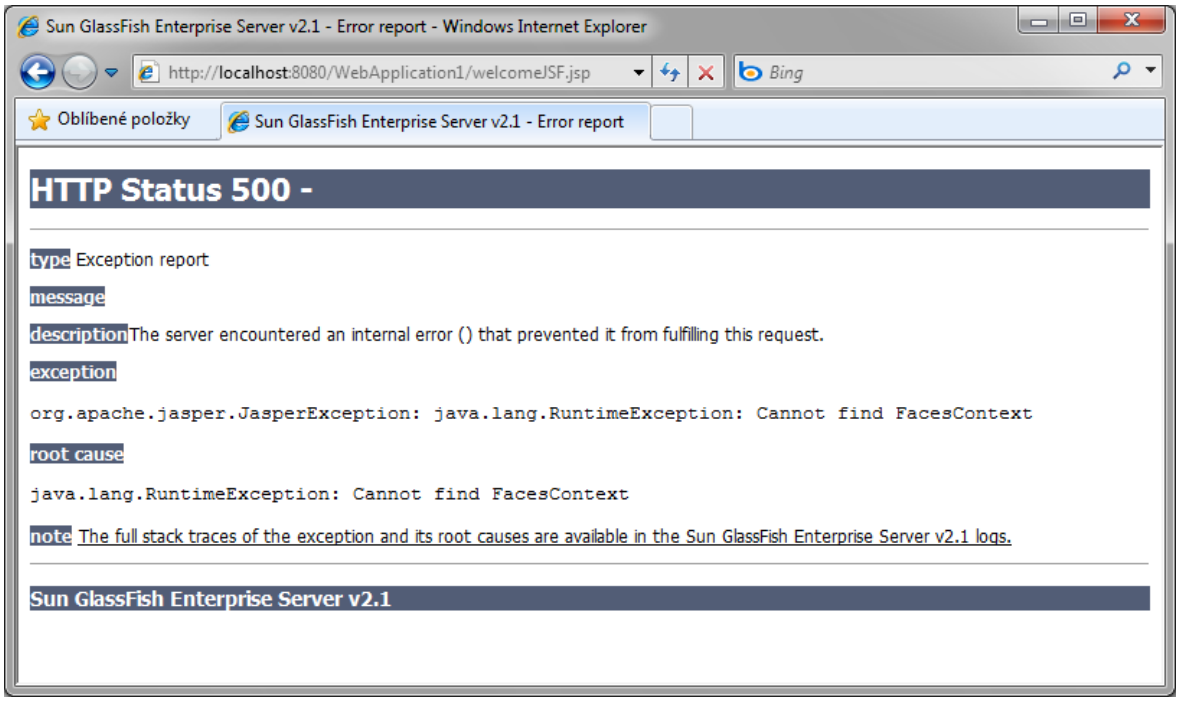
Příponu jsem zde nezmínil záměrně, neboť toto řešení v kombinaci s *extension mappingem* nepřichází v úvahu. Uvítací stránka totiž musí být fyzicky existující soubor, což např. `index.faces` není. Je to pouze pseudonázev, se kterým si umí poradit `FacesServlet`, ale webový kontejner by takový soubor nedokázal identifikovat.

Při aplikaci tohoto řešení by uživatel opět nemusel znát předponu, a ani by nedocházelo k přesměrování. Tento způsob je použit také v naší JSF aplikaci.

Možná se nyní ptáte, co by se stalo, kdyby se uživatel pokusil zadat do prohlížeče URL, které neodpovídá žádnému vzoru a požadavek by tak neprošel přes `FacesServlet`. V takovém případě by nebylo zajištěno korektní započetí životního cyklu aplikace a server by vyhodil výjimku `RuntimeException` s chybovým hlášením, že `FacesContext` nebyl nalezen. Instance třídy `FacesContext` je inicializovaná při každém

[19] Uvítací stránka aplikace se nastavuje v souboru web.xml pomocí tagu welcome-file-list (viz 3.3).

požadavku a uchovává veškeré potřebné informace, které se k němu vztahují (pro více informací o třídě `FacesContext` viz sekci 7 – „*Životní cyklus JSF aplikace*“ nebo [6]).



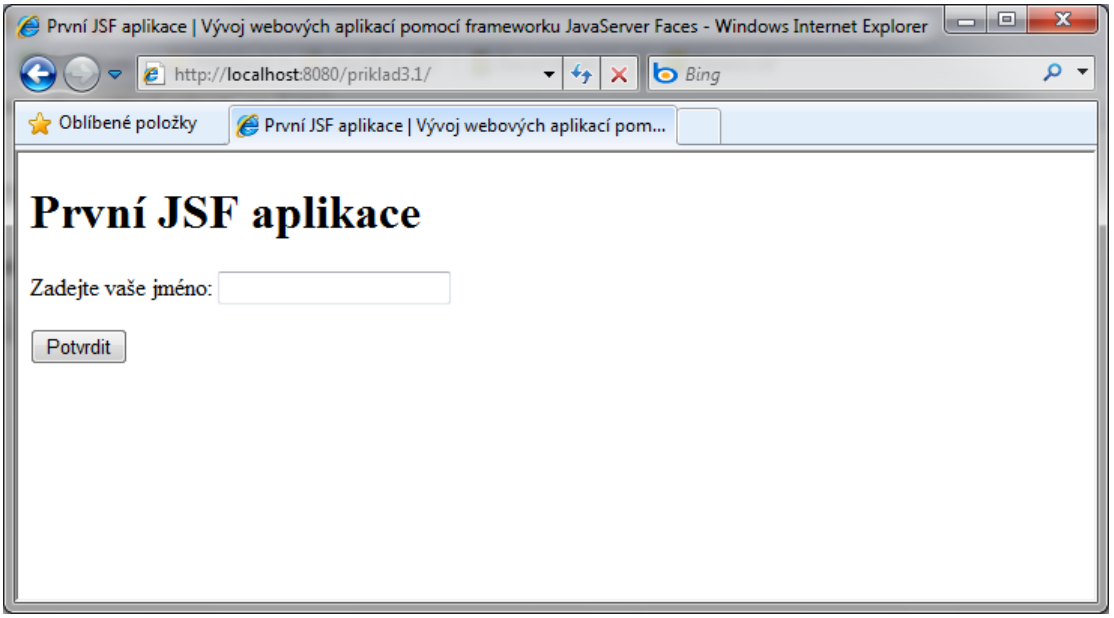
Obrázek 3.4. Chybová hláška při přeskočení `FacesServletu`

3.3.2.6. Nasazení aplikace na aplikační server (*deployment*)

Deployment je poslední krok, který musíme provést předtím, než naší aplikaci konečně spustíme. Na server je třeba nasadit soubor s příponou .war, což je klasický Java archiv se strukturou specifickou pro Java EE webové aplikace.[20] Přesný postup nasazení se však může u různých aplikačních serveru lišit. Z toho důvodů, a také proto, že se nejedná o nijak složitý proces, Vás odkážu na dokumentaci k serveru, který používáte. Pokud však používáte aplikační server GlassFish, naleznete jednoduchý postup v příloze E.

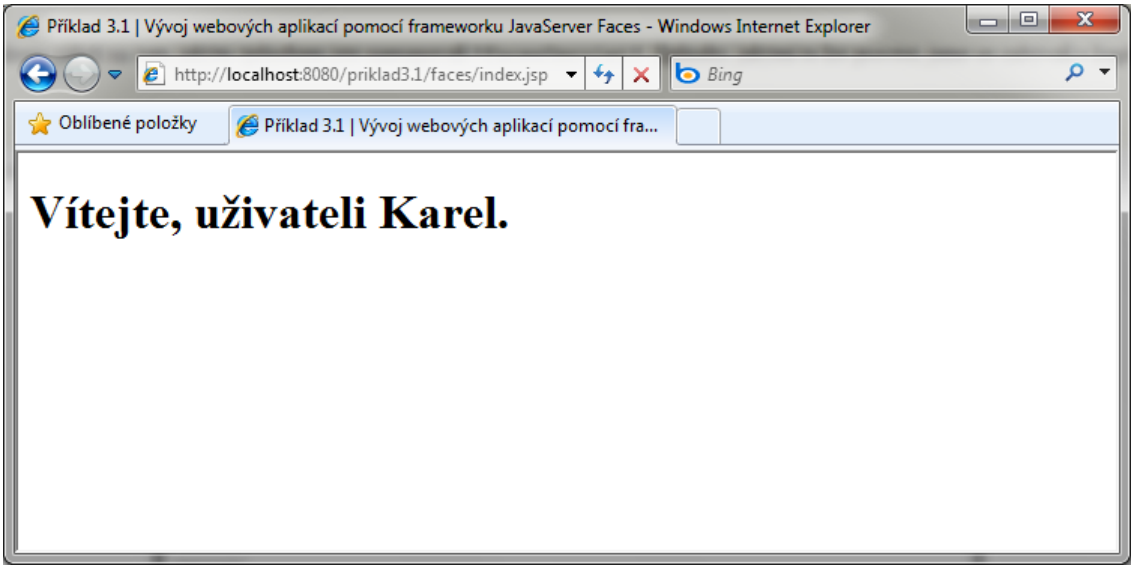
Když se Vám povede aplikaci správně nasadit, stačí do prohlížeče zadat URL adresu a měla by se spustit. V případě naší aplikace nasazené na GlassFish byste po zadání adresy **`http://localhost:8080/priklad3.1/`**[21] měli vidět přibližně to, co vidíte na obrázku 3.5.

[20] Pro více informací o Java EE archivech viz [8].
[21] Tvar URL adresy záleží na tom, jakým způsobem jste namapovali FacesServlet.



Obrázek 3.5. Úvodní stránka naší JSF aplikace

Po zadání jména a potvrzení byste pak měli vidět to, co vidíte na obrázku 3.6.



Obrázek 3.6. Druhá stránka naší JSF aplikace

3.4. Příklad k procvičení



Vytvořte aplikaci, jejíž úvodní stránka bude obsahovat vstupní textové pole a dvě tlačítka. Jedno tlačítko bude mít popisek "Jsem plnoletý" a druhé "Nejsem plnoletý". Uživatel

zadá své jméno a podle svého věku klikne na jedno z tlačítek. Následně bude přesměrován na stránku s uvítací hláškou, která bude obsahovat oslovení a informaci o přístupném obsahu, podle toho, zda je uživatel plnoletý či ne. Např. "Vítejte uživateli Jan. Veškerý obsah našich stránek je Vám k dispozici." nebo "Vítejte uživateli Jan. Litujeme, ale obsah našich stránek je pro Vás cenzurován.". Každé tlačítko bude generovat jiný logický výstup a povede tak na jinou JSP stránku.

Kapitola 4

UI komponenty

Již víme, že primárním cílem JSF frameworku je usnadnit tvorbu uživatelského rozhraní. To je v převážné většině aplikací sestaveno ze standardních komponent, jako jsou tlačítka, textová pole, *checkboxy*, *comboboxy* atp. V JSF se tyto komponenty vkládají na stránku pomocí 25 tagů z knihovny *HTML component tag library*. Jejich výčet najdete v příloze C. V této kapitole se nejprve seznámíme se základními principy komponent, dále si vysvětlíme způsob jejich renderování a nakonec si uděláme přehled základních skupin komponent včetně jejich krátké charakteristiky.

Samozřejmě zde nebude prostor, abychom se podrobně zabývali funkcionalitou a možnostmi všech komponent. Proto Vás budu průběžně odkazovat na externí zdroje. Již nyní Vám mohu doporučit [10], kde najdete podrobné informace o každém tagu.

4.1. Základní principy UI komponent modelu

Každá UI komponenta v JSF má dvě stránky - funkcionalitu a vzhled. Funkcionalita komponent je definována třídami zvanými ***UI component classes***. Jsou v nich implementovány funkce jako např. udržování stavu mezi požadavky, reakce na události, řízení konverze a validace atp. Tyto třídy přitom vůbec neřeší, jak bude komponenta vyobrazena na koncovém zařízení. Jejich prezentaci mají na starost třídy zvané **renderery** (viz následující sekci). Následuje výčet a stručný popis všech tříd UI komponent, které naleznete v referenční implementaci JSF v balíčku `javax.faces.component`.[1]

- `UIColumn` - sloupec v tabulce, zpravidla se vyskytuje uvnitř `UIData` komponenty.
- `UICommand` - komponenta, která spouští akci.
- `UIData` - tabulka uspořádaných dat, spolupracuje s instancí třídy `DataModel`.

[1] Pro bližší informace o třídách UI komponent viz [6].

- `UIForm` - formulář obsahující vstupní pole a potvrzovací tlačítko.
- `UIGraphic` - obrázek.
- `UIInput` - přijímání vstupu od uživatele.
- `UIMessage` - zpráva pro uživatele.
- `UIMessages` - skupina zpráv pro uživatele.
- `UIOutput` - výstup dat na stránce.
- `UIPanel` - řídí rozložení komponent na stránce.
- `UIParametr` - udává parametr v rámci svého rodiče.
- `UISelectBoolean` - dává uživateli možnost binární volby (ano/ne).
- `UISelectItem` - může se vyskytovat uvnitř `UISelectMany` nebo `UISelectOne`, představuje položku seznamu.
- `UISelectItems` - stejné jako `UISelectItem`, představuje však celou sadu položek.
- `UISelectMany` - dává uživateli možnost vybrat současně několik položek z množiny.
- `UISelectOne` - dává uživateli možnost vybrat právě jednu položku z množiny.
- `UIViewRoot` - kořen pomyslného stromu komponent.

Všechny tyto třídy jsou potomky abstraktní třídy `UIComponentBase`, která definuje základní charakteristiky společné pro všechny komponenty. V referenční implementaci JSF najdeme povětšinou jednoduché standardní komponenty, které jsou potřeba snad ve všech aplikacích, např. *buttony*, vstupní pole, *checkboxy*, *radiobuttony* atd.

Pokud však někdy budete potřebovat trochu složitější nebo specifickou komponentu, věřte, že všechny třídy komponent jsou rozšiřitelné jako každá jiná javovská třída. Můžete si tak naprogramovat libovolnou komponentu dle vašich potřeb. Jistě existuje také mnoho složitějších komponent, které již byly implementovány někým jiným, a jsou volně dostupné na Internetu. Jako příklad můžeme uvést komponentu kalendáře, která není obsažena v referenční implementaci a může se Vám mnohdy hodit. Naleznete ji třeba na stránce **`http://www.metacube.net/jsf/calendar`**.

Tvorba vlastních uživatelských komponent je však bohužel mimo rozsah této práce. Pro bližší informace o tvorbě uživatelských komponent viz např. [3] nebo [5].

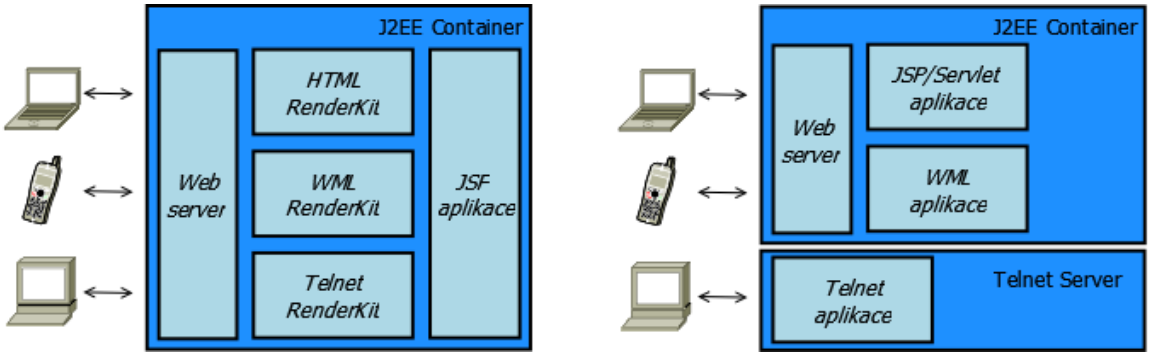
Z hlediska týmových rolí jsou podrobné znalosti o UI component třídách užitečné pouze pro roli *component author*. *Page author* pak jen vkládá na stránku příslušné JSP tagy, které se na třídy komponent odkazují.

4.2. Renderování UI komponent

V předchozí sekci jsme zmínili, že UI komponenty v JSF mají striktně oddělenou logiku od prezentace. Zatímco funkcionalita je definována ve třídách, které jsme vypsali výše, prezentaci komponenty zajišťují naprosto nezávislé třídy zvané renderery.

Renderer má za úkol převod JSF tagů na vhodný značkovací jazyk, který je srozumitelný pro dané výstupní zařízení, přičemž nejsme při výběru výstupního jazyka ničím omezováni. Je možné napsat si renderery pro HTML, WML, SVG, XUL aj. Sada rendererů, které mají stejný výstupní jazyk, se pak nazývá ***RenderKit***. Referenční implementace JSF obsahuje pouze tzv. ***Standart HTML RenderKit***,[2] který, jak z názvu vyplývá, umí převádět tagy pouze do HTML. Další RenderKity mohou být obsažený v jiných implementacích JSF nebo si je můžeme napsat sami. Tvorba vlastních rendererů je však bohužel mimo rozsah této práce. Pro bližší informace o tvorbě vlastních rendererů viz např. [5].

Tento model přináší vývojářům několik výhod. Jednou z nejzásadnějších je to, že celou aplikaci můžeme napsat pouze jednou a s využitím několika *RenderKitů* může fungovat na více výstupních zařízeních. Bez využití JSF by muselo být napsáno několik dílčích aplikací.

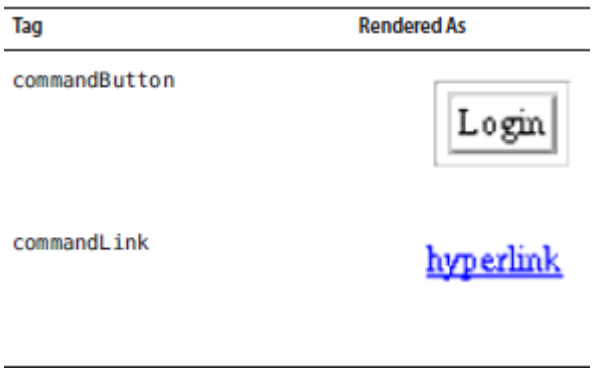


Obrázek 4.1. Architektura aplikace s a bez využití JSF (zdroj: vlastní; inspirován [5])

UI komponenty je možno nejen prezentovat na různá zařízení, ale i prezentovat různými způsoby na jednom zařízení. Jinými slovy, každá třída UI komponenty může mít v jednom *RenderKitu* přiřazeno více než jeden renderer a každý z těchto rendererů

[2]Pro více informací o Standard HTML RenderKit viz [9].

ji může prezentovat jiným způsobem. Jako příklad můžeme uvést třídu UI komponenty `UICommand`, která má v *HTML RenderKitu* přiřazeny dva renderery - `Button` a `Link`. Třída `UICommand` tak může být na HTML stránce prezentována buď jako tlačítko nebo jako hypertextový odkaz.



Obrázek 4.2. Dva způsoby zobrazení třídy `UIColumn` na HTML stránce (zdroj: [8])

Page author tak může měnit vzhled stránky volbou různých JSF tagů, aniž by se muselo zasahovat do kódu logiky. Každý JSF tag tedy představuje jednu konkrétní kombinaci třídy UI komponenty a rendereru. To se odráží také v pojmenování tagů. Na obrázku 4.2 si všimněte, že název tagu je složen ze dvou částí - první část udává třídu UI komponenty (*command*) a druhá část třídu rendereru (*Button* respektive *Link*). V příloze D naleznete souhrnný výpis tříd UI komponent[3] a k nim přiřazených rendererů z *HTML RenderKitu*. Tabulka obsahuje pro lepší představu i obrázky.

4.3. Použití UI komponent na stránce

Nyní si jednotlivé UI komponenty roztřídíme do několika skupin, které si postupně představíme. Opakuji, že zde nebudu komponenty popisovat dopodrobna. Bude se jednat pouze o stručný přehled a některé komponenty, které pokládám za méně důležité, budou zcela opomenuty. Proto využívejte i [10].

4.3.1. Základní atributy tagů

Jako první si představíme základní atributy JSF tagů [8]. Tyto atributy jsou pro většinu z nich společné.

[3] V tomto výpise nenajdete všechny třídy UI komponent, které jsme uvedli v sekci 4.1 – „Základní principy UI komponent modelu“. Důvodem je fakt, že některé komponenty nemá smysl na stránce renderovat, neboť mají pouze řídící funkci. Např. třída UIViewRoot, zastoupena tagem view slouží pouze jako kořen pomyslného stromu komponent, třída UIParam zase pouze deklaruje proměnnou v rámci svého rodiče atp.

- `id` - jedinečný identifikátor komponenty.
- `rendered` - udává podmínku, která musí být splněna, aby se komponenta na stránce zobrazila.
- `style` - definuje kaskádový styl pro komponentu.
- `styleClass` - definuje třídu, do které je komponenta zařazena v rámci kaskádových stylů.
- `value` - připojuje ke komponentě datovou hodnotu.
- `immediate` - může být použit buď u `UICommand` nebo u `UIInput` komponent a ovlivňuje životní cyklus aplikace. Tímto atributem se budeme zabývat až v sekci 7.7.1 – „Ovlivňování životního cyklu pomocí atributu `immediate`“.

Všechny tyto atributy kromě `id` přijímají výrazy jazyka EL. Existuje samozřejmě mnoho dalších atributů specifických pro konkrétní komponenty. O těch důležitých se případně zmíním v sekcích, které se danou komponentou zabývají.

K JSF tagům můžeme přidávat také některé atributy z HTML, např. `alt`, `border`, `size` atd. Pokud chceme na stránku umístit klientský skript, můžeme využít také atributy událostí z DHTML, např. `onclick`, `onmousedown`, `onfocus` atd. Tyto atributy jsou pak z JSF tagů do HTML podoby převedeny beze změny. Tak např. JSF tag `<h:inputText value="#{user.name}" size="5" onfocus="nazevFunkce()"/>` vygeneruje `<input type="text" size="5" onfocus="nazevfunkce()"/>` [3].

4.3.2. Formuláře

Formuláře představují ve webových aplikacích jeden z nejdůležitějších prvků. Slouží obecně ke komunikaci s uživatelem. S tagem formuláře jsme se již setkali v naší první JSF aplikaci. Pro připomenutí se můžete podívat na příklad 3.4. Jak vypadá jednoduchý formulář na stránce, můžete vidět na obrázku 3.5.

Hlavním rozdílem oproti HTML tagu formuláře je, že formulář v JSF nemá atribut `method` ani `action`. V JSF se totiž vždy používá metoda `POST` a všechny formuláře jsou směrovány na stejnou stránku, odkud byly odeslány. Přechod na požadovanou stránku je proveden až následně na základě pravidel pro navigaci, které jsou definovány v souboru **`faces-config.xml`** (navigaci mezi stránkami jsem vysvětlil v sekci 3.3.2.4 – „Definování pravidel pro navigaci mezi stránkami“).

Z důležitých atributů specifických pro formulář bych rád zmínil atribut `prependId`. Pokud je tato hodnota nastavena na *true* (defaultní hodnota), je *id* formuláře automaticky

zahrnuto do *id* každého z jeho prvků. Tak např. formulář má *id* `loginForm` a obsahuje dvě textová pole, která mají *id* `jmeno` a `heslo`. Při hodnotě `prependId="true"` budou názvy formulářových polí v HTML kódu `loginForm:jmeno` a `loginForm:heslo`. Pokud nastavíte `prependId="false"`, *id* formulářových polí si zachovají své původní názvy. Toto je třeba vzít v úvahu, pokud chcete na stránce použít klientský skript, ve kterém se odkazujete na formulářová pole. Musíte si být vědomi, jak jsou jednotlivá pole v konečném HTML kódu pojmenována.

4.3.3. Textové prvky

Textové prvky je možné rozdělit na vstupní a výstupní. Mezi vstupní prvky můžeme zařadit:

`inputText`
Tento tag představuje obyčejné textové pole, se kterým jsme se už také setkali v naší první aplikaci (viz příklad 3.5 nebo obrázek 3.5). Z atributů zde můžeme zmínit `readonly`, který když nastavíme na *true*, hodnota pole nepůjde změnit. Atribut `value` zde přijímá buďto řetězec, nebo EL výraz odkazující na proměnnou *backing beanu*.

`inputSecret`
Podobný tag jako `inputText`, s tím rozdílem, že se nezobrazuje text, nýbrž hvězdičky. Používá se samozřejmě na hesla. Užitečný je atribut `redisplay`, který udává, zda si má komponenta udržovat svou hodnotu mezi požadavky či ne.

Příklad 4.1. Tag `inputSecret`

```
<tr>
    <td>Heslo:</td>
    <td><h:inputSecret id="heslo" value="#{user.heslo}" ▶
redisplay="false"/></td>
</tr>
```

`inputTextarea`
Tento tag představuje vstupní pole pro delší víceřádkový text. Užitečné atributy jsou `cols` a `rows`, které udávají výšku a šířku pole. Jinak funguje taky stejně jako `inputText`.

Příklad 4.2. Tag `inputTextarea`

```
<tr>
    <td>Vlastnosti:</td>
    <td><h:inputTextarea id="vlastnosti" value="#{user.vlastnosti}" ▶
cols="15" rows="6"/></td>
</tr>
```

U těchto tagů se také často využívá atribut `required`, který udává, zda je nutné pole vyplňovat (přijímá hodnoty *true* a *false*). Atribut `requiredMessage`, pak udává zprávu, která se zobrazí v případě, že uživatel povinné pole nevyplnil. Tento atribut je přístupný také u některých dalších tagů, např. u některých selekcí (viz 4.3.5 – „Selekce“). Dále je možno nastavit těmto komponentám validátor či konvertor pomocí atributů `validator` a `converter` (pro více informací o konverzi a validaci viz 5 – „*Konverze a validace*“).

Mezi výstupní textové prvky můžeme zařadit:

`outputText`
Tento tag představuje nejjednodušší výstupní text. V základním tvaru generuje na HTML stránku čistý text, bez jakéhokoliv tagu. Pokud použijeme atribut `style`, vygeneruje se HTML tag `span`. Opět se můžete podívat na již zmíněný příklad 3.5.

`outputFormat`
Tag `outputFormat` umí vypsat text s parametry deklarovanými uvnitř svého těla pomocí tagů `param` z knihovny *core tag library*.

Příklad 4.3. Tag `outputFormat`

```
<h:outputFormat value="{0} má rád {1}.">
    <f:param value="Jirka"/>
    <f:param value="fotbal"/>
</h:outputFormat>
```

Uvedený příklad bude mít za následek vypsání textu "Jirka má rád fotbal".

U obou těchto tagů je poměrně důležitý atribut `escape`. Pokud je nastaven na *true*, automaticky se mění všechny znaky <, > a *&* na příslušné entity. Můžeme tak zabránit útokům typu *cross-site scripting*.

4.3.4. Tlačítka a odkazy

Tlačítka a odkazy jsou další elementy, které naleznete snad v každé webové aplikaci. Slouží samozřejmě pro navigaci mezi stránkami a také pro potvrzování formulářů.

V JSF existují pro tlačítka respektive odkazy tři elementy - `commandButton` a `commandLink`, za kterými stojí třída `UICommand`, a `outputLink`, za kterým stojí třída `UIOutput`. Mezi *command* tagy a *output* tagem je samozřejmě zásadní rozdíl. Tagy `commandButton` a `commandLink` vyvolávají akci a spouští cyklus JSF aplikace (pro více informací o životním cyklu JSF aplikace viz 7 – *„Životní cyklus JSF aplikace“*). O tom, na jakou následující stránku se bude aplikace ubírat, zde rozhoduje atribut `action` a pravidla definována v souboru **`faces-config.xml`**. Naproti tomu tag `outputLink` nijak cyklus JSF aplikace neovlivňuje a slouží jako prostý odkaz na zdroj uvedený v atributu `value`.

`commandButton`
S tímto tagem jsme se také již setkali v naší první JSF aplikaci (viz příklad 3.5). Generuje klasické tlačítko pro potvrzení formulářových dat.[4]

`commandLink`
Tento tag funguje téměř stejně jako `commandButton`. Aby však mohl odkaz vyvolávat akce stejně jako tlačítko, je na HTML výstup generován klientský skript, který tuto možnost zajišťuje. Aktivní prvky, které slouží jako odkaz, musíme umístit do těla tagu. V následujícím příkladu může uživatel kliknout jak na text, tak i na obrázek. V obou případech se odkaz aktivuje.

Příklad 4.4. Tag `commandLink`

```
<h:commandLink action="registrace">
    <h:outputText value="Registrovat"/>
    <h:graphicImage value="/registrace.jpg"/>
</h:commandLink>
```

Oba výše uvedené tagy přijímají dva důležité atributy - `action` a `actionListener`. Atribut `action` udává logický výstup v podobě textového řetězce, podle něhož se aplikace rozhoduje, na jakou stránku po aktivaci komponenty pokračovat. Můžeme uvést buďto přímo textový řetězec, nebo odkaz na metodu *backing beanu*, která na základě nějakého rozhodovacího procesu textový řetězec vrací (viz příklad 3.7). V atributu

[4] To platí pouze v případě, že je atribut type nastaven na submit, což je také defaultní hodnota. Další možná hodnota je reset. V tom případě tlačítko pouze nuluje všechna pole formuláře.

`actionListener` voláme metodu *backing beanu*, která obslouží akci vyvolanou při aktivaci komponenty (pro více informací o zpracování událostí viz 6 – „*Zpracování událostí*“).

`outputLink`

Jak jsme již zmínili, jedná se o obyčejný odkaz, který nijak neovlivňuje JSF aplikaci. Atribut `value` zde slouží pro zadání URL, na kterou odkaz povede. Stejně jako u tagu `commandLink` umisťujeme aktivní prvky do těla tagu.

Příklad 4.5. Tag `outputLink`

```
<h:outputLink value="http://www.google.com">
    <h:outputText value="Přejít na stránky Googlu"/>
</h:outputLink>
```

4.3.5. Selekce

Pro tvorbu selekcí máme v JSF k dispozici sedm tagů, které se dělí do třech skupin - tagy pro výběr více prvků z množiny, tagy pro výběr jednoho prvku z množiny a jeden tag pro binární volbu.

`selectBooleanCheckbox`

Toto je právě zmíněný tag pro binární volbu. Vytvoří jediný *checkbox*, který uživatel může buď zaškrtnout, nebo nechat prázdný. Atribut `value` musí odkazovat na proměnnou *backing beanu*, která je typu `Boolean`.

Příklad 4.6. Tag `selectBooleanCheckbox`

```
<h:selectBooleanCheckbox value="#{user.prijimatEmaily}"/>
```

`selectManyListbox`

Vytvoří *listbox*, ze kterého může uživatel vybrat libovolné množství položek. Jednotlivé položky se udávají buď v několika vnořených tazích `selectItem`, nebo v jednom tagu `selectItems`, jehož parametr `value` může odkazovat na kolekci, mapu, pole objektů či samostatný objekt typu `SelectItem`. Oba tyto tagy jsou z *core tag library*.

Příklad 4.7. Tag `selectManyListbox`

```
<h:selectManyListbox value="#{uzivatel.zaliby}">
    <f:selectItems value="#{user.seznamZalib}"/>
</h:selectManyListbox>
```

Objekt typu `SelectItem` má dvě hlavní vlastnosti - `value` a `label`. `Label` je popisek typu `String`, `value` však může být libovolného typu, neboť přijímá typ `Object` (viz příklad níže). Z datového typu, který si zde zvolíte, musíte následně vycházet při volbě datového typu, do kterého budete ukládat výsledek uživatelského výběru. Tato proměnná se udává v atributu `value` (viz příklad výše - `#{uzivatel.zaliby}`). V našem příkladu máme položky označené čísly, proměnná `zaliby` tedy musí být typu `Integer` respektive `int`. A jelikož se jedná o selekci s možností vícenásobného výběru, musí se jednat o pole. Samozřejmě musí být přítomen *getter* a *setter*. U selekcí bez možnosti vícenásobného výběru by se jednalo o obyčejnou proměnnou. Tento princip funguje ve všech následujících kolekcích

Příklad 4.8. Deklarace pole v *backing beanu* `uzivatel`

```
private int[] zaliby;
...
private SelectItem[] polozkyZalib = {
    new SelectItem(1, "sport"),
    new SelectItem(2, "četba"),
    new SelectItem(3, "zahrádkářství")
};
```

`selectManyCheckbox`

Funguje stejně jako tag `selectManyListbox`, jen místo *listboxu* vykresluje *checkboxy*, ze kterých se rovněž dá vybrat více hodnot. Užitečný je atribut `layout`, který říká v jakém směru se budou prvky seznamu vykreslovat. Může nabývat dvou hodnot - `lineDirection` (vykresluje prvky horizontálně) a `pageDirection` (vykresluje prvky vertikálně). V tomto příkladu jsme pro naplnění seznamu položkami použili několik tagů `selectItem`.

Příklad 4.9. Tag `selectManyCheckbox`

```
<h:selectManyCheckbox value="#{uzivatel.barvy}" layout="pageDirection">
    <f:selectItem itemValue="1" itemLabel="červená"/>
    <f:selectItem itemValue="2" itemLabel="modrá"/>
    <f:selectItem itemValue="3" itemLabel="bílá"/>
</h:selectManyCheckbox>
```

selectManyMenu
Opět funguje naprosto stejně jako předchozí dva tagy. Vykresluje *combobox* s možností vícenásobného výběru. Tento tag se v praxi příliš nepoužívá a některé prohlížeče mají s jeho zobrazováním problémy. Proto doporučuji používat raději `selectManyListbox`.

selectOneRadio
Vykresluje skupinu *radiobuttonů*, ze kterých lze vybrat právě jednu položku. Jinak funguje stejně jako předchozí tagy. Stejně jako u tagu `selectManyCheckbox` i zde lze uplatnit atribut `layout`.

selectOneListbox
Stejná funkce jako `selectManyListbox` s tím rozdílem, že lze vybrat pouze jedinou položku.

selectOneMenu
Vykresluje *combobox*, ze kterého lze vybrat právě jednu položku. Jinak funguje stejně jako předchozí tagy.

4.3.6. Zprávy

Během životního cyklu aplikace se mohou nakumulovat nějaké zprávy, které jsou spravovány instancí třídy `FacesContext` (tuto třídu jsem již zmínil v sekci 3.3.2.5 – „Namapování požadavků na instanci `FacesServlet`“). Většinou se jedná o zprávy informující o chybě konverze či validace. Při renderování můžete tyto zprávy na stránce vykreslit a to pomocí dvou tagů:

message
Tento tag vykreslí pouze jednu zprávu vztahující se ke konkrétní komponentě. Tu určíme pomocí atributu `for`, do kterého zadáme *id* komponenty.

Příklad 4.10. Tag `message`

```
<h:message for="jmeno" style="color: red;"/>
```

messages
Tag `messages` vykreslí veškerý seznam zpráv, které `FacesContext` během celého životního cyklu nakumuloval. Pomocí atributu `layout`, který může nabývat buď hodnoty `list`, nebo `table`, určíme, zda se mají zprávy vykreslit v HTML tabulce či HTML seznamu.

Příklad 4.11. Tag `messages`

```
<h:messages style="color: red;" layout="list"/>
```

Používání těchto dvou tagů je velice důležité, aby uživatel věděl, proč se mu po potvrzení formuláře zobrazuje znovu ta stejná stránka. Ve zprávách byste měli uživatele informovat o tom, jaké chyby se dopustil a jaké podmínky musí vstupní data splňovat.

4.3.7. Panely

V JSF existují dva tagy, které Vám pomáhají při tvorbě *layoutu* stránky.

`panelGrid`
K definici *layoutu* používá obyčejnou tabulku. Pomocí atributu `columns` můžeme určit počet sloupců (defaultní hodnota je 1). Dovnitř tagu následně vkládáme komponenty, které se řadí na řádku zleva doprava a po sloupcích shora dolů. Počet řádků tedy není třeba určovat. Nastaví se automaticky podle počtu komponent, které do tagu vložíme.

Příklad 4.12. Tag `panelGrid`

```
<h:panelGrid columns="2">
    <h:outputText value="#{msgs.namePrompt}:"/>
    <h:inputText id="jmeno" value="#{user.jmeno}" required="true"/>
    <h:outputText value="#{msgs.agePrompt}:"/>
    <h:inputText size="3" value="#{user.vek}"/>
</h:panelGrid>
```

`panelGroup`
Tag panelGroup seskupí několik prvků do jedné skupiny. Používá k tomu buďto HTML tag `div`, či `span`, v závislosti na hodnotě atributu `layout`.

Příklad 4.13. Tag `panelGroup`

```
<h:panelGroup layout="block">
    <h:inputText id="jmeno" value="#{user.jmeno}">
    <h:message for="jmeno"/>
</h:panelGroup>
```

4.3.8. Ostatní

Zbývají dva tagy, které se nehodily do žádné z předchozích skupin. Jedná se o tag pro tabulku a tag pro obrázek.

`graphicImage`
Velice jednoduchý tag, který vykreslí obrázek pomocí HTML tagu `img`. Atribut `value` udává cestu k souboru obrázku.

Příklad 4.14. Tag `graphicImage`

```
<h:graphicImage value="obr1.png"/>
```

`dataTable`
Smysluplný popis možností JSF tabulky by byl pro tuto práci příliš rozsáhlý. Proto se pro více informací o tabulkách podívejte např. do [8] či [3].

4.4. Příklad k procvičení



Vytvořte stránku s formulářem, do kterého bude uživatel zadávat informace o sobě. Formulář bude obsahovat následující prvky:

- *inputText pro zadání uživatelského jména*
- *inputSecret pro zadání hesla*
- *inputTextarea, kde uživatel napíše něco o sobě*
- *selectBooleanCheckbox, kde uživatel zvolí, zda chce dostávat reklamní emaily*
- *selectManyCheckbox, kde uživatel označí své koníčky*
- *selectOneMenu, kde uživatel vybere svou oblíbenou barvu*
- *selectOneRadio, kde uživatel zadá nejvyšší dosažené vzdělání*
- *jedno tlačítko pro reset formulářových dat a jedno pro jejich potvrzení*

Pole pro jméno bude povinné a v případě nevyplnění se zobrazí chybová hláška. Heslo se bude skládat jen z čísel. Chybové zprávy pro jméno a heslo se budou zobrazovat hned vedle pří-

slušných políček červenou barvou. Pod formulářem bude obrázek libovolného loga Javy, který povede na stránky http://java.sun.com.

Po potvrzení formuláře se zobrazí stránka, na které budou vypsány všechny zadané údaje.

Kapitola 5

Konverze a validace

Enterprise aplikace většinou pracují s velkými objemy dat. Je samozřejmě žádoucí, aby tato data byla za všech okolností validní a konzistentní. To je v JSF zajištěno speciálními třídami, kterým se říká **konvertory** a **validátory**. Vždy když uživatel potvrdí formulář, JSF se během svého životního cyklu pokusí zadaná data zkonvertovat na požadovaný datový typ a zvalidovat dle zadaných požadavků (pro více informací o zařazení konverze a validace do životního cyklu JSF aplikace viz 7 – „*Životní cyklus JSF aplikace*“). Pokud se během těchto procesů vyskytnou nějaké chyby, můžeme uživateli poskytnout příslušné informace pomocí tagů `message` či `messages` (viz 4.3.6 – „Zprávy“).

V JSF naleznete sadu standardních konvertorů i validátorů, které využijete v nejběžnějších případech. Všechny tyto třídy musí implementovat příslušné rozhraní - `javax.faces.convert.Converter` v případě konvertorů a `javax.faces.validator.Validator` v případě validátorů. Je také možné, aby si programátor tato rozhraní implementoval sám a vytvořil si tak vlastní třídy pro konverzi respektive validaci. Tvorba vlastních konvertorů a validátorů je však mimo rozsah této práce (pro více informací o tvorbě vlastních konvertorů a validátorů viz např. [3]). Ukážeme si proto pouze práci se základními třídami přítomnými ve standardní JSF knihovně.

5.1. Konvertory

Na data, která jsou v aplikacích zpracovávána, existují dva pohledy. Jedním možným pohledem jsou data uložena v *beanech*, která jsou ukládána do různých datových typů, a druhým možným pohledem jsou data, která vidí uživatel na monitoru, a jsou reprezentována textovými řetězci. Konvertory slouží jako most mezi těmito dvěma pohledy [8]. Např. když v *beanech* pracujeme s proměnnou typu `Date`, potřebujeme, aby se na stránce zobrazila informace o datu, která je srozumitelná pro uživatele, což může být textový řetězec ve tvaru *DD.MM.YYYY* nebo *YYYY/MM/DD* atp. Pokud naopak chceme, aby datum bylo zadáno uživatelem, je třeba zajistit správné uložení textového řetězce do proměnné typu `Date`.

Není vždy nutné přiřazovat konvertor explicitně, neboť u některých kombinací UI komponenty a přiřazeného datového typu je automaticky přiřazen vhodný konvertor. Např. když komponentě `inputText` přiřadíte proměnnou typu `Integer`, je automaticky přiřazen konvertor, který převádí textový řetězec na celé číslo. Z toho důvodu jsme dosud v našich příkladech konvertory nepotřebovali.

5.1.1. Použití konvertorů na stránce

V knihovně JSF naleznete v balíčku `javax.faces.convert` třináct standardních konvertorů, které jsem rozdělil do třech skupin:

- `BooleanConverter`, `ByteConverter`, `CharacterConverter`, `DoubleConverter`, `EnumConverter`, `FloatConverter`, `IntegerConverter`, `LongConverter`, `ShortConverter` - tyto jsou implicitně přiřazovány k příslušným primitivním typům či jejich obalovým třídám.

- `BigDecimalConverter`, `BigIntegerConverter` - tyto jsou implicitně přiřazovány k typům `BigInteger` respektive `BigDecimal`.

- `DateTimeConverter`, `NumberConverter` - toto jsou jediné konvertory, které mají v *core tag library* svůj tag a sice `convertDateTime` a `convertNumber`. Tyto tagy si představíme později v této kapitole.

Page author má čtyři možnosti, jak konvertor ke komponentě připojit [8]:

1. Spárovat komponentu s jedním z primitivních typů, či jeho obalovou třídou. Konvertor tak bude přiřazen automaticky.

Příklad 5.1. Automatické přiřazení konvertoru

```
<h:inputText id="vek" value="#{uzivatel.vek}"/>
```

V tomto příkladě by byl přiřazen konvertor podle typu proměnné `vek` *beanu* `uzivatel`. Pokud by např. tato proměnná byla typu `java.lang.Long`, byl by přiřazen konvertor `LongConverter`.

2. Druhou možností je využít atribut `converter`. Hodnota tohoto atributu musí být tzv. *converter id*, což je konstanta, kterou mají všechny třídy konvertorů. *Id* jednotlivých konvertorů naleznete v [6].

Příklad 5.2. Přiřazení konvertoru pomocí atributu `converter`

```
<h:inputText id="vek" value="#{uzivatel.vek}" ▶
converter="javax.faces.Long" converterMessage="Hodnota věku musí být ▶
číslo."/>
```

3. Třetí možnost je tag `converter` z *core tag library*. Jeho atribut `converterId` nastavíme na hodnotu *id* konvertoru, který chceme použít.

Příklad 5.3. Přiřazení konvertoru pomocí tagu `converter`

```
<h:inputText id="vek" value="#{uzivatel.vek}">
    <f:converter converterId="javax.faces.Byte"/>
</h:inputText>
```

4. Pokud pracujeme s proměnnou číselného typu nebo data a času, můžeme použít již zmíněný tag `convertNumber` respektive `convertDateTime`. Výhodou těchto tagů je, že mají několik atributů, pomocí kterých můžeme chování konvertoru konfigurovat. U třech předchozích možností jsme mohli pouze přiřadit konvertor a spokojit se se standardním chováním.

Příklad 5.4. Přiřazení konvertoru pomocí tagu `convertNumber`

```
<h:inputText id="cena" label="Cena" value="#{objednavka.cena}">
    <f:convertNumber minFractionDigits="0" maxFractionDigits="2"/>
</h:inputText>
<h:message for="cena" style="color:red"/><br/>

<h:outputText value="#{objednavka.cena}">
    <f:convertNumber type="currency" minFractionDigits="2"/>
</h:outputText>
```

První textové pole přijímá číselnou hodnotu s maximálně dvěma desetinnými místy. Druhý výstupní text tuto hodnotu vypisuje se dvěma desetinnými místy ve formátu měny, tzn. např. *452,50 Kč*. V příkladu vidíte také tag `message`, který, jak už jsem zmínil, zpravuje uživatele o chybách, které při konverzi respektive validaci nastanou (viz 4.3.6 – „Zprávy“). Zde bych rád upozornil na atribut `converterMessage`,

pomocí kterého můžete přepsat standardní chybové hlášení svým vlastním. Tuto možnost jsme využili v příkladu 5.2.

Pro více informací o tazích `convertNumber` a `convertDateTime` a jejich atributech viz [10].

5.2. Validátory

Validace je po konverzi další proces, který zajišťuje korektnost dat. Ta, než jsou uložena do modelu, musí být nejen zkonvertována na správný datový typ, ale musí také odpovídat určitým logickým pravidlům, která vyplývají z podstaty modelu. Jako příklad můžeme uvést případ, kdy chceme, aby uživatel zadal svůj věk. Jednalo by se tedy např. o typ *Integer*. Uživateli však nic nebrání zadat zápornou hodnotu, což postrádá smysl. K eliminaci takovýchto případů slouží validace.

Standardní JSF knihovna umožňuje validovat interval číselné hodnoty, délku řetězce a přítomnost hodnoty. Všechny tři možnosti si ukážeme dále. Je třeba upozornit, že veškerá JSF validace probíhá na straně serveru. Pokud byste chtěli provádět validaci na straně klienta, museli byste použít některou z klientských skriptovacích technologií, např. JavaScript.

5.2.1. Použití validátorů na stránce

Přiřazení validátorů ke komponentě se provede pouhým vnořením příslušného tagu validátoru do těla tagu komponenty, kterou chceme validovat. Prostředky pro realizaci třech výše zmíněných variant validace naleznete v *core tag library*.

Interval číselné hodnoty

Zde máte k dispozici dva tagy - `validateLongRange`, který slouží k validaci celočíselných hodnot, a `validateDoubleRange`, který slouží k validaci v oboru všech reálných čísel. Oba disponují atributy `minimum` a `maximum` pro určení rozsahu, v jakém se má zadávaná hodnota pohybovat. Funkcionalitu tagů zajišťují třídy `LongRangeValidator` a `DoubleRangeValidator` z balíčku `javax.faces.validator`.

Příklad 5.5. Validace intervalu číselné hodnoty

```
<h:inputText id="vek" value="#{uzivatel.vek}" validatorMessage="Hodnota ▶
se musí nacházet v intervalu <0;200>!">
    <f:validateLongRange minimum="0" maximum="200"/>
</h:inputText>
```

Délka řetězce

Pro kontrolu délky řetězce slouží tag `validateLength`, jehož funkcionalitu zajišťuje třída `LengthValidator`, která je rovněž z balíčku `javax.faces.validator`. Stejně jako dva předchozí tagy přijímá atributy `minimum` a `maximum`.

Příklad 5.6. Validace intervalu číselné hodnoty

```
<h:inputText id="vek" value="#{uzivatel.rodneCislo}">
    <f:validateLength minimum="10" maximum="10"/>
</h:inputText>
```

Přítomnost hodnoty

Pro kontrolu přítomnosti hodnoty není v JSF žádný tag, nýbrž atribut `required`, o kterém jsme již mluvili v sekci 4.3.3 – „Textové prvky“. Přijímá hodnotu typu `Boolean`. Pokud je nastavena na *true*, uživatel nemůže nechat před potvrzením formuláře pole prázdné.

Příklad 5.7. Validace intervalu číselné hodnoty

```
<h:inputText id="vek" value="#{objednavka.rodneCislo}" required="true" ▶
requiredMessage="Rodné číslo musí být vyplněno."/>
<h:message for="vek" style="color:red"/>
```

Zprávy o chybách se zde obsluhují stejným způsobem, jako u konverze tzn. pomocí tagů `message` a `messages`. I zde bych rád upozornil na atributy, pomocí kterých lze měnit standardní znění chybových zpráv, a sice `requiredMessage` a `validatorMessage`. Jejich použití můžete vidět v příkladech výše.

5.3. Příklad k procvičení



Vytvořte aplikaci, která bude simulovat bankovní transakci. Na úvodní stránce budou následující komponenty:

- *Vstupní textové pole pro zadání částky. Toto pole bude povinné a bude do něj možno zadat kladné reálné číslo s přesností na dvě desetinná místa. Maximální částka transakce bude 500 000.*

- *Vstupní textové pole pro zadání čísla účtu příjemce. Ten bude mít v našem příkladu tvar řetězce o 14 znacích. Toto pole bude rovněž povinné.*

- *Vstupní pole typu textarea pro zadání zprávy pro příjemce. Toto pole nebude povinné a maximální délka zprávy bude 180 znaků.*

- *Pod vstupními políčky bude tlačítko pro potvrzení transakce*

Při potvrzení transakce bude uživatel přesměrován na stránku, kde budou vypsány informace, které o transakci zadal. Částka bude vypsána ve formátu měny na dvě desetinná místa. Pokud uživatel vyplní na první stránce jakékoliv políčko chybně, zobrazí se vedle něj při pokusu o potvrzení chybová zpráva červeným písmem v češtině.

Kapitola 6

Zpracování událostí

Proces zpracování událostí v JSF vychází z *JavaBeans* modelu. Rovněž se zde používají **třídy událostí** a **třídy *listenerů***, které implementují určité rozhraní.

Když některá komponenta vygeneruje událost, je vytvořena instance příslušné třídy události, jež obsahuje informace o svém původci. Obsah těchto informací je odvislý od typu události (pro více informací o třídách událostí viz [6]). Jestliže chceme tuto událost zpracovat, je zapotřebí připojit ke komponentě, která událost generuje, instanci *listeneru*. Ve třídě *listeneru* je implementována metoda, která na vzniklou událost určitým způsobem reaguje.

V JSF existují tři typy událostí:

- ***Action event*** - je generována komponentami, které implementují rozhraní `javax.faces.component.ActionSource`, což je např. komponenta `UICommand`. Událost může tedy nastat třeba při stisku tlačítka či odkazu. Tento typ události představuje třída `javax.faces.event.ActionEvent`.

- ***Value-change event*** - je generována komponentami, které implementují rozhraní `javax.faces.component.EditableValueHolder`, což jsou například `UIInput`, `UISelectOne` a další komponenty, které umožňují uživateli zadat vstupní data. Událost může tedy nastat třeba při zaškrtnutí *checkboxu*, změny výběru v *listboxu* atp. Tento typ události je představován třídou `javax.faces.event.ValueChangeEvent`.

- ***Data-model event*** - může nastat při manipulaci s tabulkami. Jedná se však o složitější událost, jejíž popis je mimo rozsah této práce (pro více informací o *data-model event* viz [5]).

Tyto události lze v JSF zpracovávat dvěma způsoby. Buďto můžete vytvořit výše popsanou třídu *listeneru*, která bude implementovat rozhraní `ActionListener` respektive `ValueChangeListener`, nebo stačí napsat pouze metodu *backing beanu*, jež událost zpracuje.

6.1. Zpracování událostí metodou

Metoda, která zpracovávat události musí být veřejná, musí vracet typ `void` a jako parametr musí přijímat některou z tříd událostí. Může tedy vypadat např. takto:

Příklad 6.1. Metoda *backing beanu*, která zpracovává událost

```
public void zpracujAkci(ActionEvent e) {
    if (vek >= 18) {
        vystup = "plnolety";
    } else {
        vystup = "neplnolety";
    }
}

public String navigace() {
    return vystup;
}
```

V tomto příkladě využíváme akci k ovlivňování navigace mezi stránkami. Proměnná `vystup` bude totiž následně využita jako řetězec pro atribut `action` (viz metoda `navigace()`).[1] Pokud je uživatel plnoletý, bude přesměrován na stránky bez cenzury. Pokud není, obsah bude omezen.

Když chceme ke komponentě přidat metodu, která zpracovává události, využijeme atribut `actionListener` respektive `valueChangeListener`. Přiřazení naší metody k tlačítku by vypadalo následovně:

Příklad 6.2. Přiřazení metody pro zpracování událostí ke komponentě tlačítka

```
<h:commandButton value="Potvrdit" actionListener="#{objednavka.akce}" ▶
action="#{objednavka.navigace}"/>
```

Všimněte si také atributu `action`, který volá metodu `navigace`. Ta vrací proměnnou `vystup`, kterou během zpracovávání akce modifikujeme.

[1] Stejného výsledku bychom samozřejmě docílili, kdybychom if podmínku umístili do metody navigace. Zde však potřebujeme demonstrovat, jak fungují akce.

6.2. Zpracování událostí listenerem

Druhou možností jak zpracovávat události je vytvořit si třídu, která implementuje příslušné rozhraní (`ActionListener` v případě *action event* či `ValueChangeListener` v případě *value-change event*).

Každé z těchto rozhraní má pouze jednu metodu, kterou je třeba implementovat. V případě `ActionListener` je to metoda `processAction(ActionEvent)` a v případě `ValueChangeListener` metoda `processValueChange(ValueChangeEvent)`. V předešlé sekci jsem uvedl příklad zpracování *action event*, implementaci listeneru proto budu demonstrovat na *value-change event*.

Příklad 6.3. Třída implementující rozhraní `ValueChangeListener`

```
package userBeans;

import java.util.Locale;
import javax.faces.context.FacesContext;
import javax.faces.event.ValueChangeEvent;

public class ZmenaStatu implements javax.faces.event.ValueChangeListener {

    public void processValueChange(ValueChangeEvent event) {
        FacesContext context = FacesContext.getCurrentInstance();
        if (((String)event.getNewValue()).equals("us")) {
            context.getViewRoot().setLocale(Locale.US);
        } else {
            context.getViewRoot().setLocale(new Locale("cs", "CZ"));
        }
    }
}
```

Třída `ValueChangeEvent` obsahuje informace o staré a nové hodnotě komponenty, která akci vyvolala. V *if* podmínce zjišťujeme novou hodnotu např. *comboboxu* a na jejím základě nastavujeme hodnotu *locale*[2] pro *view root* (pro více informací o metodách tříd `FacesContext` a `ViewRoot` viz [6]).

[2]Na základě hodnoty *locale* se přizpůsobuje národní prostředí aplikace. Při vypisování peněžních hodnot se například nastaví příslušná měna atp. Pro více informací o třídě Locale viz http://java.sun.com/javase/6/docs/api.

Pro přiřazení tříd *listeneru* ke komponentám se místo atributů používají tagy `actionListener` a `valueChangeListener`. Oba se nacházejí v *core tag library* a v atributu `type` přijímají plný název třídy *listeneru*. Přiřazení výše popsaného *listeneru* ke *comboboxu* by vypadalo následovně:

Příklad 6.4. Přiřazení třídy pro zpracování událostí ke komponentě comboboxu

```
<h:selectOneMenu value="#{objednavka.vybranyStat}">
    <f:valueChangeListener type="userBeans.ZmenaStatu"/>
    <f:selectItems value="#{objednavka.staty}"/>
</h:selectOneMenu>
```

V tomto případě by se *listener* spustil až poté, co byste potvrdili formulář stiskem tlačítka. Pokud byste chtěli zpracovat událost ihned po změně hodnoty *comboboxu*, stačilo by k tagu `selectOneMenu` přiřadit atribut `onchange` s hodnotou `submit()`. Tím se odvoláte na funkci JavaScriptu, která zajistí okamžité potvrzení formuláře.

Příklad 6.5. Přiřazení třídy pro zpracování událostí ke komponentě comboboxu II

```
<h:selectOneMenu value="#{objednavka.vybranyStat}" onchange="submit()">
    <f:valueChangeListener type="userBeans.ZmenaStatu"/>
    <f:selectItems value="#{objednavka.staty}"/>
</h:selectOneMenu>
```

Jak metoda *backing beanu* tak třída *listeneru* mají své výhody i nevýhody. Metoda *backing beanu* je snazší na implementaci. Není třeba tvořit novou třídu a máte také jednodušší přístup k proměnným *backing beanu* pro případ, že je chcete během zpracování události modifikovat. Implementace *listeneru* Vám naproti tomu dává možnost přiřadit komponentě více než jeden *listener*. Tagů totiž můžete do komponenty vnořit libovolné množství. *Listenery* jsou pak volány v pořadí v jakém jsou napsány. Pokud je zároveň přítomen atribut *listeneru*, je volán ještě před *listenery* v tazích.

6.3. Příklad k procvičení



Vytvořte aplikaci, která bude simulovat dobročinnou sbírku. Bude mít pouze jednu stránku, na které bude:

- *Textová komponenta, která bude zobrazovat aktuální vybranou částku ve formátu měny se dvěma desetinnými místy.*
- *Vstupní textové pole, do něhož uživatel zadá částku, kterou chce přihodit. Minimální příhoz činí 0.01 a částku lze zadat s přesností na dvě desetinná místa.*
- *Combobox, který bude obsahovat dvě položky - US a CZ. Jakmile uživatel zvolí jednu z nich, okamžitě se změní měna, ve které je vypsána částka (v případě US na $ a v případě CZ na Kč). Zanedbejme zde kurzové převody.*
- *Potvrzovací tlačítko, po jehož stisknutí se aktualizuje vypsaná částka tak, že se k ní přičte nový příspěvek.*

Samozřejmostí je vypisování chybových zpráv v případě chybného vstupu.

Kapitola 7

Životní cyklus JSF aplikace

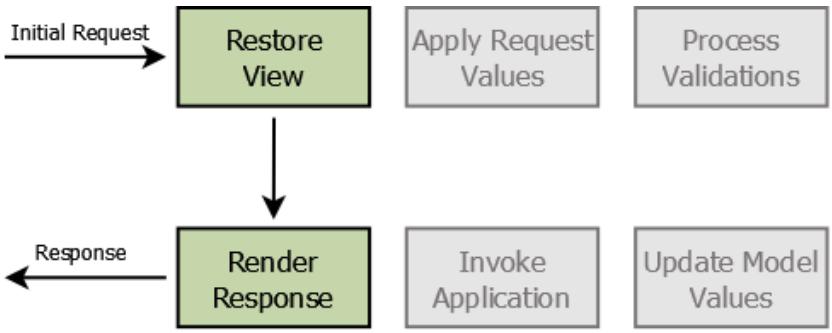
Nyní si pro lepší pochopení všech probraných oblastí popíšeme, co se děje na pozadí JSF frameworku a jak je postupně uživatelský požadavek zpracováván. Vrátíme se také k událostem, které s životním cyklem dosti souvisí.

JSF pracuje stejně jako např. HTML protokol na principu *request-response*. Aby však bylo schopno zajistit veškerou svou pokročilejší funkcionalitu, jako např. konverzi, validaci a zpracování událostí, je celý cyklus obsluhy požadavku rozdělen do **šesti fází**, kterými se mezi vznikem požadavku a zasláním odpovědi prochází. Jejich názvy můžete vidět na obrázku 7.1 a jsou také vysvětleny dále v této kapitole.

Při zpracování cyklu hraje důležitou roli instance třídy `FacesContext`, která je vytvořena na začátku každého životního cyklu voláním metody `getFacesContext()` třídy `FacesContextFactory` (pro více informací o těchto třídách viz [6]). Tato instance udržuje všechny potřebné informace vztahující se k aktuálnímu *requestu*.

V JSF se rozlišují dva druhy požadavků:

Initial Request
Jedná se o prvotní požadavek na stránku, která nebyla předtím načtena. Mohlo na ni být odkázáno z externího zdroje, např. z jiného webu. Při tomto typu požadavku se neprochází všemi šesti fázemi životního cyklu, nýbrž pouze dvěma - z první fáze *Restore View* se přechází přímo k poslední fázi *Render Response*.



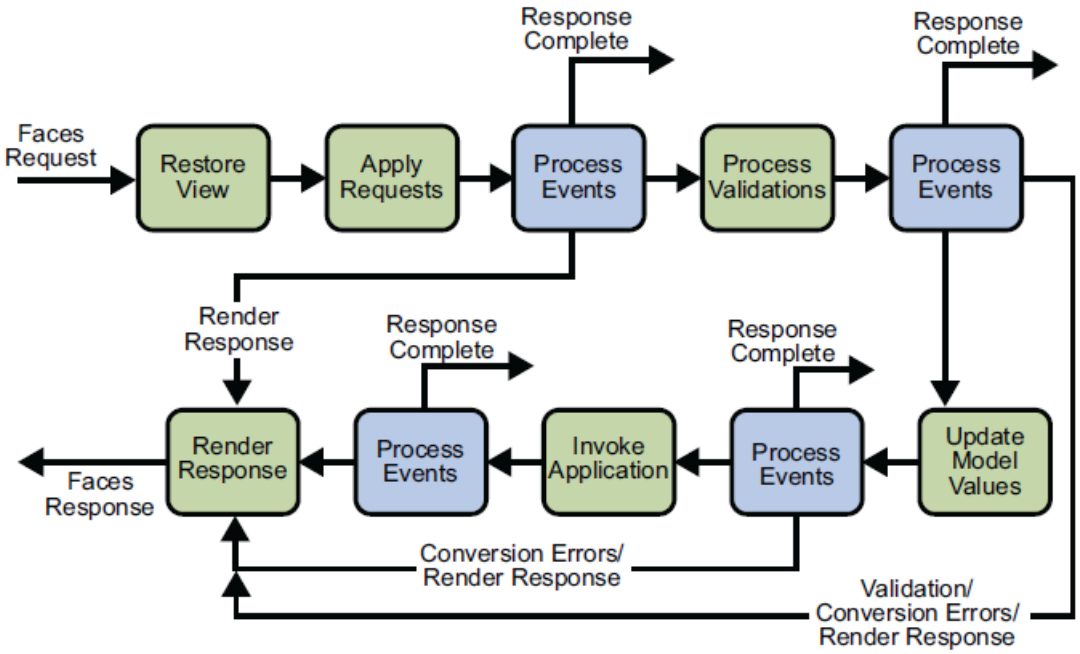
Obrázek 7.1. Životní cyklus JSF aplikace v případě *Initial Requestu* (zdroj: vlastní; inspirován [8])

Při *Initial Requestu* je v *Restore View* fázi vytvořeno nové prázdné *view*. To je následně uloženo v instanci `FacesContext`, jenž zavolá svou metodu `renderResponse()`. Tím se přeskočí všechny ostatní fáze a na základě komponent, které jsou umístěny na stránce, se postupně renderuje odpověď a zároveň s tím se komponentami postupně plní nově vytvořené *view*, aby bylo připraveno pro další *request*.

Postback Request

Na rozdíl od *Initial Requestu* požaduje stránku, která byla již jednou načtena, např. při potvrzení formuláře na stránce.[1] Při tomto typu požadavku se prochází všemi šesti fázemi životního cyklu, samozřejmě za předpokladu, že v žádné fázi nebude cyklus nějakým způsobem přerušen.

[1]Způsob zpracování formuláře jsem nastínil již v sekci 4.3.2 – „Formuláře“, kde jsem zmínil, že veškeré formuláře v JSF směřují vždy na tu samou stránku, ze které byly vyvolány. O tom, která stránka se má načíst jako následující, se rozhoduje až v pozdější fázi cyklu, konkrétně ve fázi Invoke Application.



Obrázek 7.2. Životní cyklus JSF aplikace v případě *Postback Requestu* (zdroj: [8])

Jak vidíte na obrázku 7.2, životní cyklus nemusí vždy proběhnout celý. V průběhu každé fáze, mimo první a poslední, mohou vzniknout události, které jsou po skončení fáze zaslány svým *listenerům* a zpracovány (modré obdélníky na obrázku). Při tomto zpracování mohou být volány některé metody třídy `FacesContext`, které způsobí přerušení, či zkrácení cyklu (šipky s popiskem, které vedou z modrých obdélníků). Konkrétně se jedná o metody `responseComplete()` a `renderResponse()`. Těmito situacemi se budeme zabývat na konci kapitoly v sekci 7.7 – „Události v životním cyklu aplikace“. Nyní si popíšeme průběh všemi fázemi a co se během nich děje. Během procházení následujících šesti sekcí spolupracujte také s obrázkem 7.2.

7.1. Fáze *Restor View*

Toto je první fáze životního cyklu, která nastává ihned po vzniku *requestu*. Během ní se zpracovává *view* požadované stránky (pro připomenutí jak vypadá *view* se můžete podívat na obrázek 3.4). Při *postback requestu* již *view* existuje a informace o něm jsou uloženy buď na serveru nebo na klientovi.[2] Stačí ho tedy pouze načíst. Následně je toto *view* uloženo ve `FacesContextu`.

[2] O tom, zda jsou informace uloženy na serveru nebo na klientovi rozhoduje konfigurace v souboru deployment descriptoru web.xml. Stačí do něj přidat kontextový parametr s názvem javax.faces.STATE_SAVING_METHOD a jako hodnotu zadat client respektive server. Pro více informací o možnostech deployment descriptoru viz [8].

Jak již bylo řečeno, pokud se jedná o *Initial Request*, vytvoří se nové prázdné *view* a hned nato se skočí na fázi *Render Response*, během které se *view* naplní komponentami.

7.2. Fáze *Apply Request Values*

V této fázi se aktualizují hodnoty komponent dle parametrů požadavku. Požadavek přenáší tyto parametry v podobě textových řetězců, musí být tedy zkonvertovány na příslušný typ. V této fázi se však hodnoty zatím neukládají do modelu, nýbrž pouze lokálně do instancí tříd UI komponent. Přesto však již v této fázi mohou být identifikovány některé chyby konverze, jejichž chybové zprávy jsou následně zařazeny do seznamu chybových zpráv ve `FacesContextu`.

V této fázi se také může provádět konverze, validace a zpracování událostí těch komponent, které mají parametr `immediate` nastaven na *true* (viz 7.7.1 – „Ovlivňování životního cyklu pomocí atributu `immediate`“).

Před skončením fáze jsou všechny vyvolané události zaslány svým *listenerům* a následně se přechází k další fázi.

7.3. Fáze *Process Validations*

Jak z názvu vyplývá, v této fázi se provádí validace. Lokálně uložené hodnoty se porovnávají se zadanými požadavky validátorů zaregistrovaných na komponentách.

Pokud při validaci nastanou nějaké chyby, jsou chybové zprávy opět zařazeny do `FacesContextu`. Poté se přeskakuje zbytek cyklu a volá se metoda `renderResponse()`, jenž vykreslí znova stejnou stránku. Ta, za předpokladu, že obsahuje tagy `message` či `messages`, zobrazí chybové zprávy, které se během cyklu ve `FacesContextu` nahromadily, včetně těch, které byly zařazeny v předchozí *Apply Request Values* fázi.

Pokud žádné chyby nenastaly, jsou opět všechny vyvolané události zaslány svým *listenerům* a přechází se k další fázi.

7.4. Fáze *Update Model Values*

V této fázi je již zajištěno, že data odpovídají validačním pravidlům a může se konečně přistoupit k jejich uložení do modelu čili do příslušných *beanů*. I v této fázi však můžou nastat chyby konverze, které nemohly být zachyceny ve fázi *Apply Request Values*, neboť do této chvíle byly hodnoty uloženy pouze lokálně. Jakmile se aplikace pokusí uložit hodnotu do proměnné *beanu*, která je nevyhovujícího typu, nastane chyba a opakuje se opět stejný scénář - do `FacesContextu` se zařadí chybové zprávy, voláním

metody `renderResponse()` se přeskakuje zbytek cyklu a vykresluje se stejná stránka s chybovými zprávami.

Pokud uložení do modelu proběhne v pořádku, jsou opět všechny vyvolané události zaslány svým *listenerům* a přechází se k další fázi.

7.5. Fáze *Invoke Application*

V této fázi se zpracovávají *action events*, které vyvolal uživatel na stránce, např. potvrzení formuláře stiskem tlačítka. V rámci této události se také získá hodnota z atributu `action` komponenty `UICommand`. Ta se porovná s příslušným pravidlem ze souboru **`faces-config.xml`** a tím se rozhodne, jakou stránku zobrazit jako následující.

Nyní již aplikace vykonala všechny potřebné úkony a má veškeré potřebné informace k vykreslení odpovědi.

7.6. Fáze *Render Response*

V této poslední fázi cyklu se vykresluje odpověď v podobě požadované stránky. Pokud však nastaly během cyklu nějaké chyby, vykreslí se stejná stránka s chybovými zprávami na místě tagů `message` a `messages`. Znova důrazně doporučuji tyto tagy používat, aby byl uživatel informován, jakých chyb se dopustil.

Vykreslování má na starosti příslušný *RenderKit*, který postupně prochází tagy na stránce a renderuje je do požadované podoby. V případě, že se jedná o *initial request*, je nyní komponentami také naplňováno *view*, které je následně uloženo, aby bylo k dispozici při příštím *requestu*.

7.7. Události v životním cyklu aplikace

Během zpracování událostí v životním cyklu mohou být volány některé metody třídy `FacesContext`, které přeruší či zkrátí cyklus.

Přerušení může nastat voláním metody `responseComplete()`, která se použije v případě, že požadavek byl přesměrován na jiný zdroj a JSF se již nemusí o odpověď starat. Může se jednat např. o situaci, kdy je uživatel přesměrován na obyčejnou HTML stránku bez JSF elementů, či na soubor, který má být stažen atp. Tato možnost je na obrázku 7.2 znázorněna šipkami s popiskem *"Response Complete"*.

Zkrácení životního cyklu lze provést metodou `renderResponse()`, kterou byla zmíněna již výše v souvislosti s *Initial Requestem*. Metoda zajistí násilné přeskočení zbývajících fází a přikročí rovnou k fázi *Render Response*. Tato možnost je na obrázku 7.2 znázorněna šipkami s popiskem *"Render Response"*. Zkrácení cyklu se používá například

v souvislosti s atributem `immediate`, který jsem zmínil v sekci 4.3.1 – „Základní atributy tagů“.

7.7.1. Ovlivňování životního cyklu pomocí atributu `immediate`

Za normálních okolností jsou akce typu *value-changed events* zpracovávány po fázi *Process Validations* a akce typu *action events* po fázi *Invoke Application*, což je většinou žádoucí chování [3]. Někdy však můžeme potřebovat validaci uspíšit, nebo přeskočit a k tomu slouží právě atribut `immediate`, který dokáže ovlivnit, v jaké fázi budou akce zpracovávány. Můžeme ho použít buď u `UIInput`, nebo u `UICommand` komponent.

7.7.1.1. `UICommand` komponenty s atributem `immediate`

Může nastat situace, kdy potřebujete konverzi a validaci přeskočit. Představte si například formulář, kde máte jedno pole s atributem `required` nastaveným na hodnotu *true* a na konci formuláře je kromě tlačítka *Potvrdit* také tlačítko *Zrušit*, které uživatele vrací třeba na předchozí stránku. Pokud by chtěl uživatel využít tlačítko *Zrušit*, nemá smysl vyžadovat po něm vyplnění povinných polí.

Pro tento případ se nám hodí atribut `immediate`. Nastavení tohoto atributu na hodnotu *true* zajistí, že akce vyvolaná tlačítkem bude zpracována již po fázi *Apply Request Values*. Hned nato je zpracována akce, která řídí navigaci a automaticky volá metodu `renderResponse()`. Zpracování události tak proběhne ještě před procesy konverze a validace a uživatel bude bez problémů přesměrován na požadovanou stránku.

Příklad 7.1. Využití atributu `immediate` u `UICommand` komponenty

```
<h:inputText id="vek" value="#{objednavka.cena}" required="true"/><br/>
<h:commandButton value="Potvrdit" action="success"/>
<h:commandButton value="Zrušit" action="cancel" immediate="true"/>
```

7.7.1.2. `UIInput` komponenty s atributem `immediate`

Nyní si vzpomeňte na příklad 6.5, kde jsme měli *combobox* s atributem `onchange`, jenž volal metodu `submit()`. Tato metoda na straně klienta zajišťuje potvrzení formuláře ihned po změně hodnoty. Představte si, že součástí formuláře je také povinné textové pole, jehož vyplnění však chcete kontrolovat, jen pokud je stisknuto potvrzovací tlačítko, nikoli při změně hodnoty *comboboxu*.

V tom případě nastavíte tagu *comboboxu* atribut `immediate` na *true*, čímž zajistíte, že jeho konverze a validace proběhne již po fázi *Apply Request Values* a hned nato bude vyhozena událost *value-changed event*. Situace je zde však o něco složitější, než u `UICom-`

mand komponenty. Zde totiž není volána metoda `renderResponse()` automaticky a za normálních okolností by následně proběhla konverze a validace všech ostatních polí. Pokud se tomu chcete vyhnout, musíte zavolat metodu `renderResponse()` sami ve vašem *listeneru*.

Příklad 7.2. Využití atributu `immediate` u `UIInput` komponenty

```
<h:selectOneMenu value="#{objednavka.mena}" onchange="submit()" ▶
immediate="true">
    <f:valueChangeListener type="beany.zmenaMeny"/>
    <f:selectItems value="#{objednavka.meny}"/>
</h:selectOneMenu>
```

Do *listeneru* byste pak museli přidat volání metody `renderResponse()` - `FacesContext.getCurrentInstance().renderResponse();` (pro více informací o třídě `FacesContext` a jejich metodách viz [6]).

7.8. Příklad k procvičení



Nakonec si vytvoříme trochu komplexnější příklad, ve kterém budeme vycházet z příkladu o bankovních transakcích z páté kapitoly. Vylepšíme ho pomocí nově nabytých znalostí o atributu immediate.

Na úvodní stránku přidejte textovou informaci, o aktuálním zůstatku na účtu, což bude číslo ve formátu měny. Této proměnné beanu nastavte libovolnou počáteční hodnotu. Pod informaci o zůstatku přidejte combobox, který Vám umožní volit mezi měnami - US (měna $) a CZ (měna Kč). Změna měny se projeví okamžitě po vybrání hodnoty. Nebude se tedy čekat na potvrzení celého formuláře. Zajistěte také, aby v této situaci nebylo upozorňováno na špatně vyplněná pole. Kurzové převody v tomto případě zanedbáme.

Při potvrzení transakce se nyní bude kontrolovat, zda zadaná částka není vyšší než stav účtu. V takovém případě nastane chyba. Kromě tlačítka potvrzení přidejte také tlačítko pro zrušení transakce. To Vás přesměruje na stránku s informačním nápisem "Transakce zrušena". Při zrušení transakce samozřejmě budou opomíjena chybně vyplněná pole.

Pokud bude transakce potvrzena, zobrazí se stránka, která shrnuje informace o transakci. Přidejte však textový odkaz s nápisem "Nová transakce", který povede na úvodní stránku, abyste mohli provést novou transakci. Pokud tento odkaz uživatel využije, na úvodní už uvidí aktualizovanou hodnotu stavu účtu sníženou o částku předchozí transakce.

Kapitola 8

Závěr

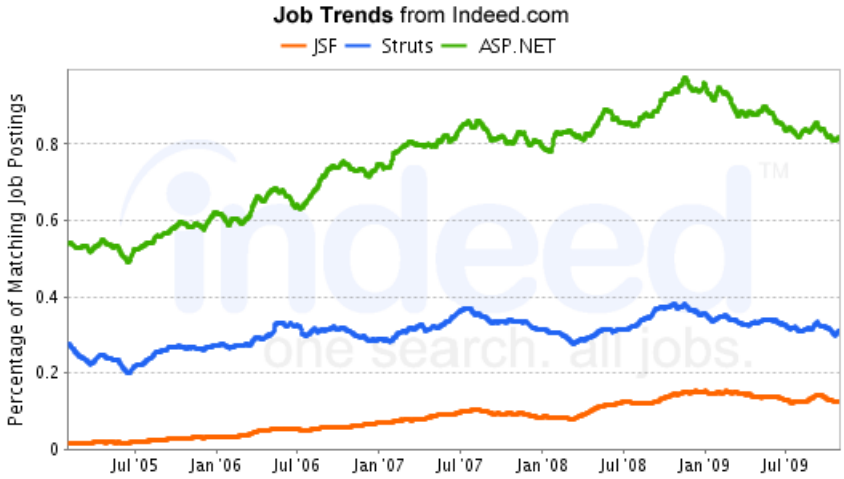
V předchozích kapitolách jsme postupně analyzovali a představili základní možnosti JSF. Na tomto místě bych rád upozornil, že se jedná o značně rozsáhlý framework a rozsah této práce nám neumožňoval zacházet do podrobností. Některé oblasti byly dokonce vynechány zcela. Tuto práci tedy berte jako pouhý úvod do JSF technologie a pro její hlubší poznání využijte také jinou odbornou literaturu.

Po přečtení textu se mnou jistě budete souhlasit v tom, že JSF framework přináší vývojářům v mnoha směrech značné usnadnění práce. Je v něm integrováno poměrně velké množství funkcionality, která by se bez jeho použití programovala manuálně a navíc s sebou přináší všechny výhody, kterými disponuje Java platforma. Na druhou stranu je třeba říci, že JSF není jediným existujícím frameworkem a pokud bychom ho kriticky srovnali s populárními nástroji jako např. Struts, ASP.NET aj., jistě bychom na něm našli také mnoho nedostatků. Obzvláště když si uvědomíme, že má za sebou relativně krátkou historii (první oficiální verze JSF vyšla v roce 2004).

Pro vývojáře, kteří na JSF přechází z jiné platformy než Java, může představovat hlavní problém relativně dlouhá doba potřebná ke zvládnutí celé technologie. Kromě velmi rozsáhlého frameworku je totiž potřeba obeznámit se s technologiemi *Java Servlet* a JSP, na kterých je JSF založeno. Pokud by vývojář přecházel z jiného frameworku pracujícího nad Javou, pravděpodobně by již měl tyto technologie zvládnuté.

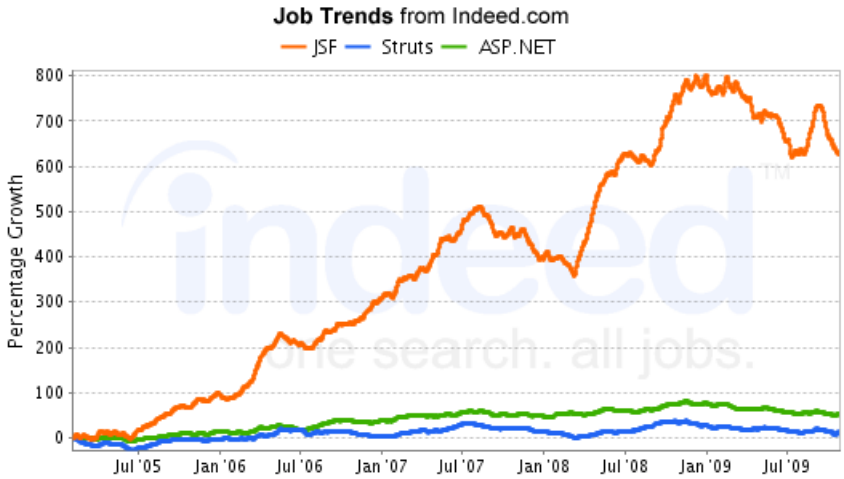
Další relativně značnou nevýhodou JSF je jeho zatím malá rozšířenost mezi vývojáři a firmami. Úroveň dokumentace proto někdy bývá v porovnání s rozšířenějšími frameworky nižší, diskusních fór a dalších zdrojů na Internetu naleznete méně atd. Nutno podotknout, že tento problém nabývá různých rozměrů v závislosti na konkrétní JSF implementaci. Např. referenční implementace je ze strany Sunu samozřejmě podporována více.

Jedním z možných způsobů jak odhadnout rozšířenost frameworku je analyzovat situaci na trhu práce. Na následujících grafech můžete vidět jak si v tomto případě JSF vede ve srovnání s velmi rozšířenými frameworky Struts a ASP.NET.



Obrázek 8.1. Podíly pracovních nabídek na jednotlivé frameworky (zdroj: www.indeed.com)

Graf na obrázku 8.1 znázorňuje podíly pracovních nabídek na jednotlivé frameworky vůči všem pracovním nabídkám na trhu práce. Jak vidíte, v tomto ohledu JSF za svými konkurenty dosti pokulhává. Mezi zaměstnavateli tedy zatím příliš populární není. Na druhou stranu je však třeba vyzdvihnout jeho dynamický vývoj. Když se podíváme na relativní přírůstky pracovních nabídek, je na tom JSF framework oproti svým konkurentům znatelně lépe (viz obrázek 8.2).



Obrázek 8.2. Přírůstky pracovních nabídek na jednotlivé frameworky (zdroj: www.indeed.com)

JSF frameworku jsou vytýkány také některé technické aspekty. Např. nízká transparentnost. V JSF se totiž odehrává mnoho procesů na pozadí a programátor vět-

šinu z nich takřka nemůže ovlivnit. To omezuje možnosti optimalizace aplikací a činí framework pro vývojáře hůře pochopitelným.

Některé výtky občas směřují na rigidní MVC přístup. Na jedné straně se dá označit za výhodu, že JSF implementuje MVC architekturu tak striktně, pokud byste však chtěli použít jiný přístup, máte jen velmi omezené možnosti [4].

Snažit se odhadnout budoucí vývoj JSF by bylo vzhledem k jeho krátké historii dosti obtížné. S ohledem na jeho dosavadní vývoj se však domnívám, že i přes zmíněné nedostatky má potenciál, aby v konkurenci ostatních frameworků nezaniknul a prosadil se. Otázkou zůstává, zda bude s ostatními frameworky na trhu fungovat simultánně nebo je začne postupně vytlačovat.

Bez ohledu na budoucí vývoj JSF věřím, že Vám má práce spolu s vytvořenými doprovodnými příklady bude schopna posloužit jako výukový materiál a poskytne Vám srozumitelný výklad této technologie.

Jak jsem zmínil v úvodu, veškerý obsah bude studentům i dalším zájemcům k dispozici online na stránkách **`http://java.vse.cz`**.

Příloha A

Výpis souborů první JSF aplikace

A.1. `index.jsp`

```
<%--
    Doprovodny priklad k bakalarske praci Vyvoj webovych aplikaci pomoci
    frameworku JavaServer Faces.
    Copyright © 2009 David Hanel.
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<f:view>
    <html>
        <head>
            <meta http-equiv="Content-Type" content="text/html; ▶
charset=UTF-8"/>
          <title>Příklad 3.1 | Vývoj webových aplikací pomocí frameworku ▶
JavaServer Faces</title>
        </head>
        <body>
            <h1><h:outputText value="První JSF aplikace"/></h1>
            <h:form>
                <p>Zadejte vaše jméno: <h:inputText ▶
```

```
value="#{uzivatel.jmeno}"/></p>
                <h:commandButton action="ok" value="Potvrdit"/>
            </h:form>
        </body>
    </html>
</f:view>
```

A.2. vitejte.jsp

```
<%--
    Doprovodny priklad k bakalarske praci Vyvoj webovych aplikaci pomoci
    frameworku JavaServer Faces.
    Copyright © 2009 David Hanel.
--%>

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<%@taglib prefix="f" uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h" uri="http://java.sun.com/jsf/html"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<f:view>
    <html>
        <head>
            <meta http-equiv="Content-Type" content="text/html; ▶
charset=UTF-8"/>
          <title>Příklad 3.1 | Vývoj webových aplikací pomocí frameworku ▶
JavaServer Faces</title>
        </head>
        <body>
            <h1>Vítejte, uživateli <h:outputText ▶
value="#{uzivatel.jmeno}"/>.</h1>
        </body>
    </html>
</f:view>
```

A.3. Uzivatel.java

```
/*
    Doprovodny priklad k bakalarske praci Vyvoj webovych aplikaci pomoci
    frameworku JavaServer Faces.
    Copyright © 2009 David Hanel.
*/

package priklady;

public class Uzivatel {
    private String jmeno;

    public String getJmeno() {
        return jmeno;
    }

    public void setJmeno(String jmeno) {
        this.jmeno = jmeno;
    }
}
```

A.4. faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>

<!--
    Doprovodny priklad k bakalarske praci Vyvoj webovych aplikaci pomoci
    frameworku JavaServer Faces.
    Copyright © 2009 David Hanel.
-->

<faces-config version="1.2"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee ▶
http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd">
```

```
    <navigation-rule>
        <from-view-id>/index.jsp</from-view-id>
        <navigation-case>
            <from-outcome>ok</from-outcome>
            <to-view-id>/vitejte.jsp</to-view-id>
        </navigation-case>
    </navigation-rule>

    <managed-bean>
        <managed-bean-name>uzivatel</managed-bean-name>
        <managed-bean-class>priklady.Uzivatel</managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
</faces-config>
```

A.5. web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" ▶
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ▶
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee ▶
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>faces/index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

Příloha B

Knihovna *core tag library*

Tabulka B.1.

Zdroj: [3]

Tag	Popis
`view`	Vytvoří *view*.
`subview`	Vytvoří *podview* aktuálního *view*.
`attribute`	Deklaruje dvojici klič-hodnota v rámci svého rodiče.
`param`	Deklaruje parametr v rámci svého rodiče.
`facet`	Přiřazuje komponentě *facet*.
`actionListener`	Přiřazuje komponentě *action listener*.
`setPropertyActionListener`	Přiřazuje komponentě *action listener*, který nastavuje hodnotu jedné proměnné.
`valueChangeListener`	Přiřazuje komponentě *value-change listener*.
`phaseListener`	Přiřazuje komponentě *phase listener*.
`converter`	Přiřazuje komponentě konvertor.
`convertDateTime`	Konvertuje datum a čas.
`convertNumber`	Konvertuje číselné hodnoty.
`validator`	Přiřazuje komponentě validátor.
`validatorDoubleRange`	Validuje rozsah reálné celočíselné hodnoty.
`validatorLength`	Validuje délku řetězce.
`validatorLongRange`	Validuje rozsah celočíselné hodnoty.
`loadBundle`	Načítá soubor se zdroji, např. texty.
`selectItem`	Jedna položka selekce.

Tag	Popis
`selectItems`	Skupina položek selekce.
`verbatim`	Ohraničuje čistý text uvnitř komponent (v JSF 1.2 už není třeba používat).

Příloha C

Knihovna *HTML component tag library*

Tabulka C.1.

Zdroj: [3]

Tag	Popis
`form`	HTML formulář.
`inputText`	Textové pole pro zadání jednořádkového textu.
`inputTextarea`	Textové pole pro zadání více řádků textu.
`inputSecret`	Textové pole pro zadání hesla.
`inputHidden`	Skryté formulářové pole.
`outputLabel`	Textový popisek.
`outputLink`	Textový odkaz.
`outputFormat`	Jako `outputText`, může však obsahovat parametry.
`outputText`	Text.
`commandButton`	Tlačítko.
`commandLink`	Odkaz, který funguje jako tlačítko.
`message`	Zobrazuje zprávu pro určitou komponentu.
`messages`	Zobrazuje všechny zprávy.
`graphicImage`	Obrázek.
`selectOneListbox`	*Listbox* s možností výběru jedné hodnoty.
`selectOneMenu`	*Combobox* s možností výběru jedné hodnoty.
`selectOneRadio`	Skupina *radiobuttonů*.
`selectBooleanCheckbox`	Jeden *checkbox*.
`selectManyCheckbox`	Skupina *checkboxů*.

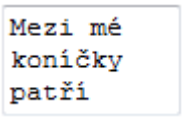
Tag	Popis
`selectManyListbox`	*Listbox* s možností vícenásobného výběru.
`selectManyMenu`	*Combobox* s možností vícenásobného výběru.
`panelGrid`	Tabulka pro účely *layoutu*.
`panelGroup`	Seskupuje několik komponent pomocí `divu`.
`dataTable`	HTML tabulka.
`column`	Sloupec HTML tabulky.

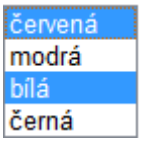
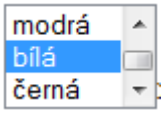
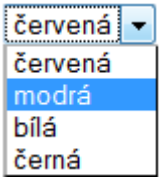
Příloha D

Třídy UI komponent a jejich renderery

Tabulka D.1.

Zdroj: [6]

Komponenta	Renderer	Popis	Obrázek
UICommand	Button	Vytvoří tlačítko pomocí HTML input elementu.	
	Link	Vytvoří HTML a element s doprovodným skriptem.	
UIData	Table	Vytvoří tabulku pomocí HTML table elementu.	-
UIForm	Form	Vytvoří formulář pomocí HTML form elementu.	-
UIGraphic	Image	Vytvoří obrázek pomocí HTML img elementu.	
UIInput	Hidden	Vytvoří skryté pole pomocí HTML input elementu.	-
	Secret	Vytvoří pole pro zadání hesla pomocí HTML input elementu.	
	Text	Vytvoří textové pole pomocí HTML input elementu.	
	Textarea	Vytvoří textové pole pomocí HTML textarea elementu.	

Komponenta	Renderer	Popis	Obrázek
UIMessage	Message	Vytvoří zprávu pro jednu konkrétní komponentu.	Jmeno is required.
UIMessages	Messages	Vytvoří skupinu zpráv pro všechny komponenty.	• Jmeno is required. • vek: 'malo' must be a number
UIOutput	Format	Vytvoří text s parametry.	Parametrický text
	Label	Vytvoří popisek pomocí HTML label elementu.	Popisek
	Link	Vytvoří odkaz pomoci HTML a elementu.	www.google.com
	Text	Vytvoří čistý text. Pokud je přítomen atribut style či styleClass, vytvoří HTML span element.	Prostý text
UIPanel	Grid	Vytvoří layout pomocí HTML table elementu.	-
	Group	Vytvoří seskupení elementu pomocí HTML div nebo span elementu.	-
UISelectBoolean	Checkbox	Vytvoří samostatný *checkbox* pomocí HTML input elementu.	
UISelectMany	Checkbox	Vytvoří skupinu *checkboxů* pomocí HTML input elementů.	
	Listbox	Vytvoří *listbox* pomocí HTML select elementu.	
	Menu	Vytvoří *combobox* pomocí HTML select elementu.	
UISelectOne	Listbox	Vytvoří *listbox* pomocí HTML select elementu.	
	Menu	Vytvoří *combobox* pomocí HTML select elementu.	

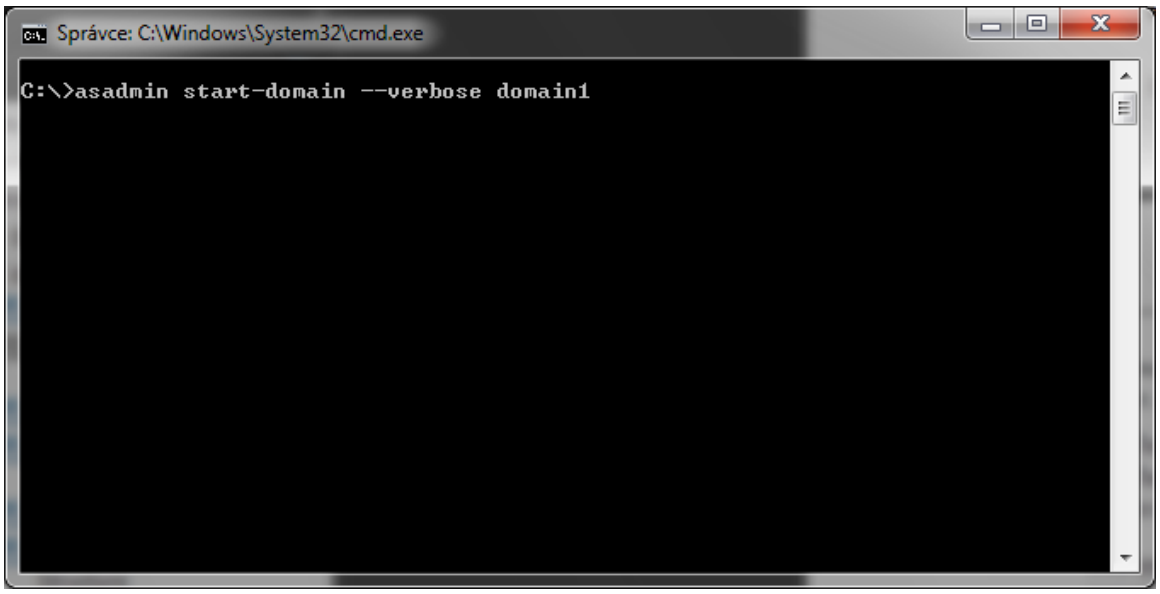
Komponenta	Renderer	Popis	Obrázek
	`Radio`	Vytvoří skupinu *radiobuttonů* pomocí HTML `input` elementů.	

Příloha E

Postup nasazení aplikace na aplikační server GlassFish

E.1. Spuštění aplikačního serveru

Jakmile máte aplikační server nainstalován, spustíte jej z příkazového řádku příkazem `asadmin start-domain --verbose domain1`. Parametr `--verbose` říká, že si přejete vypsat podrobné informace o průběhu spouštění. `domain1` je název defaultní domény. Můžete místo ní zvolit jakoukoli jinou doménu (pro více informací o doménách viz [8]).



Obrázek E.1. Spuštění aplikačního serveru z příkazového řádku

Zastavení aplikačního serveru by se pak provedlo příkazem `asadmin stop-domain domain1`. Jak spuštění tak zastavení lze provést také ze *Start menu* kliknutím na položky *Start Default Server* respektive *Stop Default Server*.

E.2. Přihlášení se k administrátorské konzoli

Když je nyní aplikační server spuštěn, můžete se připojit k administrátorské konzoli. Zadejte do svého prohlížeče URL adresu `http://localhost:4848` a následně vyplňte jméno a heslo a potvrďte. URL adresa, port, jméno i heslo se nastavují při instalaci aplikačního serveru. Toto jsou defaultní hodnoty, takže pokud jste při instalaci nic neměnili, měly by Vám také fungovat.

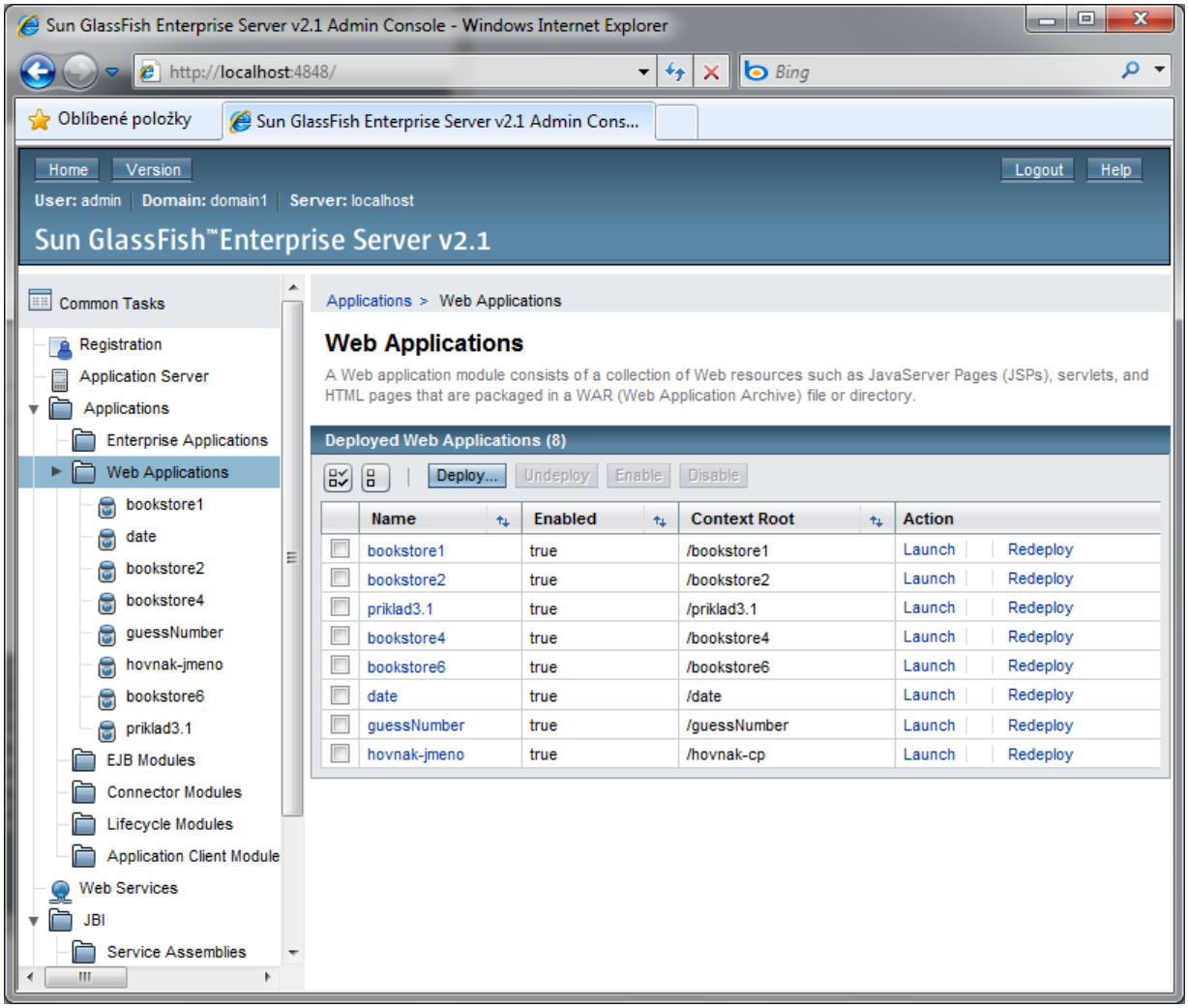


Obrázek E.2. Přihlášení se k administrátorské konzoli

Přihlásit se k administrátorské konzoli můžete opět pomocí položky ve *Start menu* s názvem *Admin Console*.

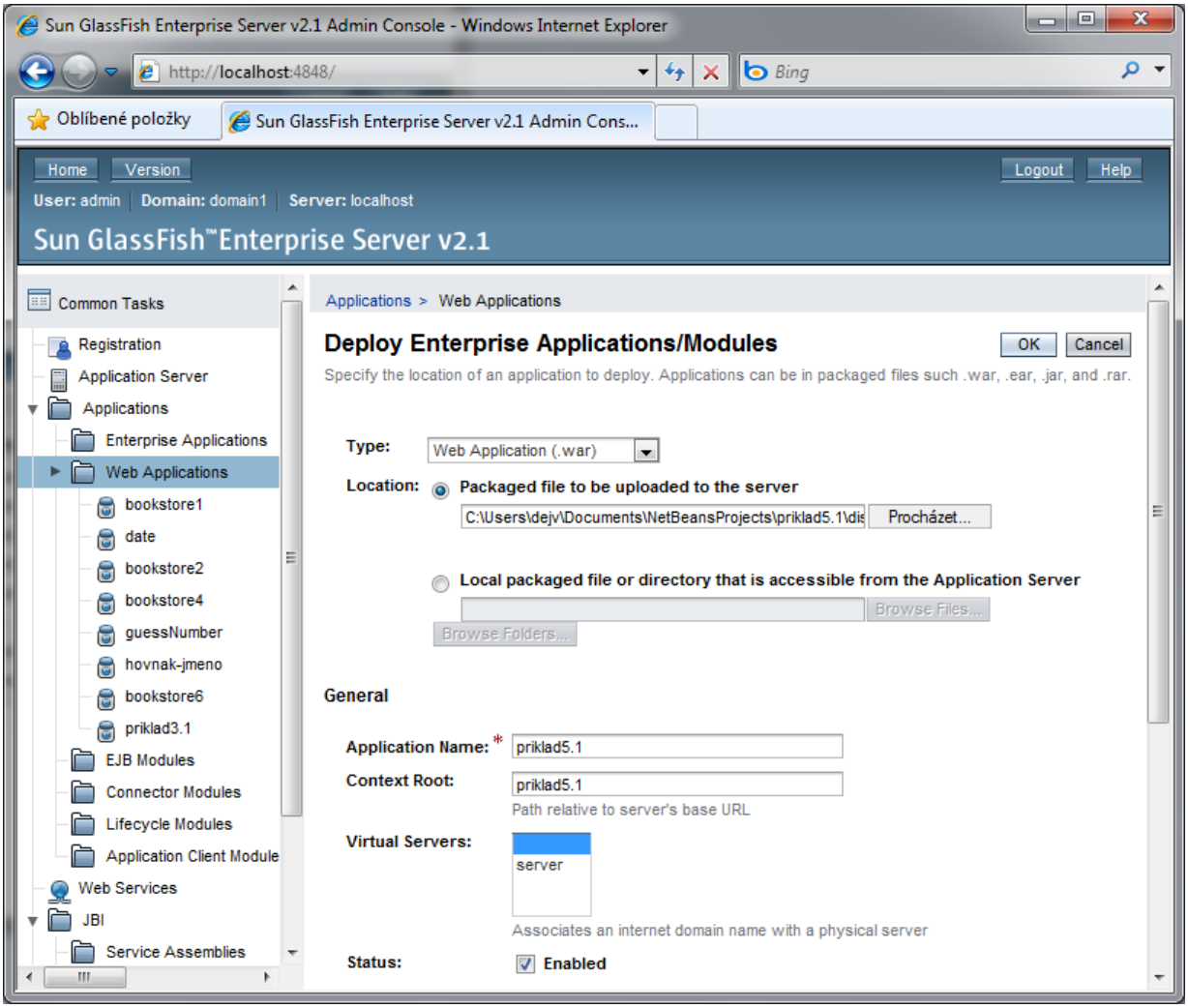
E.3. Nasazení aplikace na server

Nyní je třeba aplikaci nahrát na aplikační server, kde poběží. V levé hierarchické nabídce klikněte na položku *Web Applications*, zobrazí se Vám seznam nasazených aplikací. Nad nimi se nachází tlačítko *Deploy*, na které nyní klikněte.



Obrázek E.3. Nasazení aplikace na server

Na následující stránce kliknete na tlačítko *Procházet* a vyberete war soubor Vaší aplikace nebo její adresář. Dále zde můžete nastavit jméno aplikace, *context root* atp., můžete však všude nechat defaultní hodnoty. Nakonec klikněte na *OK* a nasazení je hotovo.



Obrázek E.4. Nasazení aplikace na server

Aplikaci nyní můžete spustit buď zadáním příslušného URL do prohlížeče, nebo odkazem *Launch* ze seznamu nasazených aplikací v konzoli.

Na závěr upozorním, že některá vývojová prostředí, jako např. NetBeans, výše uvedené kroky automatizují a lze je provést přímo z jejich prostředí. Pro podrobnosti viz nápovědu ke konkrétnímu vývojovému prostředí.

Příloha F

Příklady k procvičení

V této příloze naleznete souhrnně zadání všech příkladů, na která jste mohli v textu narazit.

F.1. Příklad z kapitoly 3 – „*Úvod do JavaServer Faces*“

Vytvořte aplikaci, jejíž úvodní stránka bude obsahovat vstupní textové pole a dvě tlačítka. Jedno tlačítko bude mít popisek "Jsem plnoletý" a druhé "Nejsem plnoletý". Uživatel zadá své jméno a podle svého věku klikne na jedno z tlačítek. Následně bude přesměrován na stránku s uvítací hláškou, která bude obsahovat oslovení a informaci o přístupném obsahu, podle toho, zda je uživatel plnoletý či ne. Např. "Vítejte uživateli Jan. Veškerý obsah našich stránek je Vám k dispozici." nebo "Vítejte uživateli Jan. Litujeme, ale obsah našich stránek je pro Vás cenzurován.". Každé tlačítko bude generovat jiný logický výstup a povede tak na jinou JSP stránku.

F.2. Příklad z kapitoly 4 – „*UI komponenty*“

Vytvořte stránku s formulářem, do kterého bude uživatel zadávat informace o sobě. Formulář bude obsahovat následující prvky:

- *inputText pro zadání uživatelského jména*
- *inputSecret pro zadání hesla*
- *inputTextarea, kde uživatel napíše něco o sobě*
- *selectBooleanCheckbox, kde uživatel zvolí, zda chce dostávat reklamní emaily*
- *selectManyCheckbox, kde uživatel označí své koníčky*
- *selectOneMenu, kde uživatel vybere svou oblíbenou barvu*
- *selectOneRadio, kde uživatel zadá nejvyšší dosažené vzdělání*

- *jedno tlačítko pro reset formulářových dat a jedno pro jejich potvrzení*

Pole pro jméno bude povinné a v případě nevyplnění se zobrazí chybová hláška. Heslo se bude skládat jen z čísel. Chybové zprávy pro jméno a heslo se budou zobrazovat hned vedle příslušných políček červenou barvou. Pod formulářem bude obrázek libovolného loga Javy, který povede na stránky http://java.sun.com.

Po potvrzení formuláře se zobrazí stránka, na které budou vypsány všechny zadané údaje.

F.3. Příklad z kapitoly 5 – „*Konverze a validace*“

Vytvořte aplikaci, která bude simulovat bankovní transakci. Na úvodní stránce budou následující komponenty:

- *Vstupní textové pole pro zadání částky. Toto pole bude povinné a bude do něj možno zadat kladné reálné číslo s přesností na dvě desetinná místa. Maximální částka transakce bude 500 000.*

- *Vstupní textové pole pro zadání čísla účtu příjemce. Ten bude mít v našem příkladě tvar řetězce o 14 znacích. Toto pole bude rovněž povinné.*

- *Vstupní pole typu textarea pro zadání zprávy pro příjemce. Toto pole nebude povinné a maximální délka zprávy bude 180 znaků.*

- *Pod vstupními políčky bude tlačítko pro potvrzení transakce*

Při potvrzení transakce bude uživatel přesměrován na stránku, kde budou vypsány informace, které o transakci zadal. Částka bude vypsána ve formátu měny na dvě desetinná místa. Pokud uživatel vyplní na první stránce jakékoliv políčko chybně, zobrazí se vedle něj při pokusu o potvrzení chybová zpráva červeným písmem v češtině.

F.4. Příklad z kapitoly 6 – „*Zpracování událostí*“

Vytvořte aplikaci, která bude simulovat dobročinnou sbírku. Bude mít pouze jednu stránku, na které bude:

- *Textová komponenta, která bude zobrazovat aktuální vybranou částku ve formátu měny se dvěma desetinnými místy.*

- *Vstupní textové pole, do něhož uživatel zadá částku, kterou chce přihodit. Minimální příhoz činí 0.01 a částku lze zadat s přesností na dvě desetinná místa.*

- *Combobox, který bude obsahovat dvě položky - US a CZ. Jakmile uživatel zvolí jednu z nich, okamžitě se změní měna, ve které je vypsána částka (v případě US na $ a v případě CZ na Kč). Zanedbejme zde kurzové převody.*

- *Potvrzovací tlačítko, po jehož stisknutí se aktualizuje vypsaná částka tak, že se k ní přičte nový příspěvek.*

  Samozřejmostí je vypisování chybových zpráv v případě chybného vstupu.

F.5. Příklad z kapitoly 7 – *„Životní cyklus JSF aplikace“*

Nakonec si vytvoříme trochu komplexnější příklad, ve kterém budeme vycházet z příkladu o bankovních transakcích z páté kapitoly. Vylepšíme ho pomocí nově nabytých znalostí o atributu immediate.

Na úvodní stránku přidejte textovou informaci, o aktuálním zůstatku na účtu, což bude číslo ve formátu měny. Této proměnné beanu nastavte libovolnou počáteční hodnotu. Pod informaci o zůstatku přidejte combobox, který Vám umožní volit mezi měnami - US (měna $) a CZ (měna Kč). Změna měny se projeví okamžitě po vybrání hodnoty. Nebude se tedy čekat na potvrzení celého formuláře. Zajistěte také, aby v této situaci nebylo upozorňováno na špatně vyplněná pole. Kurzové převody v tomto případě zanedbáme.

Při potvrzení transakce se nyní bude kontrolovat, zda zadaná částka není vyšší než stav účtu. V takovém případě nastane chyba. Kromě tlačítka potvrzení přidejte také tlačítko pro zrušení transakce. To Vás přesměruje na stránku s informačním nápisem "Transakce zrušena". Při zrušení transakce samozřejmě budou opomíjena chybně vyplněná pole.

Pokud bude transakce potvrzena, zobrazí se stránka, která shrnuje informace o transakci. Přidejte však textový odkaz s nápisem "Nová transakce", který povede na úvodní stránku, abyste mohli provést novou transakci. Pokud tento odkaz uživatel využije, na úvodní stránce musí být hodnota stavu účtu snížena o částku předchozí transakce.

Terminologický slovník

Aplikační server Serverová aplikace, která obsahuje knihovny implementující požadavky Java EE specifikace. Na aplikačním serveru jsou provozovány *enterprise* aplikace [8].

Framework Sada knihoven, které v obecné podobě implementují časté programátorské konstrukce a usnadňují tak vývojářům rutinní práci. Bývají zaměřeny na určitou vymezenou oblast, např. tvorbu webových aplikací. Často jsou v nich aplikovány různé návrhové vzory a *best practices* z dané oblasti.

Java EE Java platforma určena pro vývoj rozsáhlých *enterprise* aplikací. Základem Java EE je Java SE [8].

JavaServer Faces Serverová Java technologie určená pro tvorbu dynamických webových aplikací. JSF usnadňuje tvorbu uživatelského rozhraní a automatizuje některé úkony, jako např. správa stavů UI komponent, konverzi a validaci dat, zpracování událostí atd. [3].

JavaServer Pages Serverová Java technologie určena pro tvorbu dynamických webových aplikací. V JSP souborech je možné kombinovat statický a dynamicky generovaný obsah. JSP je postaveno nad technologií *Java Servlet* [1].

Konverze Transformace dat mezi modelem a view[1] aplikace. V modelu se nacházejí data různých typů a aby mohla být prezentována uživateli, musí být převedeny do podoby textových řetězců. Naopak aby mohly být textové řetězce zadané uživatelem uloženy v modelu, musí být převedena na vyhovující datové typy [3].

Managed Bean Třída jazyka Java, která vyhovuje určitým formálním a obsahovým požadavkům. Instance těchto tříd jsou spravo-

[1] V tomto případě je pojmem view myšlena část MVC architektury, nikoli strom komponent JSF aplikace.

vány JSF frameworkem a konfigurují se v souboru **`faces-config.xml`** [8].

Renderer — Třída, která zajišťuje prezentaci JSF tagů na konkrétním koncovém zařízení. JSF tagy jsou rendererem převáděny na značkovací jazyk, který je koncové zařízení schopno interpretovat [5].

RenderKit — Sada rendererů, které převádějí JSF tagy na stejný značkovací jazyk. Např. *HTML RenderKit*, jeho všechny renderery převádějí tagy do jazyka HTML [5].

Validace — Kontrolní proces, který zajišťuje, že data zadaná uživatelem vyhovují určitým omezením, která vyplývají z logiky modelu [3].

Životní cyklus — Posloupnost úkonů, které musí být vykonány na pozadí JSF frameworku od obdržení požadavku do vygenerování odpovědi. Mezi úkony, které musí být provedeny, patří např. konverze, validace, zpracování událostí atd. [8]

Literatura

[1] BURD, B.: JavaServer Pages: Podrobný průvodce. Computer Press, 2003. ISBN: 80-7226-804-X.

[2] FOWLER, M.: Patterns of Enterprise Application Architecture. Addison Wesley, 2002. ISBN: 0-321-12742-0.

[3] GEARY, D.; HORSTMANN, C.: Core JavaServer Faces, Second Edition. Prentice Hall, 2007. ISBN: 0-13-173886-0.

[4] HALL, M.: JSF: Introduction, Installation, and Setup. 2009. Dostupný z WWW: http://courses.coreservlets.com/Course-Materials/pdf/jsf/01-JSF-Intro.pdf.

[5] JACOBI, J.; FALLOWS, J. R.: Pro JSF and Ajax: Building Rich Internet Components. Apress, 2006. ISBN: 1-59059-580-7.

[6] Sun Microsystems, Inc.: API Documentation. Dostupný z WWW: http://java.sun.com/javaee/javaserverfaces/1.2/docs/api.

[7] Sun Microsystems, Inc.: Enterprise: Designing Enterprise Applications with the J2EE Platform, Second Edition. Dostupný z WWW: http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e.

[8] Sun Microsystems, Inc.: Java EE 5 Tutorial. Santa Clara, 2008. Part No: 819-3669-11. Dostupný z WWW: http://java.sun.com/javaee/5/docs/tutorial/doc/JavaEETutorial.pdf.

[9] Sun Microsystems, Inc.: Standard HTML RenderKit Documentation. Dostupný z WWW: http://java.sun.com/javaee/javaserverfaces/1.2/docs/renderkitdocs.

[10] Sun Microsystems, Inc.: Tag Library Documentation. Dostupný z WWW: http://java.sun.com/javaee/javaserverfaces/1.2_MR1/docs/tlddocs.

[11] Sun Microsystems, Inc.: Your First Cup: An Introduction to the Java EE Platform. Santa Clara, 2006. Part No: 819–6451. Dostupný z WWW: http://java.sun.com/javaee/5/docs/firstcup/doc/firstcup.pdf.