

Navigace

# Metadata I. A)

- Když chcete načíst data z databáze a zobrazit je na stránce, pak můžete použít `f:metadata`:
- URL: `item-detail.xhtml?id=1`
- `item-detail.xhtml`:

```
<f:metadata>  
    <f:viewAction action="#{itemController.load(param.id)}" />  
</f:metadata>  
<h1>#{itemController.item.name}</h1>
```

Hodnota parametru `id`

Před renderováním této stránky se  
zavolá metoda `itemManager.load()`

Tento tag je až od JSF 2.2 (součást Java EE 7).

Do té doby bylo nutné místo něj použít:


```
<f:event listener="#{itemController.load(param.id)}"  
          type="preRenderView" />
```

# Metadata I. B)


- Nebo můžete použít něco takového:
- `item-detail.xhtml`:

```
<f:metadata>  
  <f:viewParam name="id" value="#{itemController.item.id}" />  
  <f:event listener="#{itemController.load()}" type="preRenderView" />  
</f:metadata>  
<h1>#{itemController.item.name}</h1>
```

Do `#{itemManager.item.id}`  
nastaví hodnotu parametru `id`




Tato metoda by načetla  
záznam s `id` do atributu `item`



# Metadata II.

- Pokud používáte JSF 2.2, pak rozhodně používejte `f:viewAction`. Událost namapovaná na `preRenderView` pomocí `f:event` se totiž zavolá při každém requestu (tzn. i při každém AJAX dotazu)!
- Pokud nemůžete provést upgrade na JSF 2.2, tak abyste na předcházejícím snímku v metodě `load()` zabránili SELECTu do databáze, tak musí vypadat následovně:

```
public void load() {  
    if (!FacesContext.getCurrentInstance().isPostback()) {  
        // zavolani servisni vrstvy a nacteni zaznamu z databaze  
    }  
}
```



Knihovna OmniFaces má tuto metodu se stejnou funkcionalitou: `Faces.isPostback()`

# PostConstruct

- Pokud nezískáváte data z databáze na základě nějaké vstupní hodnoty od klienta (na předcházejících snímcích to byl atribut `id`), poté je možné na managed bean použít metodu s anotací `@PostConstruct`. Taková metoda se zavolá při konstrukci objektu:

```
@ViewScoped @Named

public class ItemListController implements Serializable {

    @Inject private ItemService itemService;

    private List<Item> itemList;

    @PostConstruct public void loadItemList() {
        itemList = itemService.getItems();
    }
}
```

Pozor na scope objektu, abyste tuto metodu nevolali mockrát!!!  
To je typická začátečnická chyba ... proto je obvyklejší tento přístup vůbec nepoužívat a používat metadata.


# Odkazy I.

- Pomocí `<h:outputLink>` je možné vytvořit HTML odkaz na jakoukoli URI:

```
<h:outputLink value="http://www.google.com">  
    page hello  
</h:outputLink>
```

- Pomocí `<h:link>` je možné vytvořit HTML odkaz na XHTML stránku uvnitř web. aplikace:

```
<h:link outcome="hello" value="page hello" />
```

- Pokud taková xhtml stránka neexistuje, pak se odkaz nevygeneruje!
- Od JSF 2.  Název XHTML stránky bez `.xhtml`
- Pokud `outcome` začíná s lomítkem, pak je adresa absolutní, pokud ne, pak je relativní.

# Odkazy II.

- Můžete také posílat parametry:

```
<h:outputLink value="http://www.google.com">
```

```
    <f:param name="q" value="java" />
```

```
    google Java
```

```
</h:outputLink>
```

```
<h:link outcome="item-detail" value="#{item.name}">
```

```
    <f:param name="id" value="#{item.id}" />
```

```
</h:link>
```

- Také můžete použít:

```
<a href="hello.xhtml">page hello</a>
```

... ale lepší je použít `outputLink` nebo `link`

# Odkazy III.

- Poslední způsob vytvoření odkazu je `h:commandLink`, který je možné používat pouze uvnitř formuláře (`h:form`) a generuje `<a>` element s `onclick` skriptem, který odešle POST formulář.
  - Ideálně nepoužívat.

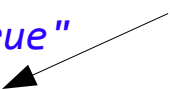


# Tlačítka I.

- Tag `h:commandButton` vygeneruje tlačítko. Tento tag je možné používat pouze uvnitř formuláře (`h:form`) a vygeneruje `<input type="submit" />`
- Vyvolá HTTP POST na formuláři.
- Nejjednodušší použití je:

```
<h:commandButton  
    immediate="true"  
    action="items?faces-redirect=true"  
    value="Cancel" />
```

Přesune řízení na `items.xhtml`



- Typické použití:

```
<h:commandButton action="#{itemManager.save()}" value="Save item" />
```

# Tlačítka II.

- Také existuje tag `h:button`, který vygeneruje `<input type="button" />` s atributem `onclick`. Výsledkem je tedy GET požadavek na příslušné URI.
  - Ideálně nepoužívat.

Formulář Mojarra (bez AJAXu)

# Formulář I.

- Typický formulář s tlačítky OK a Cancel:

```
<h:form>
    name:<br />
    <h:inputText value="#{itemManager.item.name}" title="name:" id="name" /> <br />
    description:<br />
    <h:inputText value="#{itemManager.item.description}" title="description:" /> <br />
    price: <br />
    <h:inputText value="#{itemManager.item.price}" title="price:" id="price" /> <br />
    <br />
    <h:commandButton action="#{itemManager.save()}" value="Save item" />
    <h:commandButton immediate="true" ←———— Neprovede validaci
        action="items?faces-redirect=true&includeViewParams=true"
        value="Cancel" />
</h:form>
```

Provede redirect na items.xhtml


# Formulář II.

```
@ViewScoped @ManagedBean public class ItemManager {
```

```
    private Item item;
```

```
    @EJB private ItemService itemService;
```

Objekt item je nutné při konstrukci ItemManager vytvořit



```
    @PostConstruct public void init() { item = new Item(); }
```

```
    public String save() {
```

```
        itemService.save(item);
```

Název XHTML stránky bez .xhtml



```
        return "item-edit?faces-redirect=true";
```

```
    }
```

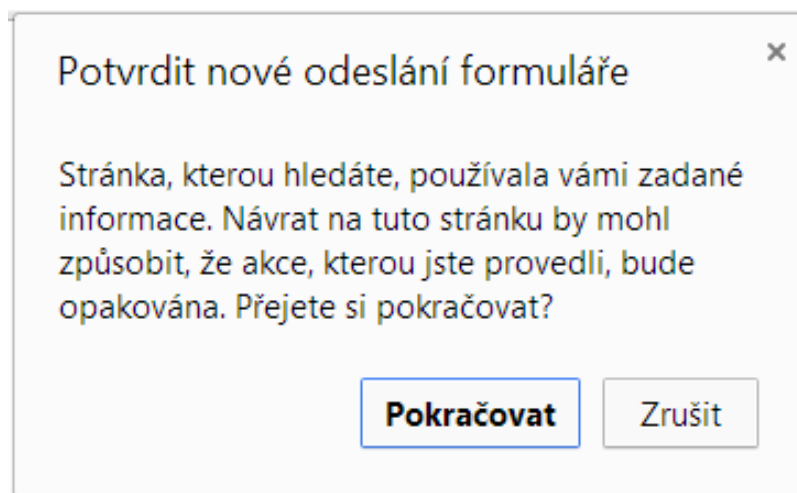
```
    public Item getItem() { return item; }
```

```
    public void setItem(Item item) { this.item = item; }
```

```
}
```

# POST-REDIRECT-GET I.

- Je best practice, abyste po vyvolání HTTP POST operace provedli redirect pomocí HTTP GET operace. Díky tomu když uživatel vyvolá HTTP POST operaci a poté zmáčkne F5 (Refresh), pak se mu nezobrazí:



- [https://blogs.oracle.com/enterprisetechtips/entry/post\\_redirect\\_get\\_and\\_jsf](https://blogs.oracle.com/enterprisetechtips/entry/post_redirect_get_and_jsf)

# POST-REDIRECT-GET II.

item-edit.xhtml:

```
<h:commandButton action="#{itemManager.save()}" value="Save item" />
```

ItemManager:

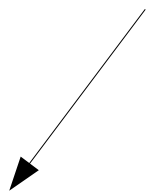
**POST:**

```
public void save() {  
    itemService.save(item);  
}
```

**PRG:**

```
public String save() {  
    itemService.save(item);  
    return "item-edit?faces-redirect=true";  
}
```

**Poznámka:** Můžete také přesunout řízení na jinou JSF stránku, ale best practice je přesunout řízení na stránku, ze které vzešel požadavek a zobrazit na ní informaci o výsledku. Blbá věc je, že se vrácením textu z této metody ukončí View Scope :-)



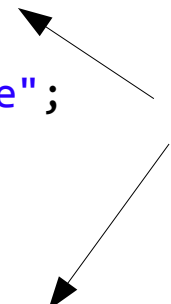
# POST-REDIRECT-GET III.

- Jak zobrazit informaci o výsledku:

ItemManager:

```
public String save() {  
    itemService.save(item);  
    FacesContext.getCurrentInstance().getExternalContext()  
        .getFlash().put("success", true);  
    return "item-edit?faces-redirect=true";  
}
```

Typické použití flash scope.  
Atribut ve flash scope přežije  
redirect a až poté je zrušen.



item-edit.xhtml:

```
<h:panelGroup rendered="#{flash['success'] eq true}"  
    style="color: green" layout="block">  
    item saved  
</h:panelGroup>
```

**Poznámka:** Na starších Mojarra implementacích to nefunguje kvůli chybě:  
<http://stackoverflow.com/questions/11194112/understand-flash-scope-in-jsf2>



# POST-REDIRECT-GET IV.

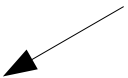
- Můžete také chtít do stránky, na kterou provádíte redirect, poslat parametry:

ItemManager:

```
public String save() {  
    itemService.save(item);  
    return "item-detail?faces-redirect=true&includeViewParams=true";  
}
```

item-detail.xhtml:      Jaké parametry se budou posílat do  
item-edit.xhtml záleží na tom,  
jaké jsou v ní definované!

```
<f:metadata>  
    <f:viewParam name="id" value="#{itemManager.item.id}" />  
    <f:event listener="#{itemManager.load()}" type="preRenderView" />  
</f:metadata>
```



# Formulář PrimeFaces

# Odlišnosti oproti Mojarra

- Pro vytvoření formuláře se používají tagy:
  - `p:panelGrid`, `p:row`, `p:column` – pro vygenerování tabulky
  - `p:inputText` atd. – pro tvorbu komponent
  - `p:commandButton` – tlačítko
  - `p:messages`, `p:message` – validační hlášky
- PrimeFaces odesílají formulář přes AJAX POST, tudíž standardně nemusíme řešit POST-REDIRECT-GET (PRG). Nicméně pokud bychom u `p:commandButton` nastavili atribut `ajax="false"`, pak bychom PRG řešit museli.

# PrimeFaces formulář I.

- Formulář s autoUpdate:

```
<h:form>
```

```
<p:panelGrid>
```

```
<p:row>
```

```
<p:column>
```

```
<p:messages showDetail="true" autoUpdate="true" />
```

```
quantity: <p:inputText value="#{itemController.quantity}" />
```

```
<p:commandButton value="Add to basket"
```

```
action="#{itemController.addItemToBasket()}" />
```

```
</p:column>
```

```
</p:row>
```

```
</p:panelGrid>
```

```
</h:form>
```

Zobrazí detail zprávy  
(viz. Manager dále)

Automaticky provede  
update hodnot tohoto  
tagu pomocí AJAXu.

# PrimeFaces formulář II.

- Formulář bez autoUpdate:

```
<h:form>
  <p:panelGrid id="grid">
    <p:row>
      <p:column>
        <p:messages showDetail="true" />
        quantity: <p:inputText value="#{itemController.quantity}" />
        <p:commandButton value="Add to basket"
          Vyvolá update
          messages
          přes AJAX → action="#{itemController.addItemToBasket()}"
                       update="grid" />
      </p:column>
    </p:row>
  </p:panelGrid>
</h:form>
```

↑  
Pokud by bylo id="grid" mimo tento strom,  
pak by se také dalo použít :grid  
(pak se začíná od rootu),  
Také zde může být výčet idček: grid :text :dalsi, ...

# Modální dialog

Tlačítko pro zobrazení modálního dialogu

```
<h:form>
```

```
  <p:commandButton value="add" oncomplete="PF('itemDialog').show()" resetValues="true"
  update=":itemForm"
```

```
    action="#{itemController.clear()}" />
```

```
</h:form>
```

Dialog

```
<p:dialog widgetVar="itemDialog" closeOnEscape="true" modal="true">
```

```
  <h:form id="itemForm">
```

```
    <p:commandButton value="save" action="#{itemController.save()}" update=":itemsTable, itemForm"
    oncomplete="handleDialogSubmit('itemDialog', xhr, status, args)" />
```

```
  </h:form>
```

```
</p:dialog>
```

```
<script type="text/javascript">
```

```
  function handleDialogSubmit(dialogName, xhr, status, args) {
```

```
    if (!args.validationFailed) {
```

```
      PF(dialogName).hide();
```

```
    }
```

```
  }
```

```
</script>
```

Pokud ve formuláři v dialogu nejsou chyby,  
pak se zavře.

# update

- Pokud provádíte update komponenty v jiném než aktuálním formuláři, pak musíte nejprve zadat ID příslušného formuláře!
  - Čili: `update=":form2:grid"`
- Uvnitř atributu update můžete použít:
  - Název\_ID
  - :Název\_ID (když je mimo aktuální komponent. strom)
  - @(jquery\_selector)
    - Můžete použít JQuery!
      - @(selector) ~ \$("selector")
    - <http://blog.primefaces.org/?p=1867>

# update

- Optimalizační tip: Provádějte update pouze těch komponent, které se změnily. Když budete updatovat vše, pak budete mít aplikaci pomalou.
- Je možné volat update pouze na JSF komponenty, nemůžete zavolat update na HTML tag ... ale můžete takový tag zapouzdřit do JSF komponenty a na tu poté zavolat update).



# Manager

- Manager:

```
public void addItemToBasket() {  
    basket.add(item, quantity);  
    FacesContext.getCurrentInstance().addMessage(null,  
        new FacesMessage(FacesMessage.SEVERITY_INFO, "Result:", "Item added to basket"));  
}
```

summary

Typ zprávy:  
INFO, WARN,  
ERROR, FATAL

detail

Přidá zprávu, která se zobrazí klientovi

Poznámka: Uvnitř jedné metody je vhodné volat  
`FacesContext.getCurrentInstance()` max. jednou.

- Poznámka: OmniFaces obsahuje užitečnou metodu, která dělá to samé: `Messages.addInfo()`;

# Best Practices

# Mazání záznamu I.

```
<p:dataTable value="#{itemController.items}" var="item" id="dataTable">
  <p:column>
    <h:form>
      <p:commandButton action="#{itemController.delete(item)}"
        value="delete" update=":dataTable" />
    </h:form>
  </p:column>
  <p:column>
    <a href="item.xhtml?name=#{item.shortName}"> #{item.title} </a>
  </p:column>
</p:dataTable>
```

Od JSF 2 je možné  
sem vložit objekt

# Mazání záznamu II.

- Nebo:

```
<p:dataTable value="#{itemController.items}" var="item" id="dataTable">
  <p:column>
    <h:form>
      <p:commandButton action="#{itemController.delete()}"
        value="delete" update=":dataTable">
        <f:setPropertyActionListener value="#{item}" target="#{itemController.item}" />
      </p:commandButton>
    </h:form>
  </p:column>
  <p:column>
    <a href="item.xhtml?name=#{item.shortName}"> #{item.title} </a>
  </p:column>
</p:dataTable>
```

Smaže záznam  
itemController.item

Před zavoláním akce na commandButton  
nastaví hodnotu itemController.item

# Jednoduchý formulář

- Pokud máte jednoduchý formulář, ve kterém vyplňujete například pouze jeden input, pak nemusíte vytvářet JSF managed bean s jedním atributem, ale můžete použít tento přístup:

```
<h:form>
```

Quick load video URL:

```
<p:inputText binding="#{youtubeVideo}" style="margin-right:10px;width:560px" />
```

```
<p:commandButton value="Load"
```

```
    action="#{groupDetailController.loadYoutubeVideo(youtubeVideo.value)}"
```

```
    update=":formGrid" />
```

```
</h:form>
```

- Pozor! Tento přístup nezneužívejte! Na něco takového je super, ale jeho použití ve větším rozsahu by se jednalo o anti-pattern.