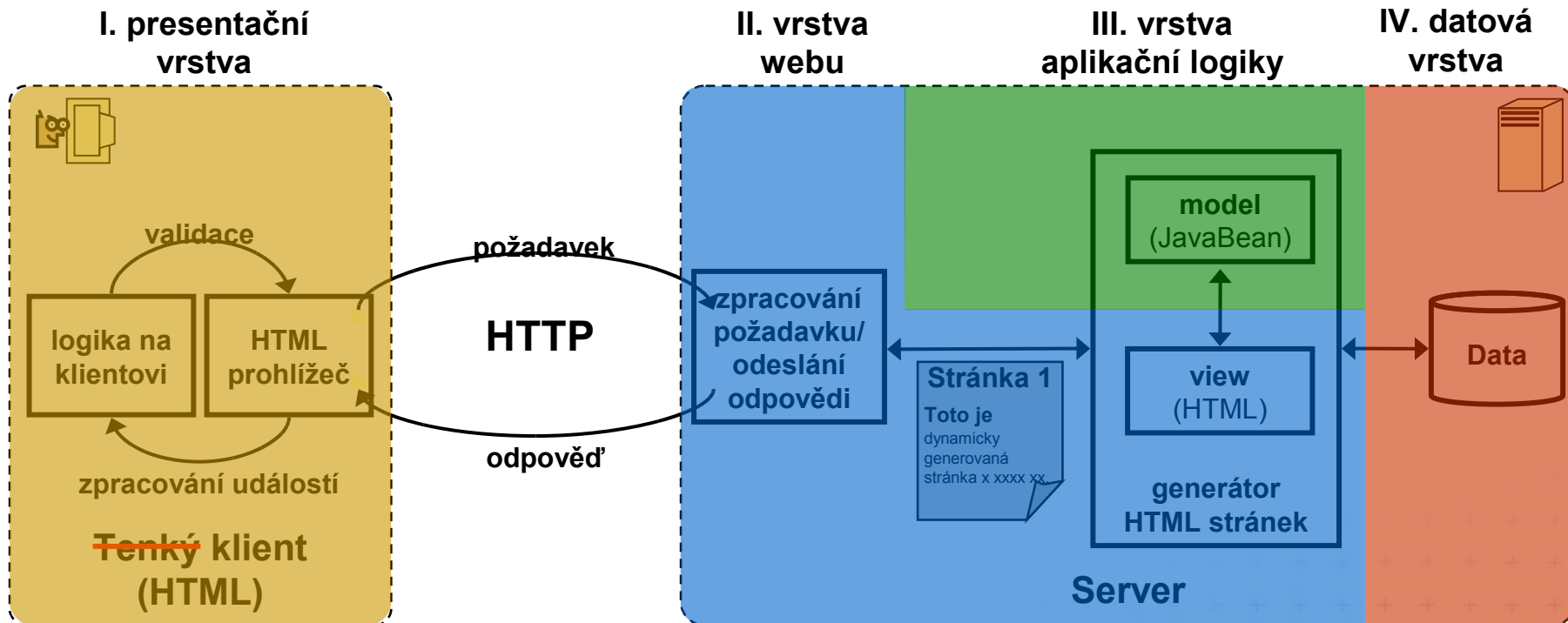

Oddělení aplikační a prezentační logiky Smarty

Martin Klíma



Architektura web aplikace: dynamický web



Architektura MVC

M = Model

V = View

C = Controller

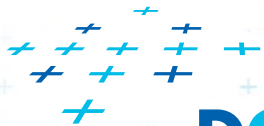
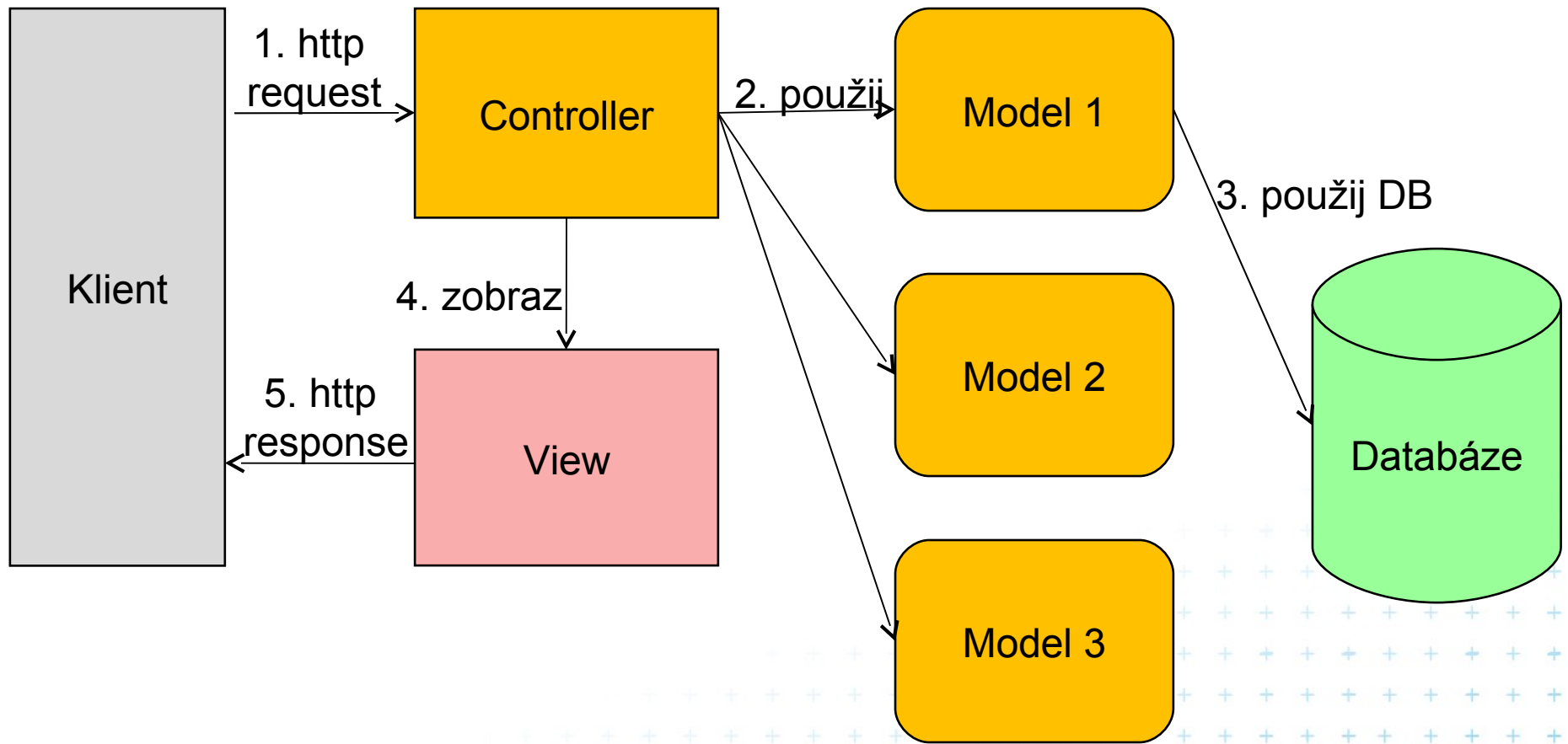
Model reprezentuje aplikační logiku

View reprezentuje prezentační logiku

Controller reprezentuje logiku, která to vše řídí



MVC



Výhody MVC

- Dobré oddělení aplikační (Model) a prezentační (View) logiky
- Přehlednost
- Objektovost
- Možnost dělby práce
 - view má na starosti grafik
 - model má na starosti programátor
- Lepší testování
- Co se v praxi nejvíc mění (u webové aplikace)

– view



Nevýhody MVC

- Složitější projekt
- Více různých souborů
- Pro malé věci možná zbytečné
 - to je možná moc silné tvrzení 😊



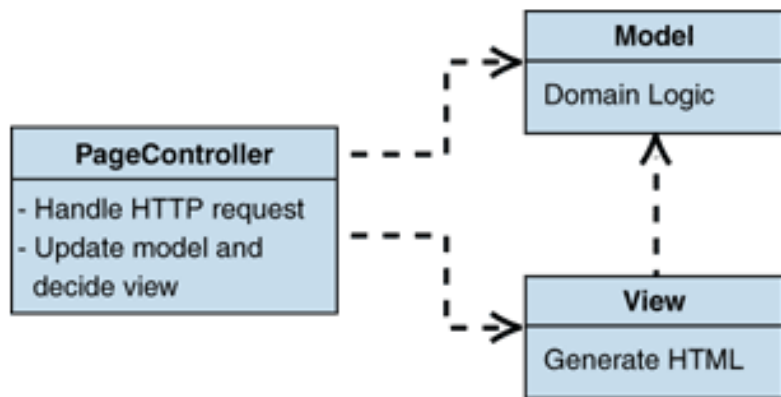
Varianty MVC

- Existuje řada variant
 - Page Controller
 - Front Controller
 - Composite View

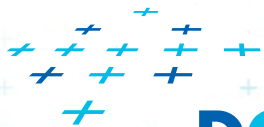


Varianty MVC – Page Controller

Každá adresa (v PHP skript) má svůj vlastní controller



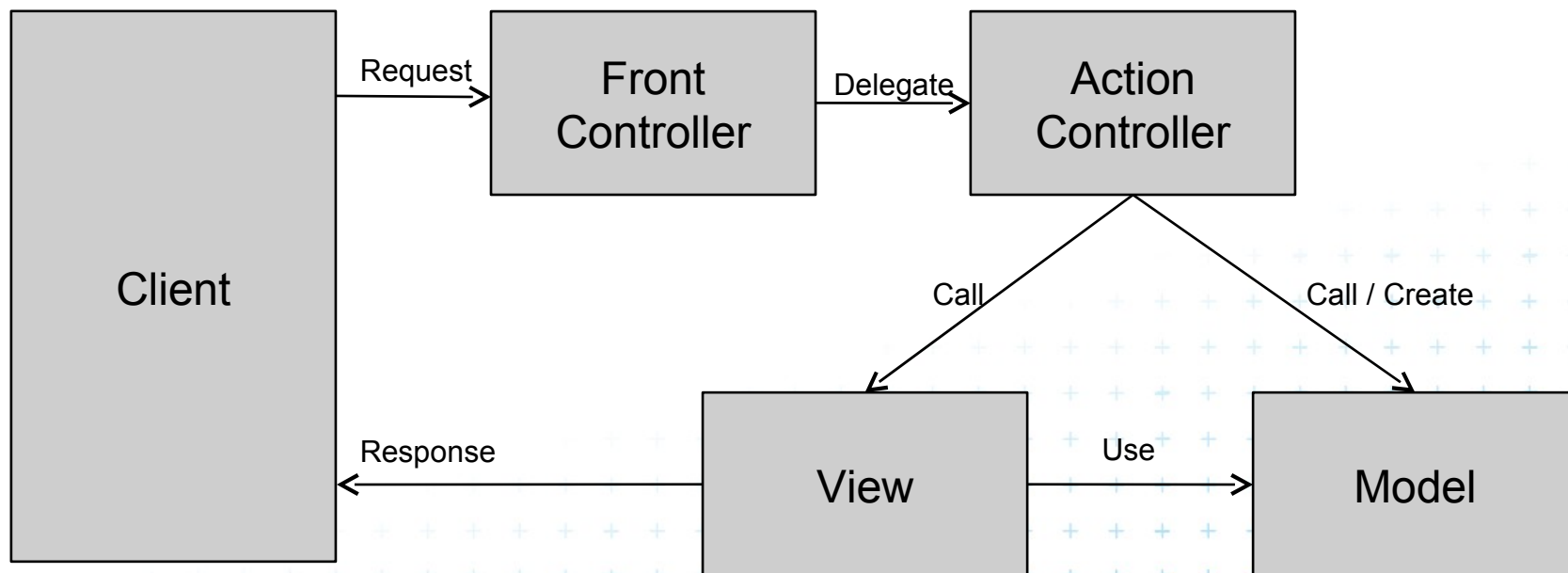
Zdroj MSDN <http://msdn.microsoft.com/en-us/library/ms978764.aspx>



Varianty MVC – Front Controller

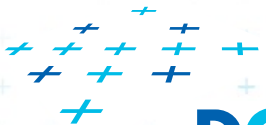
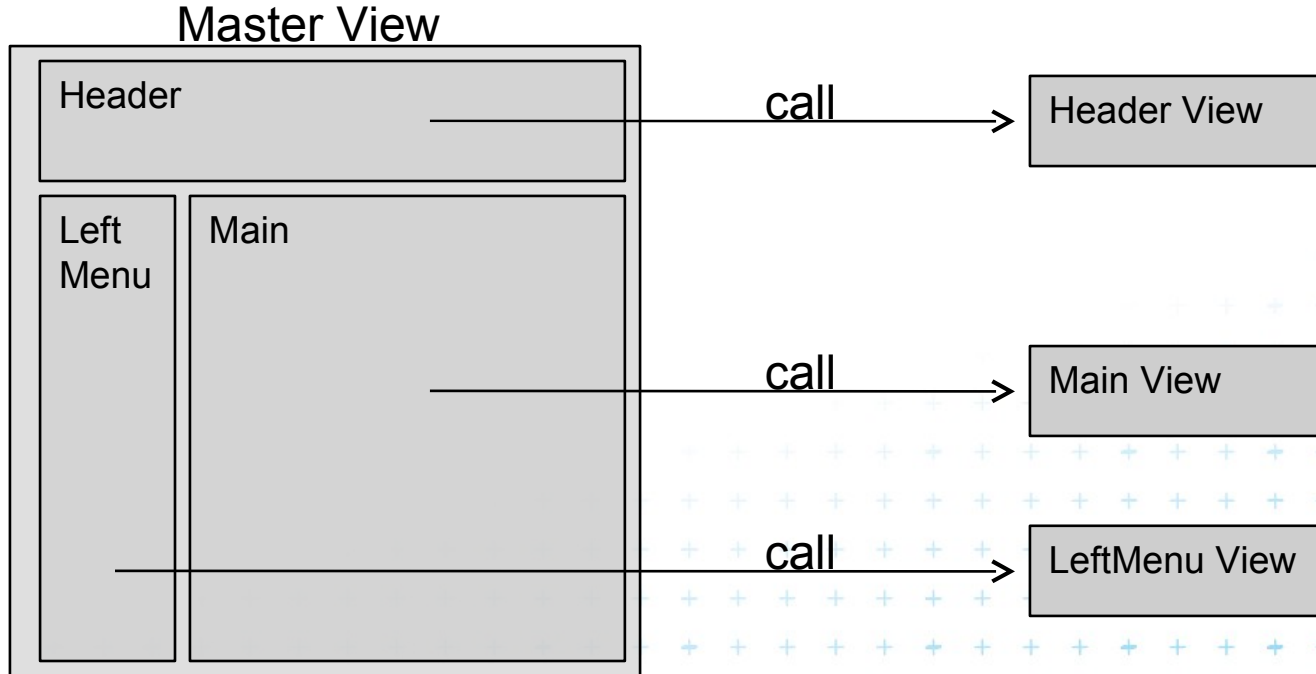
Jeden Controller pro všechny stránky. V PHP se dá zajistit pomocí mod-rewrite v pache.

Front Controller udělá globální práci nutnou pro všechny dotazy a deleguje dotaz na konkrétní implementaci



Composite View

- View se skládá z globální předlohy, která rozvrhuje menší části definované jinými View
- Skinovatelné aplikace, abstrakce



PHP a MVC

- PHP nemá nativní podporu pro MVC
- Nicméně je to možné "naroubovat"
- Frameworky, které to podporují
 - Prado
 - Zend Framework
 - FastTemplate
 - PHPLib
 - Smarty smarty.php.net



Jak na to v PHP - Smarty

Použití šablon (template)

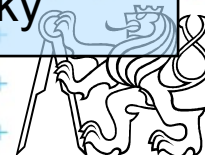
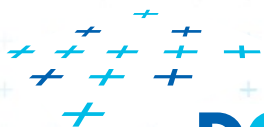
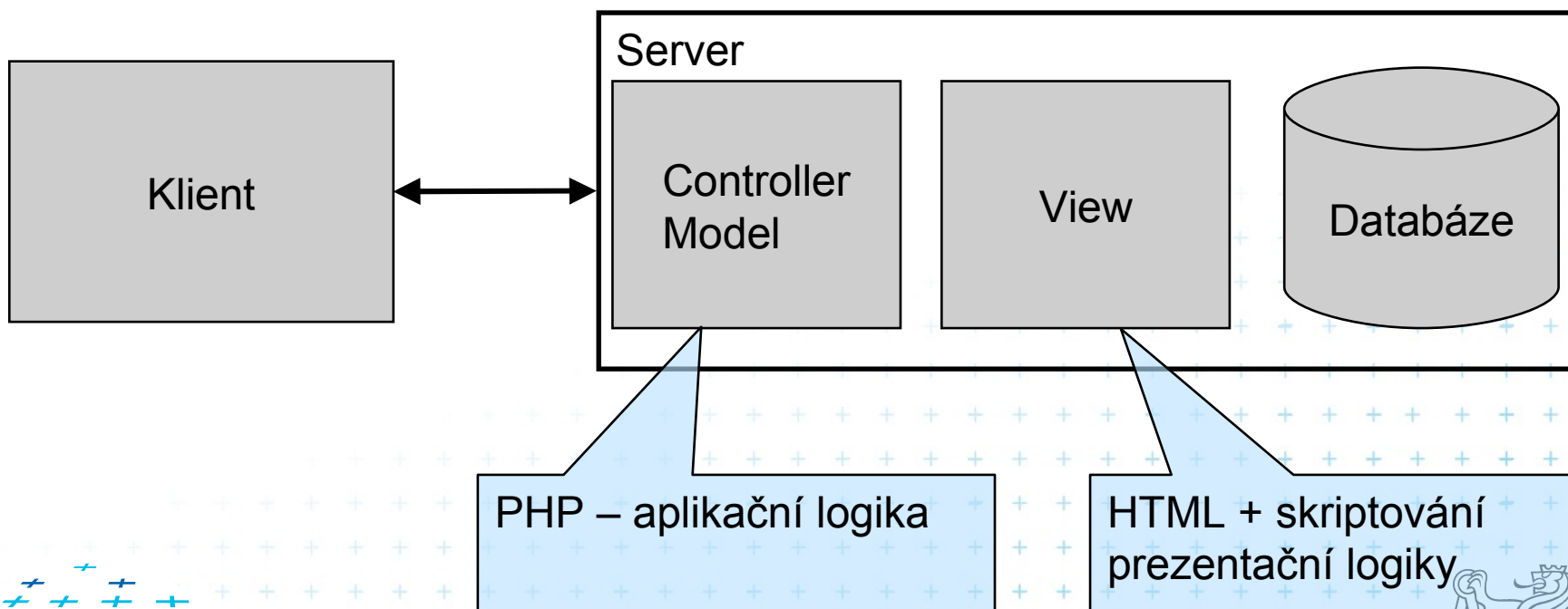


Co šablony dělají

Oddělují aplikační kód a prezentační kód

Je to podpora pro tvorbu View

Controller a Model tvoříme sami



Výhody a nevýhody

■ Výhody

- Jednodušší vývoj
- Možnost dělby práce
- Průhledný a dobře spravovatelný kód
- Může být značně rychlé (cache)

■ Nevýhody

- Složitý projekt, nutno integrovat více souborů
- Aditivní výpočetní výkon
- Nutnost parsovat šablony
- Nutnost naučit se další skriptovací jazyk



Existující enginy

- × Smarty
- × FastTemplate
- × PHPLib
- × ...
- × Můj vlastní?

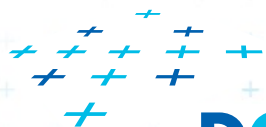
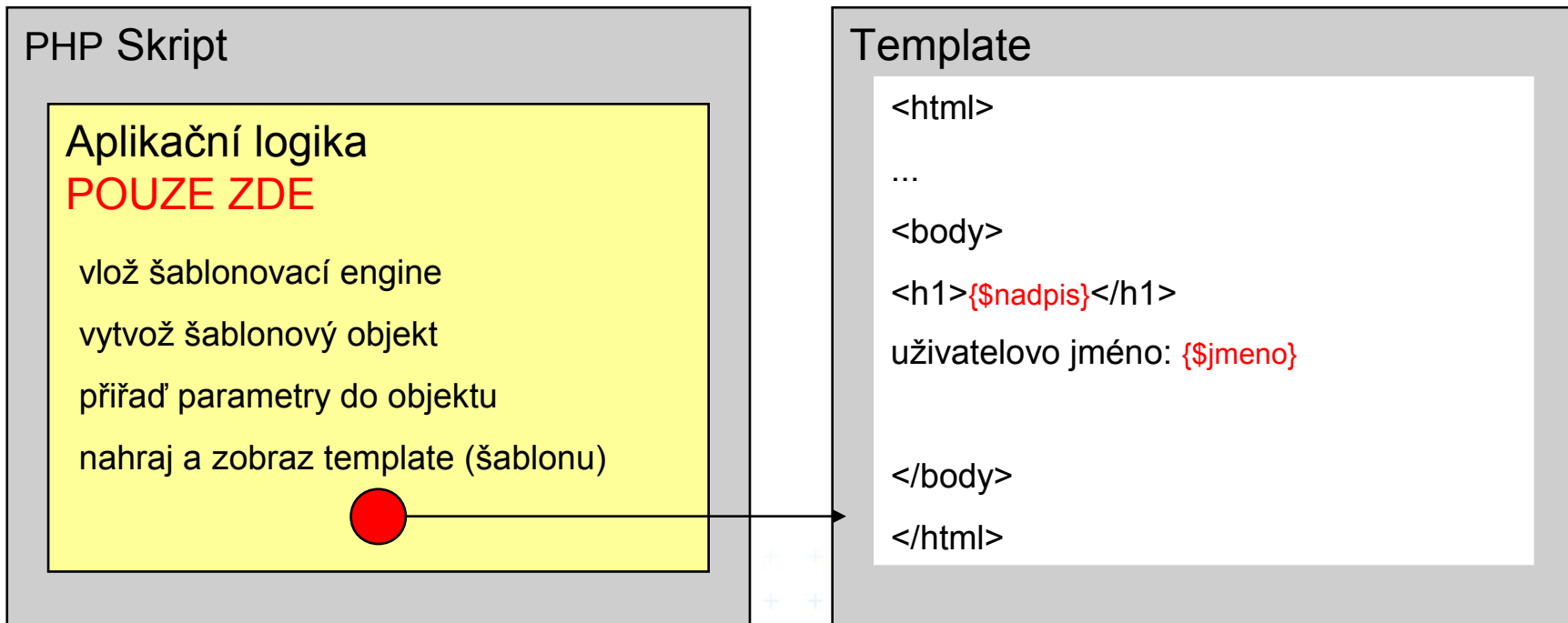
Čím se liší

- Schopností rychle parsovat
- Schopností provádět vlastní skriptovací kód
- Cache

- Náš dnešní favorit: Smarty



Jak to vypadá



Jak to funguje

1. Proved' PHP skript – aplikační logika (C, M)
2. Vytvoř se Templatovací objekt (V)
3. Tomuto objektu se přiřad' datové struktury (M) z ap. logiky
4. Zavolej metodu display
 1. je zapnuta cache? ano-jdi na bod 8
 2. ne – existuje přeložená šablona? ano – jdi na bod 5
 3. proved' lexikální analýzu šablony
 4. expanduj příkazy pseudoskriptu do podoby PHP kódu
 5. přeloženou šablonu ulož
 6. proved' příkazy pseudoskriptu
 7. výsledek ulož do cache (pokud je zapnuta)
 8. výsledek zobraz na standardní výstup



Jednoduchý projekt se šablonou smarty

PHP Skript

```
<?php
// inicializace smarty engine
require_once("init_smarty.php");

// vyrobim si data, v tomto pripade info o
datumu (to je model)
$datum = date("d.m.Y");

// vytvor sablonu
$templatovaci_objekt = new T_Template();

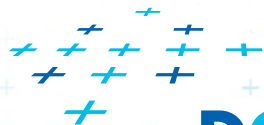
//prirad data do sablony
$templatovaci_objekt->assign("datum",
$datum);

// nech to zobrazit
$templatovaci_objekt-
>display('hello_world.html');

?>
```

Template

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
strict.dtd">
<html>
  <head>
    <title>
      Hello world
    </title>
  </head>
  <body>
    Dnes je: {$datum}
  </body>
</html>
```



Základní metody třídy Smarty (ale je jich více)

dokumentace je na:

<http://smarty.php.net/>

Přiřazení

assign()

assign_by_ref()

register_object()

Výstup

display()

fetch()

Rozšíření

register_function()

register_modifier()

Cache

is_cached()

clear_cache()



DCGI



Přiřazení

```
<?php
// toto cele je controller (page controller)

// inicializace smarty engine
require_once("init_smarty.php");

// vyrobim si data, v tomto pripade info o datumu (model)
$datum = date ("d.m.Y");

// vyrobim slozita data (model)
$pole_hodnot = array("Jmeno"=>"Martin", "Prijmeni"=>"Klima");

// vytvor sablonu
$templatovaci_objekt = new T_Template();

//prirad data do sablony
$templatovaci_objekt->assign("datum", $datum);
$templatovaci_objekt->assign_by_ref("clovek", $pole_hodnot);

// nech to zobrazit
$templatovaci_objekt->display('prirazeni.html');
?>
```

Přiřazení jednoduché hodnoty

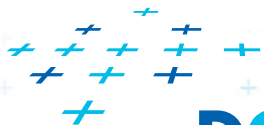
Přiřazení složité hodnoty



Použití v šabloně

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>
      Hello world
    </title>
  </head>
  <body>
    Dnes je: {$datum} <br/>
    Jmeno: {$clovek.Jmeno}<br/>
    Prijmeni: {$clovek.Prijmeni}
  </body>
</html>
```

Asociativní pole jsou
adresována pomocí tečkové
notace



Výstupy display & fetch

- display vrací výstup na standardní výstup
- fetch vrací výstup do proměnné

- použití: například mail se šablonou



Přiřazení

```
<?php
// inicializace smarty engine
require_once("init_smarty.php");

// vyrobim si data, v tomto pripade info o datumu
$datum = date ("d.m.Y");

// vyrobim slozita data
$pole_hodnot = array("Jmeno"=>"František", "Prijmeni"=>"Vomáčka");

// vytvor sablonu
$templatovaci_objekt = new T_Template();

//prirad data do sablony
$templatovaci_objekt->assign("datum", $datum);
$templatovaci_objekt->assign_by_ref("jmeno", $pole_hodnot);

// ziskej sloucený mail
$text_dopisu = $templatovaci_objekt->fetch("mail.tpl");

// odesli ho
mail("vomacka@post.cz", "Automaticky mail", $text_dopisu);

$templatovaci_objekt->assign("dopis",$text_dopisu );
// nech to zobrazit
$templatovaci_objekt->display('fetch.html');
?>
```

Získání výsledku mailu

Výstup html

Šablony

fetch.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>
      Ukazka sestaveni dopisu
    </title>
  </head>
  <body>
    Ahoj, prave Ti byl zaslan tento mail:<br/>
    <pre>
      {$dopis}
    </pre>
  </body>
</html>
```

mail.tpl

```
Vážený pane/paní {$jmeno.Jmeno} {$jmeno.Prijmeni}
blablabla
blablalba
blablabla

S pozdravem
  Martin Klíma
```



Přřazení – register object

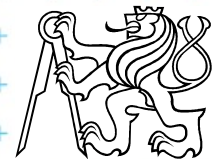
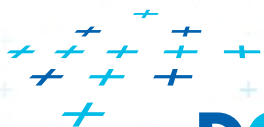
```
class Trida {  
    public function metodaJedna ($params, &$smarty) {  
        echo "toto je metoda 1";  
    }  
  
    public function metodaDva($params, &$smarty) {  
        echo "tot je metoda 1";  
    }  
}  
  
// inicializace smarty engine  
require_once("init_smarty.php");  
  
// instance Tridy  
$objekt = new Trida();  
  
// vytvor sablonu  
$templatovaci_objekt = new T_Template();  
  
// registruj novou funkci  
$templatovaci_objekt->register_object('obj',  
$objekt, array("metodaJedna"));  
// nech to zobrazit  
$templatovaci_objekt->display('register_object.html');
```

Registrace tridy a definice viditelných metod

Přiřazení – register object šablona

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>
      Registrace objektu
    </title>
  </head>
  <body>
    {obj->metodaJedna} <br/>
    {*obj->metodaDva*} <br/>
  </body>
</html>
```

Chyba, proto je to zakomentováno



Rozšiřování funkčnosti – register function

```
<?php
```

```
function zjistuCenu($params, &$smarty) {  
    $cena = 100;  
    switch ($params['mena']) {  
        case "CZK": return $cena." CZK";  
        case "EUR": return ($cena/26)." EUR";  
        case "USD": return ($cena/18)." USD";  
    }  
}
```

```
// inicializace smarty engine  
require_once("init_smarty.php");
```

```
// vytvor sablonu  
$templatovaci_objekt = new T_Template();
```

```
// registruj novou funkci  
$templatovaci_objekt->register_function('cena', 'zjistuCenu');
```

```
// nech to zobrazit  
$templatovaci_objekt->display('register_function.html');  
?>
```

Registruji novou funkci pod názvem "cena" ukazující na funkci "zjistuCenu"

kuk: register_function.php

Použití rozšíření v šabloně

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>
      Registrace funkce
    </title>
  </head>
  <body>
    Cena v CZK: {cena mena="CZK"} <br/>
    Cena v EUR: {cena mena="EUR"} <br/>
    Cena v USD: {cena mena="USD"} <br/>
  </body>
</html>
```

kuk: register_function.html



Úkol: zobraz seznam klientů

- Předpoklad je, že máme databázi uživatelů.
- Zobraz jenom jejich seznam
- Aplikační logika
 - vytvoří datovou strukturu – pole uživatelů
 - vytvoří objekt šablony
 - přiřadí pole
 - nechá zobrazit
- Prezentační logika
 - definuje, jak toto pole zobrazit



Jak vypadá funkce pro načtení pole

```
define("UZIVATEL","session_uzivatel");
define("DB_HOST", "localhost");
define("DB_UZIVATEL", "root");
define("DB_HESLO", "root");
define("DB_JMENO", "test");
// vraci vysledek sql dotazu nad db
function dotaz ($sql) {
    mysql_connect(DB_HOST, DB_UZIVATEL,DB_HESLO);
    mysql_select_db(DB_JMENO);
    $vysledek = mysql_query($sql);
    return $vysledek;
}
function poleUzivatelu () {
    $vysledek = array();
    // sestav dotaz
    $dotaz = "SELECT * FROM uzivatel";
    // nacti vysledek
    $sql_vysledek = dotaz($dotaz);
    // osetreni chyby pri komunikaci s db
    if (!$sql_vysledek) die("Nepodarilo se spojit s datovymi");
    // iteruj radky v db
    while ($radek = mysql_fetch_assoc($sql_vysledek))
        $vysledek[] = $radek;
    }
    // vrat pole radku
    return $vysledek;
}
```

Načteme tabulku z
MySQL databáze



Jak vypadá aplikační logika (C)

```
<?  
require_once("funkce.inc");  
require_once("init_smarty.php");  
  
// nacti pole uzivatelu - volam model  
$pole_uzivatelu = poleUzivatel();  
  
// vytvor sablonu  
$templatovaci_objekt = new T_Template();  
  
//prirad data do sablony - model do view  
$templatovaci_objekt->assign_by_ref("uzivatele", $pole_uzivatelu);  
  
// nech to zobrazit - view pouzije model  
$templatovaci_objekt->display('index.tpl');  
  
?>
```

načtení pole z DB

nový objekt
Smarty

přiřazení datových
struktur

spuštění procesu
zobrazení



Jak vypadá template

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=windows-
1250">
<title>Pokusna sablona</title>
</head>
<body>
<table summary="Seznam uzivatelu db">
{section name=i loop=$uzivatele}
    <tr>
        <td>{$uzivatele[i].Jmeno}</td>
        <td>{$uzivatele[i].Prijmeni}</td>
    </tr>
{/section}
</table>
</body>
</html>
```

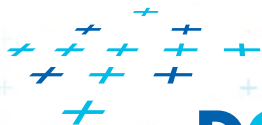
Chyba!!!



Jak vypadá template - správně

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=windows-
1250">
<title>Pokusna sablona</title>
</head>
<body>
<table summary="Seznam uzivatelu db">
{section name=i loop=$uzivatele}
    <tr>
        <td>{$uzivatele[i].Jmeno|escape:"html"}</td>
        <td>{$uzivatele[i].Prijmeni|escape:"html"}</td>
    </tr>
{/section}
</table>
</body>
</html>
```

Ochrana proti
HTML neplatnému
výstupu



Modifikátory

- Existuje řada připravených modifikátorů

capitalize	lower
cat	nl2br
count_characters	regex_replace
count_paragraphs	replace
count_sentences	spacify
count_words	string_format
date_format	strip
default	strip_tags
escape	truncate
indent	upper
	wordwrap

- Modifikátory lze řetězit

```
{$uzivatel.Jmeno|truncate|escape:"html"|capitalize}
```



Nový požadavek – text ve dvou sloupcích

- PHP kód zůstává beze změny
- Mění je jen prezentační logika
 - programátor nemusí dělat prakticky nic
 - vše je na grafikovi 😊



Modifikovaná prezentační logika

```
<table summary="Seznam uzivatelu db">
```

```
{section name=i loop=$uzivatele}
```

```
{if ($smarty.section.i.iteration mod 2) == 1}
```

Lichá položka?
Ano: vlož značku <tr>

```
<tr>
```

```
{/if}
```

```
<td>{$uzivatele[i].Jmeno|escape:"htmlall"}</td>
```

```
<td>{$uzivatele[i].Prijmeni|escape:"htmlall"}</td>
```

```
{if ($smarty.section.i.iteration mod 2) == 0}
```

Sudá položka?
Ano: ukonči značku
</tr>

```
</tr>
```

```
{/if}
```

```
{/section}
```

```
{if $smarty.section.i.rownum mod 2 == 1}
```

```
<td> --- </td>
```

```
<td> --- </td>
```

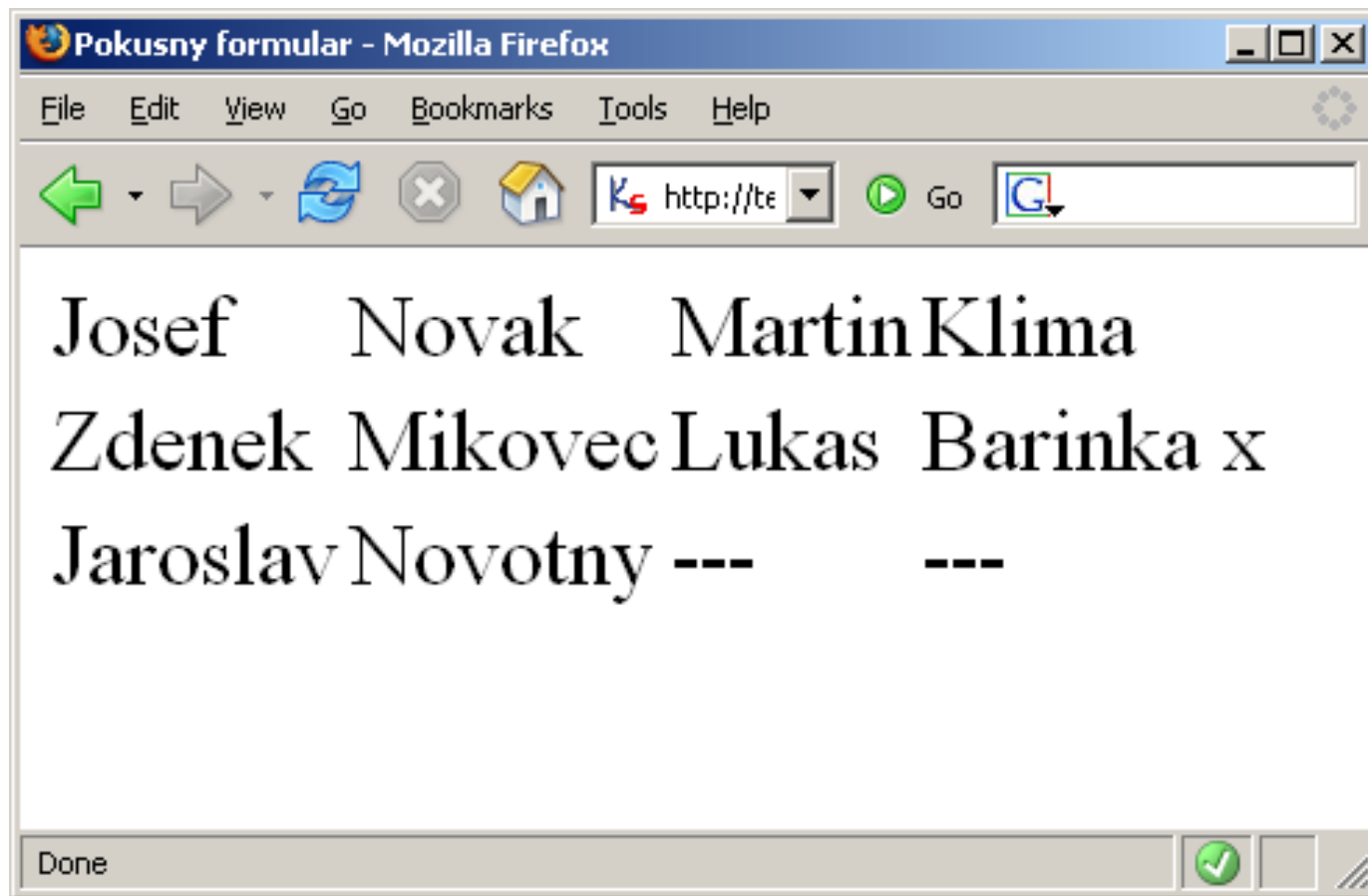
Co se stane, když je
lichý počet položek

```
</tr>
```

```
{/if}
```

```
</table>
```

Výsledek



Další požadavek: střídání řádků

Řešení:

- opět čistě jen prezentační logika



Prezentační logika – střídání řádků

```
<table summary="Seznam uzivatele db">
{section name=i loop=$uzivatele}
{if ($smarty.section.i.iteration mod 2) == 1}
  <tr class="{cycle values="lichy, sudy"}">
{/if}
  <td>{$uzivatele[i].Jmeno|escape:"htmlall"}</td>
  <td>{$uzivatele[i].Prijmeni|escape:"htmlall"}</td>
{if ($smarty.section.i.iteration mod 2) == 0}
  </tr>
{/if}
{/section}
{if $smarty.section.i.rownum mod 2 == 1}
  <td> --- </td>
  <td> --- </td>
</tr>
{/if}
</table>
```

Cyklicky střídá
parametry oddělené
čárkou



Střídání řádků – vložení CSS stylu

```
<title>Pokusny formular</title>
```

```
<style type="text/css">
```

```
tr.lichy { background-color: grey; }
```

```
tr.sudy { background-color: white; }
```

```
</style>
```

```
</head>
```

```
<table summary="Seznam uzivatelu db">
```

```
{section name=i loop=$uzivatele}
```

```
{if ($smarty.section.i.iteration mod 2) == 1}
```

```
<tr class="{cycle values="lichy, sudy"}">
```

```
{/if}
```

```
<td>{$uzivatele[i].Jmeno|escape:"htmlall"}</td>
```

```
<td>{$uzivatele[i].Prijmeni|escape:"htmlall"}</td>
```

```
{if ($smarty.section.i.iteration mod 2) == 0}
```

```
</tr>
```

```
{/if}
```

```
{/section}
```

```
{if $smarty.section.i.rownum mod 2 == 1}
```

```
<td> --- </td>
```

```
<td> --- </td>
```

```
</tr>
```

```
{/if}
```

```
</table>
```

Pozor!

Konflikt značek { a }

Smarty parser

interpretuje tyto značky



Střídání řádků – vložení CSS stylu

```
<title>Pokusny formular</title>
```

```
{literal}
```

```
<style type="text/css">
```

```
tr.lichy { background-color: grey; }
```

```
tr.sudy { background-color: white; }
```

```
</style>
```

```
{/literal}
```

```
</head>
```

```
<table summary="Seznam uzivatelu db">
```

```
{section name=i loop=$uzivatele}
```

```
{if ($smarty.section.i.iteration mod 2) == 1}
```

```
<tr class="{cycle values="lichy, sudy"}">
```

```
{/if}
```

```
<td>{$uzivatele[i].Jmeno|escape:"htmlall"}</td>
```

```
<td>{$uzivatele[i].Prijmeni|escape:"htmlall"}</td>
```

```
{if ($smarty.section.i.iteration mod 2) == 0}
```

```
</tr>
```

```
{/if}
```

```
{/section}
```

```
{if $smarty.section.i.rownum mod 2 == 1}
```

```
<td> --- </td>
```

```
<td> --- </td>
```

```
</tr>
```

```
{/if}
```

```
</table>
```

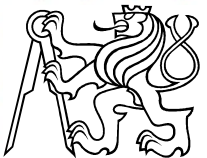
Neinterpretuj jako Smarty kód

Toto už není konflikt



Vkládání šablon – Composite View

- Šablony lze libovolně vnořovat
- Výhoda:
 - Řešení elementárních problémů
 - Elegance
 - Znovupoužitelnost
- Nevýhoda:
 - Více souborů
 - Problém s cache (může být výhoda)



Příklad

- Vytvořte seznam uživatelů
- Seznam bude obecný
- Každá položka v seznamu bude zobrazovat detail uživatele



Řešení: aplikační logika zůstává

```
require_once("funkce.inc");
require_once("init_smarty.php");

// nacti pole uzivatelu
$pole_uzivatelu = poleUzivatelu();

// vytvor sablonu
$templatovaci_objekt = & new T_Template();

//prirad data do sablony
$templatovaci_objekt->assign_by_ref("uzivatele", $pole_uzivatelu);

// nech to zobrazit
$templatovaci_objekt->display('index2.tpl');
```



Šablona první úrovně – rozložení seznamu

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=windows-1250">
<title>Composite View</title>
</head>
<body>
{foreach from=$uzivatele item=aktualni_uzivatel}
{include file="detail_uzivatele.tpl"}
{/foreach}
</body>
</html>
```

Iteruj přes pole
\$uzivatele
aktualni položku prirad
do promenne
\$aktualni_uzivatel

Vlož definici detailu



Šablona druhé úrovně – detail

Smarty komentář. Je odstraněn při překladu šablony

{* v této šabloně předpokládám, že informace o uživateli jsou v poli v proměnné \$aktualni_uzivatel*}

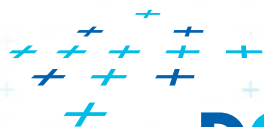
```
<hr>
```

```
<div>Jméno: {$aktualni_uzivatel.Jmeno}</div>
```

```
<div>Příjmení: {$aktualni_uzivatel.Prijmeni}</div>
```

```
<div>Logovací jméno: {$aktualni_uzivatel.LogovaciJmeno}</div>
```

Vložená šablona dědí definované proměnné



Šablona druhé úrovně – detail

Smarty komentář. Je odstraněn při překladu šablony

{* v této šabloně předpokládám, že informace o uživateli jsou v poli v proměnné \$aktualni_uzivatel*}

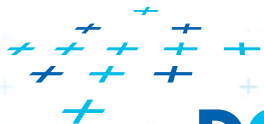
```
<hr>
```

```
<div>Jméno: {$aktualni_uzivatel.Jmeno|escape:"html"}</div>
```

```
<div>Příjmení: {$aktualni_uzivatel.Prijmeni|escape:"html"}</div>
```

```
<div>Logovací jméno: {$aktualni_uzivatel.LogovaciJmeno|escape:"html"}</div>
```

Vložená šablona dědí definované proměnné



Výsledek

Pokusny formular - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

← → ↻ × 🏠 Ks ▶

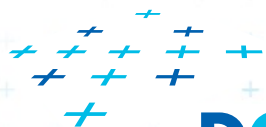
Jméno: Josef
Příjmení: Novak
Logovací jméno: novak

Jméno: Martin
Příjmení: Klima
Logovací jméno: xklima

Jméno: Zdenek
Příjmení: Mikovec
Logovací jméno: xmikovec

Jméno: Lukas
Příjmení: Barinka x
Logovací jméno: barinkl

Jméno: Jaroslav
Příjmení: Novotny
Logovací jméno: xnovotny



DCGI



Cache

- Jednou vygenerovaný výsledek může být uložen a použit znovu
- Výhoda:
 - Výrazné zrychlení odezvy
 - Méně dotazů do db
- Nevýhoda:
 - Zabírá prostor na disku
 - Aditivní kód
 - Uživatel nedostává aktuální data
 - Delší zpracování stránek, které nejsou v cache



Příklad – jen aplikační logika

```
require("setup.php");
```

```
// zavolam model
```

```
$uzivatele = Uzivatele::getList();
```

Zde **NE**šetříme čas

```
//vyrobim smarty objekt
```

```
$smarty = MojeSmarty::getInstance();
```

Zapni cache
s dobou života 120 sec

```
$smarty->caching = true;
```

```
$smarty->cache_lifetime = 120; // v sekundach
```

```
// priradim mu hodnoty z modelu
```

```
$smarty->assign("uzivatele", $uzivatele);
```

Zde šetříme čas

```
// zavolam view
```

```
$smarty->display("smarty_uzivatele2.html", "smarty3-caching/index.php");
```

Příklad – cache - oprava

```
require ("setup.php");

// toto je controller, pohrajeme si s cache

//vyrobim smarty objekt
$smarty = MojeSmarty::getInstance();
$smarty->caching = true;
$smarty->cache_lifetime = 120; // v sekundach

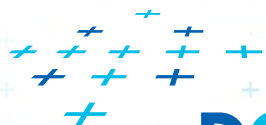
if (!$smarty->is_cached("smarty_uzivatele2.html")) {
    // zavolam model jen kdyz je to potreba
    $uzivatele = Uzivatele::getList();
    // priradim hodnoty z modelu
    $smarty->assign("uzivatele", $uzivatele);
}

// zavolam view
$smarty->display("smarty_uzivatele2.html");
```

Zapni cache
s dobou života 120 sec

Zde šetříme čas
když to lze

Zde šetříme čas



Debug - ladění

Umožňuje zobrazit parametry přiřazené k template objektu.

Do těla template stačí napsat značku **{debug}**



Výsledek

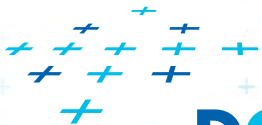
```
http://testserver - Smarty Debug Console - Mozilla Firefox

Smarty Debug Console
included templates & config files (load time in seconds):
no templates included
assigned template variables:
{$SCRIPT_NAME}      /smarty2/index2.php
{$app_name}         Test
{$uzivatele}       Array (5)
                   \r0 => Array (6)
                   \r  ID => 1
                   \r  LogovaciJmeno => novak
                   \r  Jmeno => Josef
                   \r  Prijmeni => Novak
                   \r  Heslo => nov
                   \r  Povolen => 1
                   \r1 => Array (6)
                   \r  ID => 2
                   \r  LogovaciJmeno => xklima
                   \r  Jmeno => Martin
                   \r  Prijmeni => Klima
                   \r  Heslo => klim
                   \r  Povolen => 1
                   \r2 => Array (6)
                   \r  ID => 3
                   \r  LogovaciJmeno => xmikovec
                   \r  Jmeno => Zdenek
                   \r  Prijmeni => Mikovec
                   \r  Heslo => mik
                   \r  Povolen => 1
                   \r3 => Array (6)
                   \r  ID => 4
                   \r  LogovaciJmeno => barinkl
                   \r  Jmeno => Lukas
                   \r  Prijmeni => Barinka x
                   \r  Heslo => bar
                   \r  Povolen => 1
                   \r4 => Array (6)
                   \r  ID => 8
                   \r  LogovaciJmeno => xnovotny
                   \r  Jmeno => Jaroslav
                   \r  Prijmeni => Novotny
                   \r  Heslo => nov
```



■ Životní cyklus šablony

- Prochází řadou filtrů
- Před překladem
- Po překladu
- Po vykonání



Prefilter

- Textový filter, kterým je prohnán template ještě předtím, než je zkompilován
- Hodí se k odstranění uživatelských komentářů, preprocessing obecně



Postfilter

- Textový filter, kterým je prohnán template poté, co byl zkompileván
- Hodí se např. k přidání nějaké spec. informace



Output filter

- Textový filter, kterým je prohnán template poté, co byl zkompilován
- Pracuje tedy nad kompletním výstupem
- Hodí se např. k zvýraznění některých slov, jejich potlačení, ochrana před vulgaritami atd.



Jak se filtry používají

```
require_once("funkce.inc");
require_once("init_smarty.php");
function muj_output_filter($tpl_output, &$smarty)
{
    // provede nejakou textovou manipulaci a vrati tento modifikovany
    text
    // budeme nahrazovat tyto vyrazy
    $vzor[] = '/Trabant/';
    $vzor[] = '/Tatra 613/';
    $vzor[] = '/Lada/';
    $nahrazeni[] = 'splášený vysavač';
    $nahrazeni[] = 'papalášfáro';
    $nahrazeni[] = 'Žigulík';
    // proved nahrazeni a vrat vysledek
    return preg_replace($vzor, $nahrazeni, $tpl_output);
}

// vytvor sablonu
$templatovaci_objekt = & new T_Template();

// registrace výstupní funkce
$templatovaci_objekt->register_outputfilter("muj_output_filter");

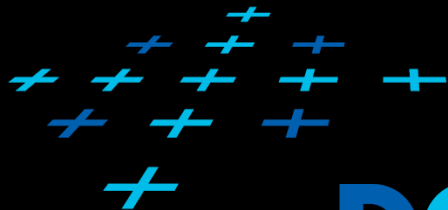
// nech to zobrazit
$templatovaci_objekt->display('smarty_filter1.tpl');
```

Jak se filtry používají – pokr.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=windows-1250">
<title>Pokus s filtrem</title>
</head>
<body>
<h1>Filtr</h1>
<p>Na této stránce se dějí zajímavé věci s filtry.</p>
<p>Franta říkal, že nejlepší auto je Trabant.</p>
<p>Pepa ale říkal, že on si koupí jedině Tatra 613.</p>
<p>Já si ale myslím, že nejlepší je Lada.</a>
</body>
</html>
```

Výsledek

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=windows-1250">
<title>Pokus s filtrem</title>
</head>
<body>
<h1>Filtr</h1>
<p>Na této stránce se dějí zajímavé věci s filtry.</p>
<p>Franta říkal, že nejlepší auto je splašený vysavač.</p>
<p>Pepa ale říkal, že on si koupí jedině papalášfáro.</p>
<p>Já si ale myslím, že nejlepší je Žigulík.</a>
</body>
</html>
```



DCGI

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

Používejte Smarty, je to dobré 😊

Děkuji za pozornost,

Martin Klíma