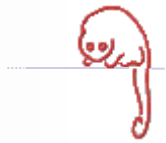
Tomáš Matoušek
Ladislav Prošek

Jan Benda
Martin Malý
Pavel Novák
Václav Novák

Phalanger

The PHP Language Compiler for the .NET Framework

Často kladené otázky



- Co je lepší, PHP nebo .NET?
  - chybně položená otázka
    - PHP je jazyk
    - .NET je platforma pro libovolný jazyk

- Co je lepší, PHP nebo C#?
  - opět nekorektní
    - PHP je dynamický jazyk
      - nemusí se dbát na typy – výhoda a zároveň nevýhoda
      - volání funkcí pomocí jména, vyhodnocování kódu za běhu, …
    - C# je staticky typovaný jazyk
      - každá proměnná musí mít deklarovaný svůj typ
    - záleží na účelu, pro který je jazyk zvolen, pak lze diskutovat o vhodnosti volby
      - "It's the purpose, what defines us."
    - problémy PHP
      - není jasné, co to vlastně PHP je (chybí specifikace, mírně chaotický design)

Mýty

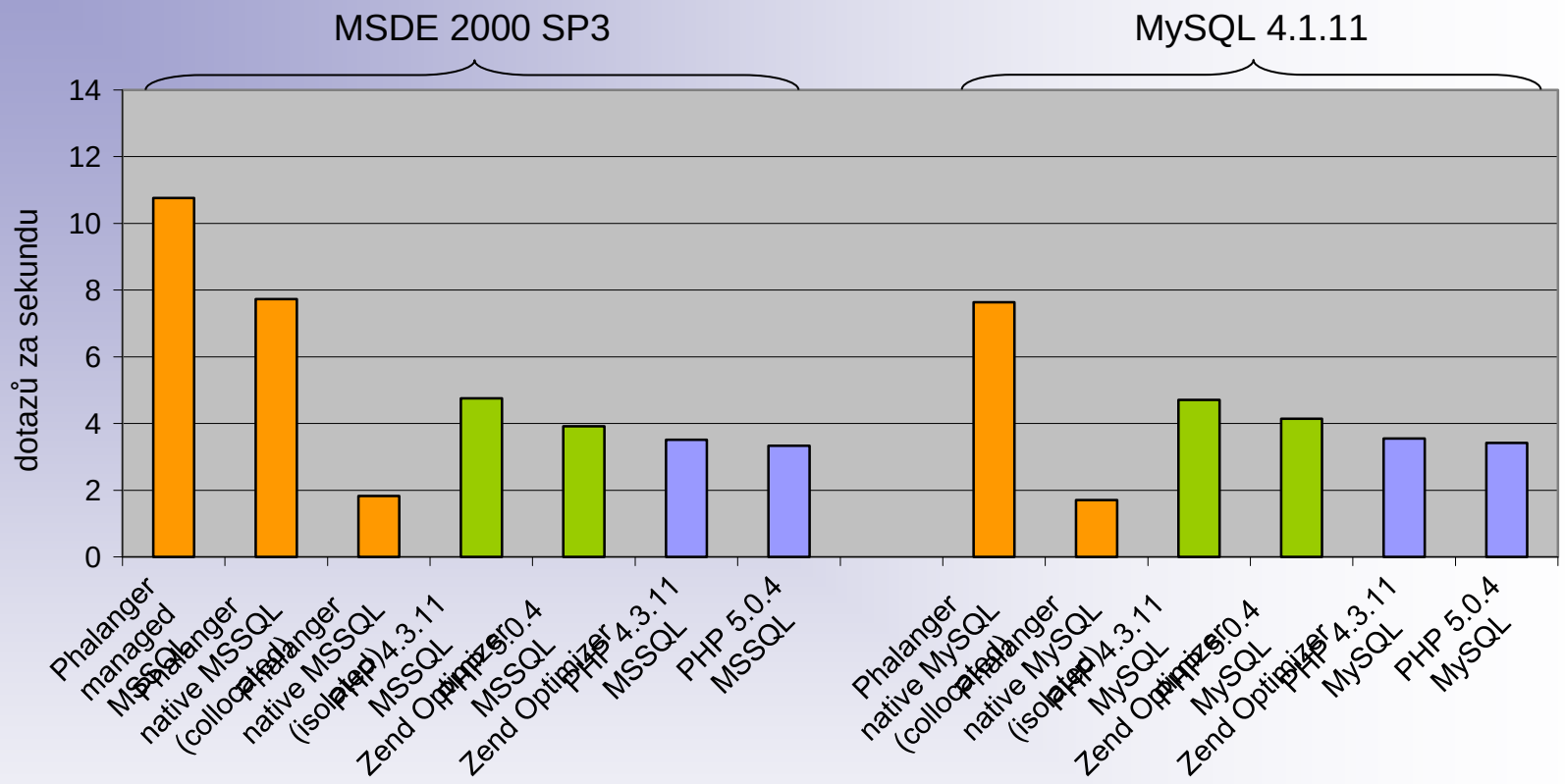
- .NET Framework je jen pro staticky typované jazyky
- PHP je interpretovaný jazyk, nelze kompilovat
- .NET Framework je pomalý
- .NET Framework je pouze komerční

- Phalanger dokazuje, že nic z toho neplatí!

Co je a co umí Phalanger?

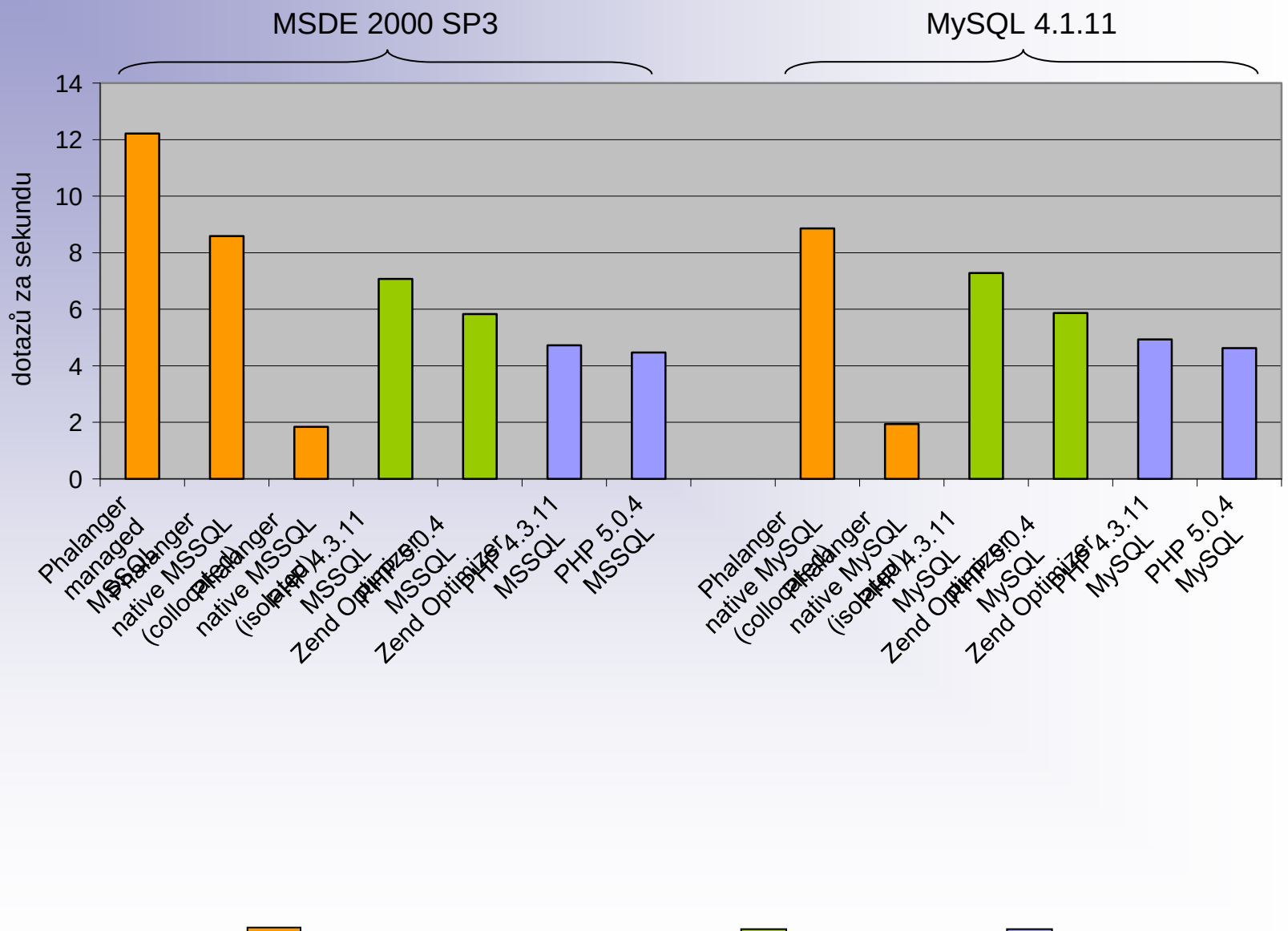
- **Kompilátor a běhové prostředí jazyka PHP na platformě .NET Framework**
  - od verze 2.0 i podpora implementace Mono
- Rozšiřuje množinu jazyků platformy .NET o PHP verze 4 a 5
  - od verze 2.0 i o nové prvky jazyka PHP 6
- Umožňuje běh existujících PHP web aplikací v prostředí ASP.NET
  - bez nutnosti změn v kódu
- Zvyšuje výkon existujících PHP web aplikací
- Zajišťuje vyšší spolehlivost web serveru hostujícího PHP aplikace
- Rozšiřuje možnosti PHP aplikací o funkcionalitu platformy .NET
  - umožňuje psát konzolové a Windows aplikace
  - od verze 2.0 přímý přístup k libovolným knihovnám platformy .NET
- Zpřístupňuje zkompilované PHP funkce a třídy ostatním .NET jazykům
- Nabízí možnost využití téměř všech PHP knihoven v libovolném jazyku platformy
- Integruje jazyk PHP do Visual Studia .NET
  - od verze 2.0 i do Visual Studia 2005

Srovnání výkonu (IIS 5.1)

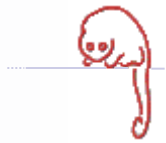


Phalanger 1.0 PHP + Zend Optimizer PHP

Srovnání výkonu (Apache 2.0.54)



Zvýšení výkonu



- použití Phalangeru významně zvýší výkon web aplikace
  - měřeno na aplikaci phpBB 2.0.13 (diskusní fórum)
    - parametry benchmarku
      - hardware: Intel Pentium M 1,4GHz, 1GB RAM
      - platforma: Windows XP Professional SP2
      - web servery: IIS 5.1, Apache 2.0.54
      - db servery: MSDE 2000 SP3, MySQL 4.1.11
      - Phalanger 1.0 Beta 3, PHP 5.0.4, PHP 4.3.11
  - nejvýkonnější měřená PHP konfigurace
    - zdarma: Apache, MySQL, PHP 4.3.11 — 4,93 req/s
    - $960/rok: Apache, MySQL, PHP 4.3.11, Zend Optimizer — 7,28 req/s
  - nejvýkonnější Phalanger konfigurace
    - Apache (mod_aspdotnet), MSDE, Managed MSSQL — 12,21 req/s
  - závěr
    - použití Phalangeru urychlilo phpBB o 67% resp. o 148%

- prostor pro další zvyšování výkonu
  - vyšší výkon s každou novou verzí .NET Frameworku
    - využití neustálého vylepšování JITteru, CLR, ASP.NET
  - další optimalizace ve Phalangeru

Obsah

- Představení systému Phalanger
- Možnosti aplikací založených na Phalangeru
  - konzolové a Windows aplikace
  - web aplikace – integrace do ASP.NET
  - konfigurace
- Phalanger Class Library
  - implementace PHP funkcí v C#
  - rozšiřitelnost knihovny
- Zpřístupnění PHP Extensions .NET aplikacím
  - využití existujícího nativního kódu v managed aplikaci
- Kompilace jazyka PHP
  - skripty jako jednotky kompilace
  - proměnné a funkce
  - objektově orientované rysy jazyka
- Verze 2.0
  - rozšíření PHP na plnohodnotný jazyk platformy .NET (PHP/CLR)

Součásti systému Phalanger

- Phalanger Core
  - kompilátor
    - lexikální a syntaktická analýza
      - využito nástrojů CsLex a GPPG (generátory lexeru a parseru pro C#)
    - generátor IL kódu
      - Reflection.Emit
      - kód je generován do souborů nebo jen do paměti
  - prostředí pro běh PHP aplikací
    - metody volané zkompilovaným kódem
    - integrace s ASP.NET serverem
- Phalanger Class Library
  - PHP funkce a třídy implementované v C#
  - přes 500 PHP funkcí
    - operace s poli, řetězci, regulárními výrazy, funkce pro práci s databází MS SQL, …
- Extension Manager
  - modul umožňující volat PHP funkce implementované v PHP extensions
- Integrace do VS.NET 2003 a 2005
  - nový typ projektu, zvýrazňování, kontrola syntaxe, …

Možnosti použití

- Web aplikace v PHP
  - konfigurace pomocí souborů Web.config
  - podpora pro různé styly vývoje aplikace
    - standardní vývoj PHP aplikací
      - aplikace = množina skriptů ve virtuálním adresáři na web serveru
      - „plochý" model (linear model)
      - edit-save-refresh
      - skripty se kompilují automaticky, programátor se kompilací nemusí zabývat
    - ASP.NET web aplikace (Phalanger 2.0)
      - aplikace je strukturovaná, tvořená z komponent, designery
      - část PHP kódu generována na základě XML souborů
      - explicitní kompilace a deployment
- Konzolové a Windows aplikace v PHP
  - výsledek kompilace je .exe soubor (assembly)
  - konfigurace pomocí .exe.config souborů

Web aplikace – plochý model

- skripty v daném virtuálním adresáři nastaveném jako web aplikace

- konfigurace virtuálního adresáře v IIS
  - stejná jako pro jiné ASP.NET aplikace
  - přípona .php mapována na ISAPI modul ASP.NET v1.1
    - c:\windows\microsoft.net\framework\v1.1.4322\aspnet_isapi.dll
  - dotaz od klienta je z IIS přeposlán ASP.NET worker processu

- konfigurace ASP.NET
  - nastavena při instalaci Phalangeru v Machine.config
  - přidána položka do seznamu HTTP handlerů
  - zajišťuje předání dotazu třídě implementované v Phalanger Core

```
<system.web>
  <httpHandlers>
    <add verb="*" path="*.php" type="PHP.Core.PageFactory, PhpNetCore, Version=1.0.0.0,..."/>
    <add verb="*" path="*.aspx" type="System.Web.UI.PageHandlerFactory" />
    ...
    <add verb="*" path="*" type="System.Web.HttpMethodNotAllowedHandler" />
  </httpHandlers>
</system.web>
```

Rozhraní pro spolupráci s ASP.NET

- rozhraní IHttpHandler
- třída implementující toto rozhraní zpracovává dotazy a generuje stránky
- vyvolána při přijetí dotazu poslaného do worker processu z ISAPI modulu
- vykonávána jedním vláknem, které dostalo na starosti zpracování dotazu
- pracuje v kontextu připraveném v ASP.NET
- zjistí, zda dotazovaný skript již byl zkompilován
  - tj. zda assembly je uložena v cache na disku
  - není zavolá kompilátor, který tuto assembly vytvoří
- načte assembly do paměti
- spustí globální kód skriptu

Konfigurace PHP web aplikací

- PHP definuje konfiguraci na několika úrovních
  - soubor php.ini
  - soubory .htaccess v adresářích (jen Apache server)
  - změna nastavení během běhu skriptu (funkcemi ini_get, ini_set, ...)
    - některé volby takto nelze modifikovat
- Phalanger
  - využívá hierarchickou konfiguraci ASP.NET (Web.config, Machine.config)
  - konfigurace uložená po načtení z XML v konfiguračních objektech
  - mapuje hodnoty v konfiguračních objektech na originální nastavení PHP
  - možnost zakázat změnu hodnoty v souborech na nižší úrovni
- konfigurační objekty
  - aplikační
    - konfigurace platná pro všechny dotazy do aplikace
    - nelze modifikovat v podadresářích aplikace
  - globální
    - obsahuje konfiguraci platnou pro všechny dotazy do daného adresáře
  - lokální
    - konfigurace měnitelná za běhu
    - asociovaná s dotazem

Konfigurační sekce

- definice specifické konfigurační sekce
  - při instalaci je přidán záznam do Machine.config
  - specifikována třída mající na starost parsování sekce
    - implementuje IConfigurationSectionHandler

```
<configuration>
  <configSections>
    <section name="phpNet" type="PHP.Core.ConfigurationSectionHandler,…"/>
  </configSections>
</configuration>
```

Příklad konfiguračního souboru

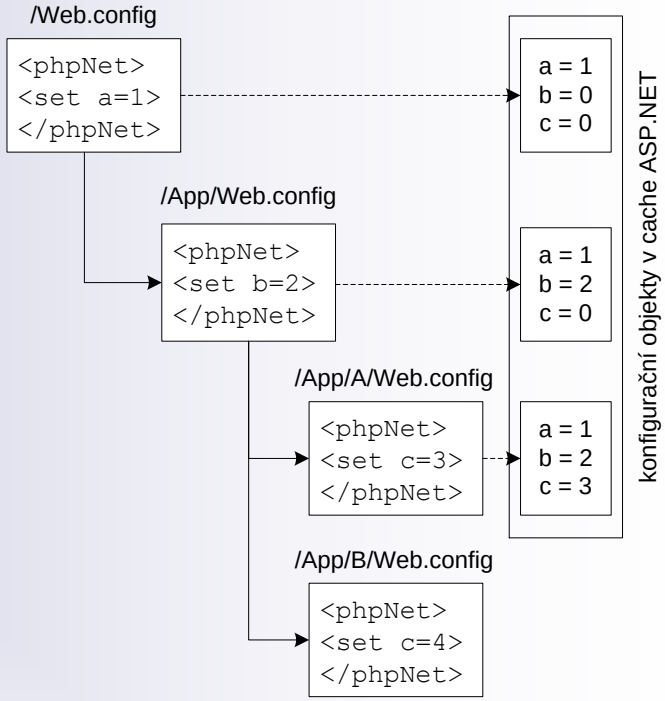
```
<configuration>
  <phpNet>
    <!-- Seznam knihoven (assemblies), které jsou použity v aplikaci -->
    <classLibrary>
      <add assembly="PhpNetClassLibrary,…" section="bcl"/>
      <add assembly="PhpNetMsSql,…" section="mssql" />
      <add assembly="php_sockets.mng,…" section="sockets" />
    </classLibrary>

    <!-- Nastavení Session Handleru -->
    <session-control>
       <set name="AutoStart" value="false" phpName="session.auto_start" />
       <set name="Handler" value="aspnet" />
    </session-control>

    <!-- Nastavení Phalanger Base Class Library -->
    <bcl>
      <mailer>
        <set name="SmtpServer" value="localhost" phpName="SMTP" allowOverride="false" />
        <set name="SmtpPort" value="25" phpName="smtp_port" />
        <set name="DefaultFromHeader" value="phalanger@localhost.cz" phpName="sendmail_from" />
      </mailer>
      <highlighting>
        <set name="Background" value="white" phpName="highlight.bg" />
        <set name="String" value="navy" phpName="highlight.string" />
      </highlighting>
    </bcl>

    <!-- Nastavení PHP extension "sockets" -->
    <sockets>
      <set name="sockets.use_system_read" value="On" />
    </sockets>
  </phpNet>
</configuration>
```

Atribut phpName slouží jen pro lepší orientaci uživatelů znalých konfigurace PHP (při parsování je ignorován)

Hodnotu tohoto nastavení již nepůjde změnit v souborech nižší úrovně.

Zpracování hierarchické konfigurace

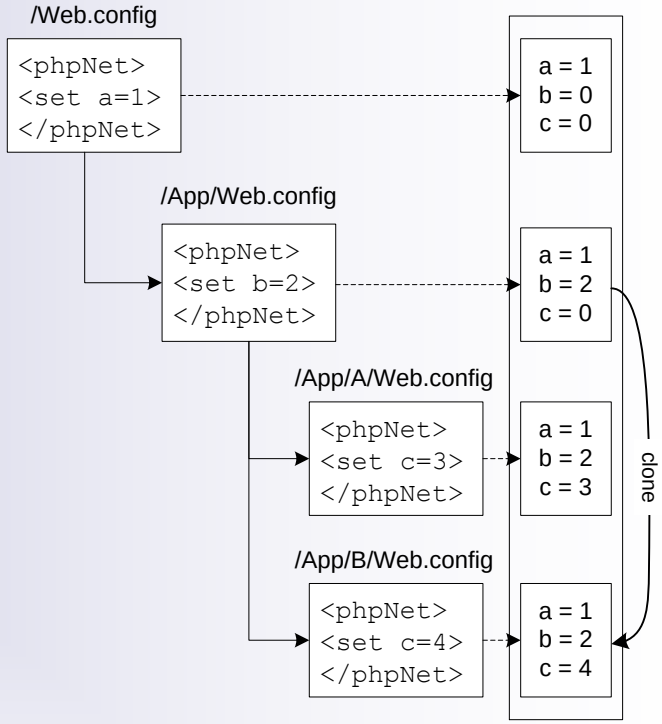
- ASP.NET zpracovává soubory Web.config nacházející se podél cesty dotazu
- konfigurační handler je volán pro každý zpracovávaný soubor
- handler vrací konfigurační objekt zpracovávaného souboru
- výsledek je uložen do cache
- do handleru je předán konfigurační objekt asociovaný s nadřazeným souborem



konfigurační objekty po dotazech
GET /App/A/file.php

Zpracování hierarchické konfigurace

- ASP.NET zpracovává soubory Web.config nacházející se podél cesty dotazu
- konfigurační handler je volán pro každý zpracovávaný soubor
- handler vrací konfigurační objekt zpracovávaného souboru
- výsledek je uložen do cache
- do handleru je předán konfigurační objekt asociovaný s nadřazeným souborem



konfigurační objekty po dotazech
GET /App/A/file.php
GET /App/B/file.php

Phalanger Class Library

- implementace funkcí a tříd použitelných v PHP
  - v libovolném jazyce podporujícím typový systém .NETu a custom attributes
- sada knihoven (assemblies)
  - Base Class Library
    - základní knihovna
    - funkce pro práci s poli, řetězci, …
  - native extension
    - obalena tzv. managed wrapperem generovaným automaticky (viz dále)
    - obsahuje managed stuby volající nativní kód
    - navenek se chová stejně jako ostatní knihovny
  - managed extension
    - reimplementace PHP extension v managed jazyku
    - transparentně nahrazuje nativní extension
    - rychlejší než použití nativní implementace
  - uživatelské knihovny
    - při dodržení jistých pravidel lze jednoduše přidat vlastní funkcionalitu

Deskriptor knihovny

- popisuje vlastnosti knihovny
  - jméno knihovny
  - seznam PHP extensions implementovaných knihovnou
- definuje chování knihovny
  - při jejím načtění
    - možnost registrovat nové session handlery, serializery apod.
  - při načítání její konfigurační sekce
    - parsování sekce
    - vytváření vlastních konfiguračních objektů

- třída deskriptoru
  - jedna třída může být deskriptorem i pro více knihoven
    - např. native extensions mají jednu třídu deskriptoru
- instance deskriptoru
  - pro každou načtenou knihovnu je vytvořena právě jedna instance deskriptoru

Obsah knihoven

- implementace
  - PHP konstant
    - literální fieldy označené atributem [ImplementsConstant]
  - PHP funkcí
    - statické metody označené atributem [ImplementsFunction]
  - PHP tříd a rozhraní
    - třídy zděděné (nepřímo) od třídy PhpObject
    - rozhraní obsahující IPhpInterface
    - označení atributem [ImplementsType]
- omezení kladené na implementace
  - signatury funkcí musí obsahovat jen typy dostupné PHP
  - PHP třídy musí mít některé specifické vlastnosti
  - a další ...

Adaptace volacích konvencí

- jazyk PHP má jiné konvence volání než .NET
  - předávání parametrů kopií
  - konverze typů
  - variabilní počet aktuálních parametrů
  - ...
- cíl návrhu knihoven
  - odstínit co nejvíce vlastní funkcionalitu od specifických vlastností jazyka PHP
  - možnost používat „pohodlně" knihovní funkce i z jiných jazyků
  - umožnit co nejobecnější implementace
- kompilátor obaluje volání knihovní funkce v PHP kódu
  - řízeno atributy definovanými v Core
    - např. [PhpDeepCopy], [CastToFalse], ...

Extensions

- PHP
  - mnoho funkcí a tříd implementováno v tzv. PHP extensions
    - php_gtk.dll
    - php_imap.dll
    - php_oci8.dll
    - php_pdf.dll
    - php_sockets.dll
    - …
- Phalanger
  - možnost využít funkcionalitu v extensions z libovolné .NET aplikace
    - managed wrappers
      - mají stejné rozhraní jako Class Libraries
    - managed reimplementace knihovny php4ts.dll
    - nastavitelná úroveň izolace
      - možnost hostovat nativní extensions v odděleném procesu

Managed wrappers

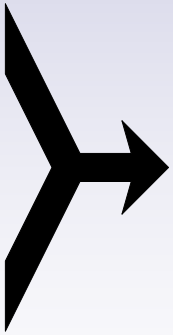
- assemblies generované automaticky z nativních extensions a typových informací ve formátu XML

php_sockets.dll

```
...
PHP_FUNCTION(socket_write)
{
  zend_parse_parameters(...);

  ZEND_FETCH_RESOURCE(...);

  // ...

  RETURN_LONG(...);
}
...
```

php_sockets.xml

```
...
<function returnType="int" name="socket_write" >
  <param type="resource" name="socket" />
  <param type="string" name="buf" />
  <param optional="true" type="int" name="length" />
</function>

...
```

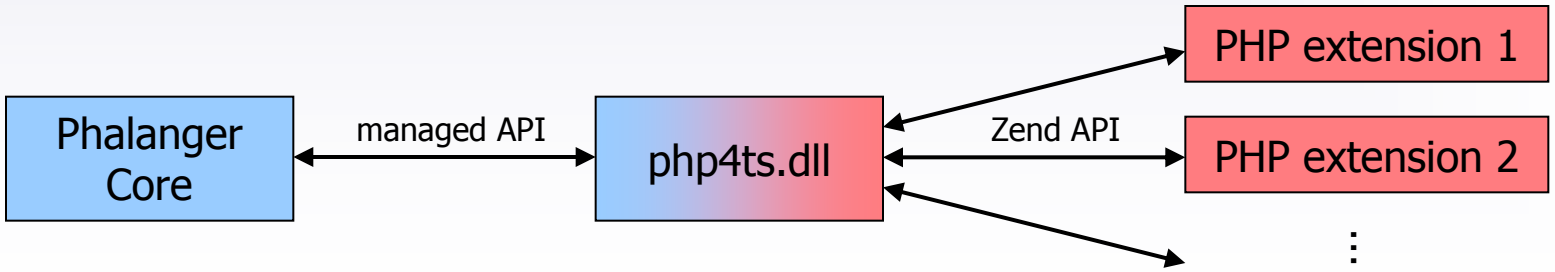


php_sockets.mng.dll

```
.method public hidebysig static int32 socket_write(
  class [PhpNetCore]PHP.Core.PhpResource socket,
  string buf,
  int32 length) cil managed
{
  .custom instance void
  PhpNetCore]PHP.Core.ImplementsFunctionAttribute ...
  // ...
  call object
  [PhpNetCore]PHP.Core.Externals::InvokeFunction(...)
  // ...
  ret
}

.method public hidebysig static int32 socket_write(
  class [PhpNetCore]PHP.Core.PhpResource socket,
  string buf) cil managed
{
  .custom instance void
  PhpNetCore]PHP.Core.ImplementsFunctionAttribute ...
  // ...
  call object
  [PhpNetCore]PHP.Core.Externals::InvokeFunction(...)
  // ...
  ret
}
```

php4ts.dll

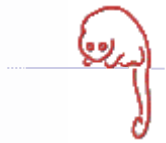
- základní součást PHP interpretu
- exportuje ~ 800 symbolů
  - běhová podpora pro extensions – Zend API
- reimplementována jako mixed-mode assembly s použitím C++/CLI
  - C++/CLI generuje IL i nativní kód
    - pomoci #pragma managed, #pragma unmanaged lze přepínat na úrovni funkcí
    - i funkce přeložené do IL mohou pracovat s unmanaged daty
      - obdoba klíčového slova unsafe v C#
    - jen malá režie při volání unmanaged z managed a naopak
  - adaptační vrstva mezi managed světem Phalangeru a unmanaged světem PHP extensions



Podporovaná funkcionalita

- volání funkcí implementovaných v extensions
  - ve wrapperech reprezentovány statickými metodami
- používání tříd implementovaných v extensions
  - do wrapperů nagenerovány managed verze tříd
  - podpora „magických" metod (__get, __set, __call, …)
- binding parametrů
  - dlouhodobé svázání proměnné ve skriptu s její nativní reprezentací uvnitř extension
  - např. oci_bind_by_name & oci_execute
- zpětná volání z extensions do skriptu

Izolace extensions



- extensions je možné izolovat v odděleném procesu
  - vhodné zejména pro web aplikace, kde by nedůvěryhodný nativní kód ohrožoval bezpečnost a stabilitu web serveru
  - nastavuje se pro každou extension zvlášť
    - atribut *isolated* v konfiguračním souboru
  - izolované extensions hostovány v procesu ExtManager
    - automaticky spouštěn, pokud neběží
- komunikace mezi Phalanger Core a procesem ExtManager využívá infrastrukturu .NET Remoting
  - každé volání funkce nebo metody implementované v extension znamená meziprocesovou komunikaci
  - výrazné snížení výkonu
    - nutná serizalizace parametrů
    - režie spojená s komunikací, synchronizací, …

Kompilace jazyka PHP

- podpora verze 6 se zpětnou kompatibilitou s verzemi 4 a 5
  - volba v konfiguraci

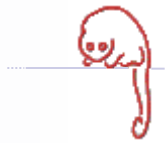
- hlavní problém jazyka PHP – chybějící specifikace
  - některé konstrukce jazyka nejsou zcela jednoznačně definovány
    - bylo třeba prozkoumat, jak se chová PHP interpret
  - kód spoléhající se na chyby v PHP interpretu není podporován
  - některé konstrukce jsou pokládány za chybné a nejsou povoleny
    - např. vícenásobné použití formálního argumentu funkce
      function f($a, $a) { … }
  - nepodporovány konstrukce dokumentované PHP jako deprecated
    - např. použití modifikátoru & na aktuální argumenty
      f(&$a); se chová stejně jako f($a);
    - kompilátor hlásí varování, pokud je tak nastaveno v konfiguraci

Kompilace skriptů

- standard mode
  - kompilace probíhá postupně po skriptech
  - inkluze
    - statické – optimalizace, pokud je kompilátor schopen najít cíl
    - dynamické – cíl je nalezen (a zpřípadně zkompilován) až za běhu aplikace
  - výsledek kompilace skriptu
    - web aplikace
      - jeden skript odpovídá jedné assembly
      - možnost zkompilovat všechny skripty do jedné assembly a distribuovat aplikaci bez zdrojového kódu
    - ostatní aplikace
      - všechny skripty aplikace jsou kompilovány do jedné assembly
      - přidán entry-point aplikace

- pure mode (verze 2.0)
  - zakázány inkluze a globální kód, povoleny jen globální deklarace
  - všechny deklarace jsou vidět globálně ve všech skriptech (jako v C#)

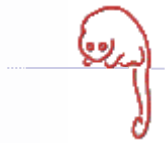
Kompilace kódu skriptu



- obsah skriptu
  - deklarace funkcí, tříd nebo rozhraní
  - globální kód
    - kód vně deklarací
    - text vně skriptovacích závorek <? ?>, <?php ?>, <% %>, …
      - zkonvertován na volání echo()

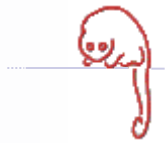
- výsledek kompilace
  - globální kód + deklarace funkcí statická třída Default (non-pure)
    - PHP funkce statické metody stejného jména
    - globální kód hlavní statická metoda Main()
  - PHP třída .NET třída
    - dědičnost zachována, základní třída je PhpObject
  - pomocné metody
    - Run apod.

Kontext skriptu



- množina objektů asociovaných s běžícím skriptem
  - instance třídy PHP.Core.ScriptContext
- vznik kontextu
  - PHP web aplikace – na začátku zpracování dotazu
  - konzolové a Windows PHP aplikace – při spuštění
  - ostatní aplikace (napsané v jiném jazyce) – manuálně
- předávání kontextu
  - parametr každé uživatelské PHP funkce
  - parametr konstruktoru každé uživatelské PHP třídy
  - ostatní kód (nepsaný v PHP)
    - kontext skriptu je umístěn v call-contextu aktuálního logického vlákna
- obsah kontextu
  - lokální konfigurační objekt
  - tabulka globálních proměnných
  - tabulka deklarovaných uživatelských PHP funkcí, tříd a rozhraní
  - a další …

Deklarace funkcí, tříd, rozhraní

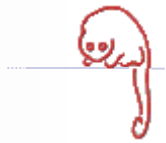


- PHP
  - neumožňuje zrušit ani přepsat již deklarovanou entitu
  - umožňuje podmíněnou deklaraci

```
if (podmínka) { function f(…) { … } } else { function f(…) { … } }
if (podmínka) { class C { … } } else { class C { … } }
```

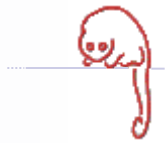
- Phalanger
  - každá z deklarací je zkompilována zvlášť
    - jména statických metod jsou pozměněna (např. f#0, f#1)
  - lokální tabulka deklarovaných typů a funkcí (na kontextovém objektu)
    - v okamžiku deklarace
      - je přidán delegát do tabulky deklarovaných funkcí resp. type-object do tabulky deklarovaných tříd a rozhraní
    - v okamžiku použití je v tabulce nalezena příslušná entita
  - globální tabulka typů a funkcí
    - načítaná pomocí reflexe použitých knihoven
    - několik fází načítání
      - nenačítají se najednou všechny typy, metody a funkce
      - mnohé nebudou potřeba

Proměnné



- PHP
  - dynamicky typované
    - v kódu není specifikován typ proměnné
    - proměnná má typ podle poslední operace na ní provedené
  - typy proměnných
    - bool, int, double, string, array, resource, null
    - uživatelské typy – třídy, rozhraní
  - proměnné mohou mít alias
    - modifikátor & (např. $a =& $b, function f(&$a) {}, …)
- Phalanger
  - statické typy
    - System.Object (proměnná bez aliasu)
    - Core.PhpReference (proměnná s aliasem)
  - dynamické typy
    - Boolean, Int32, Double, String, Core.PhpArray, Core.PhpResource
    - Core.PhpObject
      - společná nadtřída pro uživatelské PHP třídy, které nedědí z .NET třídy

Uložení proměnných



- globální proměnné
  - tabulka na kontextu skriptu
- lokální proměnné
  - optimalizovaná funkce
    - proměnné známé za překladu
      - standardní lokální proměnné na zásobníku
    - proměnné vznikající za běhu
      - lokální tabulka run-time proměnných
  - neoptimalizovaná funkce
    - lokální tabulka všech proměnných

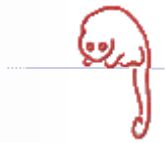
- lokální tabulky proměnných
  - mapují jména na hodnoty
  - vznikající až když je to opravdu nezbytné

- funkci lze optimalizovat, neobsahuje-li
  - dynamický kód (nevíme, jak se chová k proměnným)
  - kód potenciálně vyžadující tabulku všech proměnných

Objektový model

- PHP 5
  - vícenásobná dědičnost rozhraní
  - jednonásobná dědičnost tříd
  - public, protected, private, static, abstract, final
  - možnost přidávat fieldy do instancí za běhu
  - „magické" metody
    - __construct – konstruktor, ale dědí se
    - __destruct – destruktor
    - __get – vyvolána při čtení neexistujícího fieldu
    - __set – vyvolána při zápisu do neexistujícího fieldu
    - __call – vyvolána při volání neexistující metody
    - __sleep – vyvolána před serializací objektu
    - __wakeup – vyvolána po deserializaci objektu
    - __clone – kopírování objektu

Objektový model



- Phalanger
  - podporuje téměř celý objektový model PHP 5
    - __destruct funguje s jinou sémantikou
      - na .NETu není deterministická finalizace
      - objekty uklizeny při skončení zpracování dotazu
  - kompiluje PHP třídy a rozhraní do .NET tříd a rozhraní
    - zachovává hierarchii dědění
    - je možné dědit i od .NET tříd
      - drobná omezení
  - instancím .NET tříd lze v PHP také přidávat fieldy za běhu
    - cíl: co nejmenší rozdíl mezi .NET třídami a PHP třídami

Problémy

- některé kombinace atributů nejsou v .NET povoleny
  - abstract static, final static
  - řešení pomocí custom atributů [PhpAbstract], [PhpFinal]
- bázová třída nebo rozhraní nemusí být za kompilace známá
  - odložení kompilace na dobu běhu

```
class A extends B
{ /* … */ }
```

```
eval('class A extends B
{ /* … */ }');
```

- každý webový dotaz musí mít svoji kopii statických fieldů
  - statické fieldy opatřeny atributem [ThreadStatic]
  - líná inicializace před prvním přístupem
- třída může zvýšit viditelnost zděděného fieldu z protected na public
  - třída opatřena custom atributem [PhpPublicField("jméno_fieldu")]

Nevirtuální přístup

- překlad konstrukcí A::c, A::$x, A::f()
  - třída A je známá v době kompilace
    - přímý přístup
  - třída A není známá v době kompilace
    - mělo by stávat spíš jen ojediněle
    - kompilátor vyemituje volání operátoru, který
      - najde typ A v tabulkách (lokální a globální)
      - najde field/metodu v tabulce pro typ A
    - pomalejší
- objektové operátory dostávají jako parametr typový kontext
  - určuje typ, v jehož kontextu se má operace provést
  - umožňuje operátoru provést kontrolu přístupu (viditelnosti)
    - vynucení pravidel pro protected a private členy

Virtuální přístup

- překlad konstrukcí $a->x, $a->f()
  - obecně není možné určit typ $a v době kompilace
    - možnost statické analýzy, ale ne vždy dává výsledek
  - emituje se volání operátoru, který
    - z instance získá referenci na typ
    - na typu najde field/metodu v tabulce za kompilace známých fieldů/metod
    - pokud field nenalezen, pak hledá v tabulce run-time fieldů
      - tabulka asociovaná s instancí
  - optimalizace konstrukcí $this->x, $this->f()
    - typ $this bývá v době kompilace známý

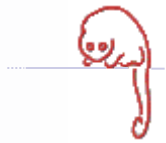
```
class A
{
    public $x = "foo";

    public function dump()
    { echo $this->x; }
};
```

```
class B extends A
{
    public function __get($a)
    { return "bar"; }
}

$a = new B;
unset($a->x);
$a->dump();
```

Namespaces

- nutné kvůli přístupu na .NET knihovny
  - zatím nejsou implementovány v PHP
    - je k dispozici jen návrh a pokusná implementace (patch)
  - Phalanger
    - podporuje téměř úplně návrh autorů PHP
- syntax jména
  - oddělovač - šesti-tečka :::
  - př. System:::Windows:::Forms:::Form
- deklarace
  - namespace My:::Namespace { class Trida { … } function Funkce { … } }
- importování symbolů
  - svázáno se souborem (jako v jazycích C#, Java, …)
  - syntax:
    - import namespace System;
    - import class System:::Windows:::Forms:::Form as F;
    - import function My:::Namespace:::Funkce as f;

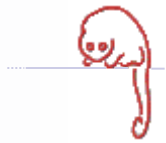
PHP/CLR



- rozšíření PHP o syntaxi potřebnou
  - pro práci s .NET třídami
  - pro umožnění plné integrace s komponentovým přístupem ASP.NET
- změny
  - v duchu jazyka
  - jednoduchá syntax, co nejméně omezení
  - co nejvíce dynamičnosti (nechceme yet another VB)

- rozšíření
  - generické typy a metody
  - custom attributes
  - getters, setters
  - parciální třídy
  - vynucené volání konstruktoru nadtřídy
  - libovolné identifikátory (bez omezení klíčových slov)
  - LINQ

PHP/CLR: Generika



- důvod
  - specifikace generických argumentů při používání .NET tříd
- gramatika
  - přidány nové tokeny <:, :>
  - nutnost pro zachování zpětné kompatibility
    - př. f(m < A, B > (C));
- deklarace generických typů a funkcí
  - př:

```
import namespace System:::Collections:::Generic;
class MyDict<:K, V:> extends Dictionary<:K, V:> { … }
function f<:K:> { … }
```

- defaultní hodnoty typových parametrů

```
class MyDict<:K = object, V = object:> extends Dictionary<:K, V:> {…}
```

- použití generických typů
  - př.

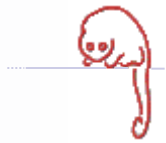
```
$t = "MyDict"; $k = "int"; $v = "List";
new MyDict();                // instanciace MyDict<:object,object:>
new MyDict<:int:>();         // instanciace MyDict<:int, object:>
new $t<:$k,$v<:int:>:>;      // instanciace MyDict<:int, List<:int:>:>
function f<:K:> { return new MyDict<:K,K:>(); }
```

PHP/CLR: Custom Attributes

- důvod
  - některé .NET aplikace vyžadují atributy
    - př.: Web services – [WebMethod] označuje metodu služby
  - výhodné i pro jiné účely
    - statické PHP fieldy musí mít sémantiku thread-static
    - jak v PHP/CLR aplikaci nadeklarovat skutečný statický field?
      - atribut [AppStatic]
- syntaxe
  - stejná jako v C#

```
class C extends System:::Web:::Services:::WebService
{
  [AppStatic]
  private static $x;

  [WebMethod(Description = "Foo", EnableSession = false)]
  public function GetFoo() { … }
}
```

PHP/CLR: Getters, Setters



- důvod
  - přepsání zděděných .NET properties
    - .NET property a field jsou z hlediska PHP totéž

- syntax

```
import namespace System:::Windows:::Forms;
class MyForm extends Form
{
  protected $DefaultSize
  {
    function __get() { return new Size(10, 10); }
  }
}
```

PHP/CLR: Language INtegrated Query

- zobecnění foreach cyklu
  - generátory dat, filtry, projekce, třídiče, …

C# 3.0:

```
var result =
  from c in customers
  where c.City == "London"
  from o in c.Orders
  where o.OrderDate.Year == 2005
  select new {c.Name, o.OrderID, o.Total}
```

- důvod
  - být „in“ ☺
  - může zpřehlednit program, zobecnit přístup k datům
- syntax

PHP/CLR:

```
$result =
  from $customers as $c
  where $c->City == "London"
  from $c->Orders as $o
  where $o->OrderDate->Year == 2005
  select array($c->Name, $o->OrderID, $o->Total)
```