Jan Benda
Martin Malý
Tomáš Matoušek
Pavel Novák
Václav Novák
Ladislav Prošek

Phalanger

The PHP Language Compiler for the .NET Framework

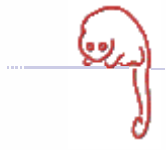
Obsah

- Představení projektu 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
- Kompilace jazyka PHP
  - skripty jako jednotky kompilace
  - proměnné a funkce
  - objektově orientované rysy jazyka
- Zpřístupnění PHP Extensions .NET aplikacím
  - využití existujícího nativního kódu v managed aplikaci

Phalanger

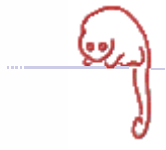
- Rozšiřuje množinu jazyků platformy .NET o PHP verze 4 a 5
- Umožňuje běh existujících PHP web aplikací v prostředí ASP.NET bez nutnosti větších změn v jejich kódu
- Zvyšuje výkon existujících PHP web aplikací
- Zajišťuje vyšší spolehlivost web serveru hostujícího PHP aplikace
- Rozšiřuje možnosti PHP aplikací o funkcionalitu platformy .NET
- Zpřístupňuje zkompilované PHP funkce a třídy a všechny knihovny funkcí ostatním .NET jazykům
- Integruje jazyk PHP do Visual Studia .NET

Součásti projektu Phalanger



- Phalanger Core
  - kompilátor
    - lexikální a syntaktická analýza
      - využito nástrojů flex a bison
    - generátor IL kódu
      - Reflection.Emit
      - kód je generován do souborů nebo jen do paměti
  - prostředí pro běh PHP aplikací
    - metody volané zkompilovaným kódem
    - zajištění spolupráce 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
  - nový typ projektu, zvýrazňování a kontrola syntaxe

Možnosti využití



- Web aplikace v PHP
  - množina skriptů ve virtuálním adresáři na web serveru
  - konfigurace pomocí souborů Web.config
- Konzolové a Windows aplikace v PHP
  - výsledek kompilace je .exe assembly
  - konfigurace pomocí .exe.config souborů
- Knihovny PHP funkcí a tříd
  - výsledek kompilace je .dll
  - možnost volaní PHP funkcí a dědění z PHP tříd v C#, VB.NET, …
- Funkcionalita v Class Library a PHP Extensions
  - lze používat z libovolné .NET aplikace
  - možnost přidávat nové knihovní funkce a třídy do PHP

Web aplikace

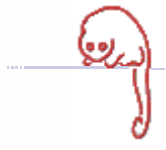
- skripty v daném virtuálním adresáři nastaveném jako web aplikace

- 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>
```

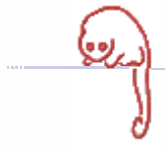
Rozhraní pro spolupráci s ASP.NET



- interface System.Web.IHttpHandlerFactory
  - vyrábí instance třídy implementující System.Web.IHttpHandler
  - musí být thread-safe
- interface System.Web.IHttpHandler
  - třída implementující toto rozhraní zpracovává dotazy a generuje stránky
  - metodou ProcessRequest(HttpContext context)

- metoda ProcessRequest
  - 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
    - property: Request, Response, Server, Session, Cache, ...
  - zjistí, zda dotazovaný skript již byl zkompilován
    - tj. zda assembly je uložena v cache na disku (adresář HttpRuntime.CodegenDir)
    - není → zavolá třídu kompilátoru, která tuto assembly vytvoří
  - načte assembly do paměti
    - pomocí Assembly.LoadFrom()
  - spustí globální kód skriptu
    - pomocí MethodInfo.Invoke()

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 System.Configuration.IConfigurationSectionHandler

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

Příklad konfiguračního souboru

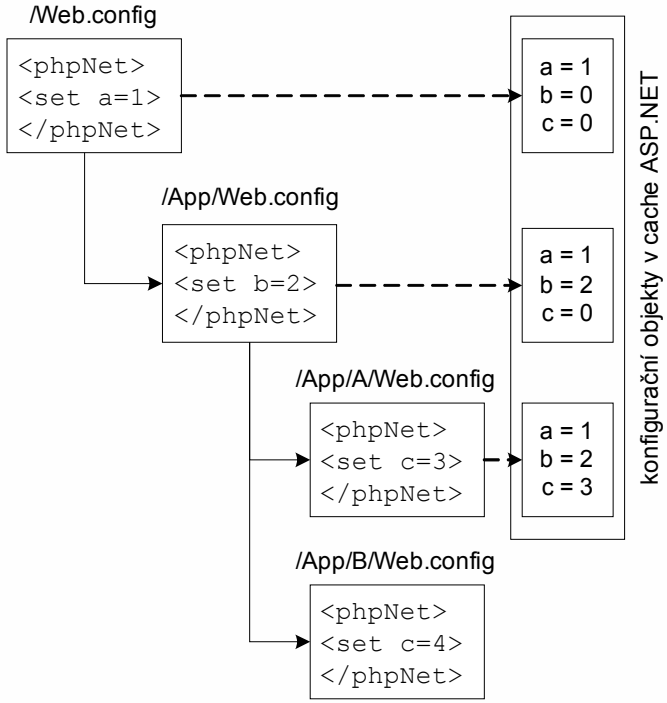
```
<configuration>
  <phpNet>
    <!-- Seznam knihoven Class Library, které jsou použity v aplikaci -->
    <classLibrary>
      <add assembly="PhpNetClassLibrary,…" section="bcl"/>
      <add assembly="PhpNetMsSql,…" section="mssql" />
      <add assembly="php_sockets.mng,…" section="sockets" />
 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

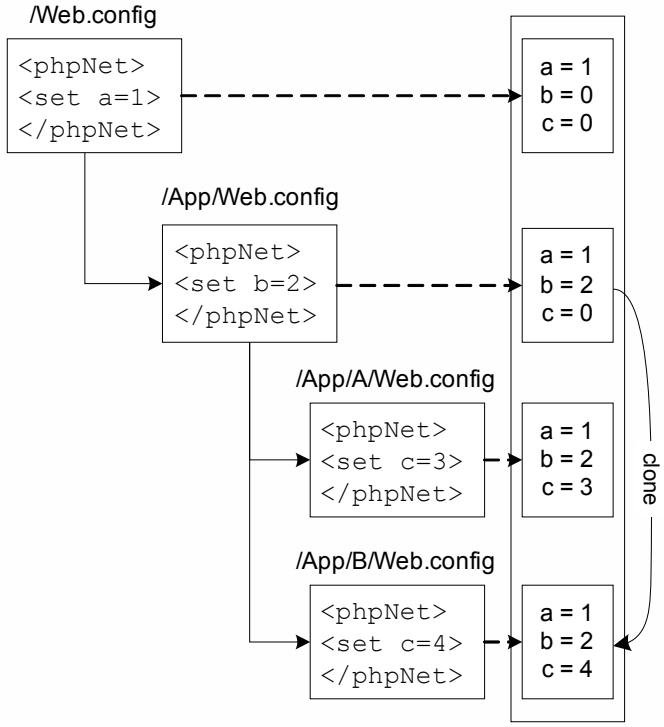
- ASP.NET zpracovává soubory Web.config nacházející se podél cesty dotazu
- konfigurační handler je volán pro každý zpracovávaný soubor
- handler vrací konfigurační objekt zpracovávaného souboru
- výsledek je uložen do cache
- do handleru je předán konfigurační objekt asociovaný s nadřazeným souborem



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

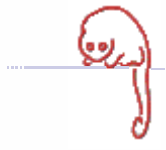
Zpracování hierarchické konfigurace

- 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



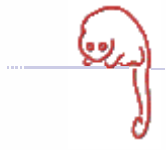
konfigurační objekty po dotazech
GET /App/A/file.php
GET /App/B/file.php

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

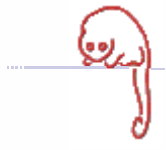
Deskriptor knihovny



- 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

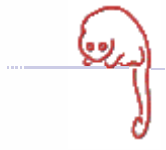
Obsah knihoven



- 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

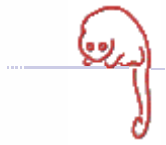
- omezení kladené na implementace
  - signatury funkcí musí obsahovat jen typy dostupné PHP
  - PHP třídy musí mít některé specifické vlastnosti
  - a další …

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], …

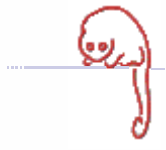
Kompilace jazyka PHP



- podpora verze 5 se zpětnou kompatibilitou s verzí 4
  - volba v konfiguraci

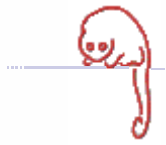
- hlavní problém jazyka PHP – chybějící specifikace
  - některé konstrukce jazyka nejsou zcela jednoznačně definovány
    - bylo třeba prozkoumat, jak se chová PHP interpret
  - kód spoléhající se na chyby v PHP interpretu není podporován
  - některé konstrukce jsou pokládány za chybné a nejsou povoleny
    - 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



- kompilace probíhá postupně po skriptech
- výsledek kompilace skriptu
  - web aplikace
    - jeden skript odpovídá jedné assembly
    - přidána metoda ProcessRequest
      - vyvolána z metody ProcessRequest HTTP handleru
      - spouští hlavní metodu skriptu Main()
  - ostatní aplikace
    - všechny skripty aplikace jsou kompilovány do jedné assembly
    - přidána metoda Run
      - entry-point aplikace
      - spouští hlavní metodu skriptu Main()

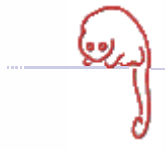
Kompilace kódu skriptu



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

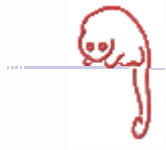
- výsledek kompilace
  - globální kód + deklarace funkcí → statická třída Default
    - PHP funkce → statické metody stejného jména
    - globální kód → hlavní statická metoda Main()
  - PHP třída → .NET třída
    - dědičnost zachována, základní třída je PhpObject
  - pomocné metody
    - Run, ProcessRequest,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

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

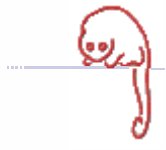


- PHP
  - neumožňuje zrušit ani přepsat již deklarovanou entitu
  - umožňuje podmíněně deklarovat entitu

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

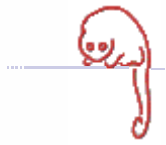
- Phalanger
  - každá z deklarací je zkompilována zvlášť
    - jména statických metod jsou pozměněna (např. f$mdecl0, f$mdecl1)
  - v okamžiku deklarace
    - je přidán delegát do tabulky deklarovaných funkcí resp. type-object do tabulky deklarovaných tříd a rozhraní
  - v okamžiku použití je v tabulce nalezena příslušná entita
  - obecný mechanismus pro použití neznámé entity
    - nejprve se hledá v tabulkách
    - poté v knihovnách

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
  - 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 třídy

Uložení proměnných

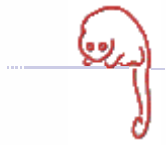


- globální proměnné
  - tabulka na kontextu skriptu
- lokální proměnné
  - optimalizovaná funkce
    - proměnné známé za překladu
      - standardní lokální proměnné na zásobníku
    - proměnné vznikající za běhu
      - lokální tabulka run-time proměnných
  - neoptimalizovaná funkce
    - lokální tabulka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

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

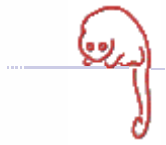
- funkci lze optimalizovat, neobsahuje-li
  - dynamický kód (nevíme, jak se chová k proměnným)
  - kód potenciálně vyžadující tabulku všech proměnných

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 – obdoba ICloneable.Clone

Objektový model

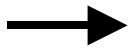


- Phalanger
  - podporuje téměř celý objektový model PHP 5
    - zatím nefunguje __destruct
      - chybí deterministická finalizace
  - kompiluje PHP třídy a rozhraní do .NET tříd a rozhraní
    - zachovává hierarchii dědění
  - společný abstraktní předek všech tříd PHP.Core.PhpObject
    - obsahuje tabulku fieldů přidaných za běhu PhpObject.RuntimeFields
    - implementuje funkcionalitu společnou pro všechny třídy
      - podpora konverzí, serializace, …

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

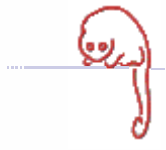
```
class A extends B
{ /* … */ }
```



```
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 (System.Type) v interních tabulkách nebo pomocí reflexe
      - přístoupí na field nebo vyvolá metodu pomocí reflexe
    - pomalé
- objektové operátory dostávají System.RuntimeTypeHandle jako jeden z parametrů
  - určuje typ, v jehož kontextu se má operace provést
  - umožňuje operátoru provést kontrolu přístupu (viditelnosti)
    - vynucení pravidel pro protected a private členy

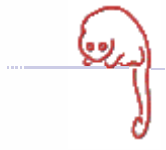
Virtuální přístup

- překlad konstrukcí $a->x, $a->f()
  - obecně není možné určit typ $a v době kompilace
  - běžné konstrukce – je třeba se vyvarovat pomalé reflexe
  - vždy se emituje volání operátoru, který
    - ověří, že $a je instancí potomka třídy PhpObject
    - zavolá na $a metodu __GetFieldTable resp. __GetMethodTable, čímž dostane referenci na tabulku fieldů resp. metod dané třídy
    - dotazem na tabulku získá "handle" k danému fieldu nebo metodě, na kterém provede požadovanou operaci
  - optimalizace konstrukcí $this->x, $this->f()
    - typ $this bývá v době kompilace známý

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

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

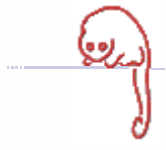
```
class B extends A
{
    public function __get($a)
    { return "bar"; }
}

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

Interoperabilita



- třídy napsané ve Phalangeru je možné konzumovat nebo rozšiřovat v jiných .NET jazycích
  - opačným směrem to v současné době přímo nefunguje
    - je nutné napsat proxy třídu odvozenou od PhpObject a ručně implementovat několik pomocných metod
    - automatické generování proxy tříd plánováno na některou z dalších verzí Phalangeru
- ukázka
  - použití PHP třídy v C# programu

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

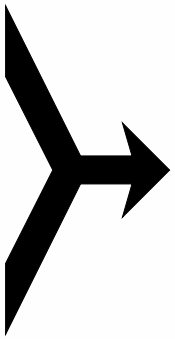
- assemblies generované automaticky z nativních extensions a typových informací ve formátu XML

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>

...
```

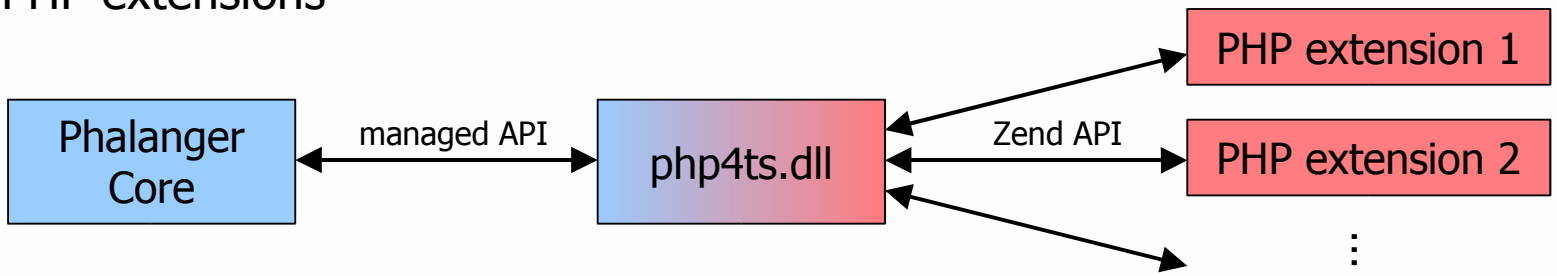


php_sockets.mng.dll

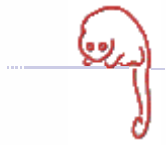
```
.method public hidebysig static int32 socket_write(
  class [PhpNetCore]PHP.Core.PhpResource socket,
  string buf,
  int32 length) cil managed
{
  .custom instance void
  PhpNetCore]PHP.Core.ImplementsFunctionAttribute ...
  // ...
  call object
  [PhpNetCore]PHP.Core.Externals::InvokeFunction(...)
  // ...
  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++ with Managed Extensions
  - MC++ generuje IL i nativní kód
    - pomoci #pragma managed, #pragma unmanaged lze přepínat na úrovni funkcí
    - i funkce přeložené do IL můžou 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

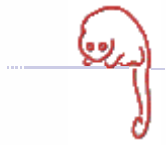


Podporovaná funkcionalita



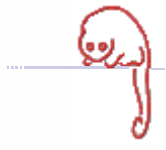
- volání funkcí implementovaných v extensions
  - ve wrapperech reprezentovány statickými metodami
- používání tříd implementovaných v extensions
  - do wrapperů nagenerovány managed verze tříd
  - podpora „magických" metod
- binding parametrů
  - dlouhodobé svázání proměnné ve skriptu s její nativní reprezentací uvnitř extension
  - např. oci_bind_by_name & oci_execute
- zpětná volání z extensions do skriptu
- podpora nových typů streamů
  - např. „compress.zlib://" implementovaný v php_zlib.dll

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, …

.NET Remoting



- komplikace způsobené rozdělením aplikace do více procesů
  - php4ts.dll v procesu ExtManager nezná metadata uživatelských tříd
    - nutné transformovat parametry před vzdáleným voláním
  - zpětná volání obsluhována „náhodným" vláknem z poolu
    - vyřešeno asynchronním vzdáleným voláním s explicitním čekáním na callback

- komunikace s izolovanými extensions využívá Shared Memory Channel
  - komunikační kanál pro .NET Remoting
  - přenáší data přes sdílnou paměť
    - funguje jen v rámci jednoho počítače
  - rychlejší alternativa k TcpChannel a HttpChannel, které jsou součástí .NET Framework BCL