PERSISTENT APPLICATION CONTAINER FOR PHP PLATFORM

Bc. Jiří Matula[1]

[1] *Ostravská univerzita v Ostravě, Přírodovědecká fakulta, Katedra informatiky a počítačů, 30. dubna 22, 701 03 Ostrava, +420 602 743 880, matula.jirik@gmail.com*

Abstrakt

Dnešní aplikace se obecně stávají mnohem komplexnějšími, přičemž flexibilní architektura aplikace je mnohem důležitější než počet implementovaných funkcionalit. Toto si žádá použití kontejnerů zodpovědných za správu závislostí v objektově orientovaných řešeních. Platforma PHP skrývá v sobě možnost implementovat kontejner, který by setrvával v operačním paměti a mohl by tak zajistit sdílení objektů mezi skripty navzdory faktu, že PHP je skriptovacím jazykem. Celý koncept persistentního aplikačního kontejneru zlepšuje tak výkon, znovupoužitelnost komponent a umožňuje adaptivně kešovat objekty v PHP aplikacích.

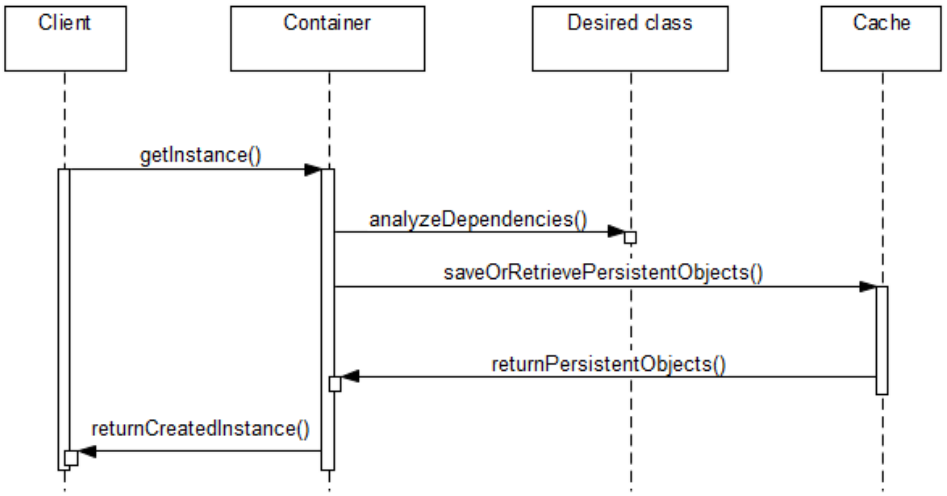
Klíčová slova: *dependency injection; cache; kontejner; PHP*

Dependency injection a kontejnery

Dependency injection (dále DI) je technika, jež odebírá třídám zodpovědnost za získávání instancí závislých tříd [1]. Toto vede k rozvolnění vazeb mezi komponenty a umožňuje stavět přizpůsobivější aplikace. Ovšem delegovat zodpovědnost za získávání instancí nelze do nekonečna, tudíž jednotlivé instance musí být nakonec někým vytvořeny. Tuto zodpovědnost má na starosti právě DI kontejner [2].

Návrh persistentního aplikačního kontejneru

Celý návrh konceptu vychází z kombinace uplatnění principů DI a kešování. Předáním odpovědnosti tříd za získávání instancí, je kontejner schopen získat tyto instance téměř odkudkoliv. S tímto připadá v úvahu serializovat již jednou vytvořené objekty, uložit do operační paměti (cache) a opět jej využít v rámci běhu následujících skriptů. Důležitým faktem zůstává, že odpadá jejich opětovná inicializace prováděná v konstruktoru, jelikož ne vždy všechny objekty je totiž rychlé vyrobit. Tímto pak dochází ke zmenšení nároků aplikace samotné a zkrácení doby její odezvy.



Obrázek 1. UML sekvenční diagram persistentního aplikačního kontejneru v aplikaci.

Ovšem abychom mohli uplatnit tzv. persistentní objekty a využít potenciál kontejneru, musí splňovat jejich definice tříd následující podmínky:

1. Každý persistentní objekt musí umožňovat serializaci.
2. Persistentní objekty nesmí obsahovat neserializovatelné třídní členy.
3. Statické členy nesmí být součástí persistentního objektu.

Výsledky a diskuse

Využití persistentní aplikační kontejneru je vhodné zejména pro aplikace, u kterých jsou kladeny vysoké nároky na přizpůsobitelnost architektury. Je to jeden z předpokladů mnoha projektů vyvíjených v agilním prostředí.

Pokud zohledníme tuto perspektivu, nativní implementace má následující problémy:

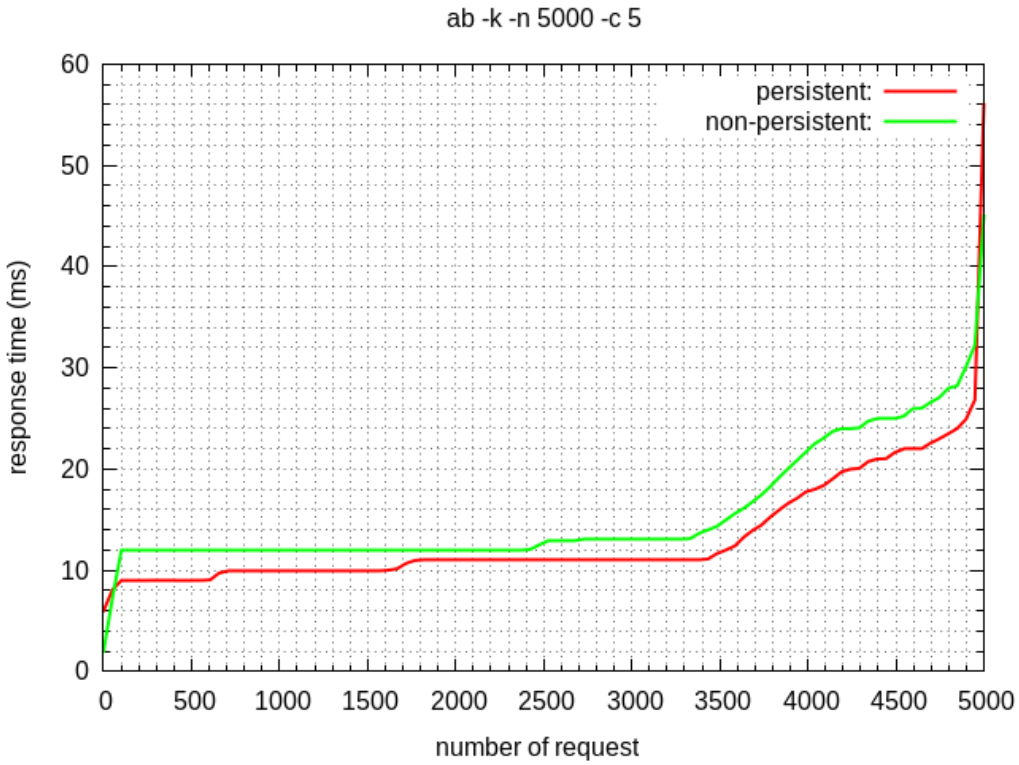
1. Třídy jsou zodpovědné za získávání instancí závislých tříd.
2. Konfigurace jednotlivých tříd jsou často definovány v rámci tříd nebo poblíž tvoření instance objektu.
3. Skryté závislosti definované statickém prostoru rozbíjejí modularitu.

Tyto problémy ohrožují udržitelnost aplikace při vývoji. Skryté závislosti definované ve statickém prostoru neumožňují v závislé třídě definovat její závislosti pouze pomocí rozhraní. Tudíž, jakákoliv změna ve statickém prostoru, může vést ke změně v závislé třídě, čímž porušujeme Single responsibility principle [3, s. 115], který říká, že každá třída by měla mít pouze jediný důvod ke změně. Dalším problémem jsou konfigurace nadefinované v rámci tříd nebo poblíž tvoření instance objektu, každá změna v konfiguraci pak vede ke změně zdrojového kódu třídy. Pro přizpůsobitelnou architekturu je důležité, aby změny v aplikaci nebyly invazivní. Pokud zvážíme výše uvedené fakty, tato podmínka nemůže být splněna.

V porovnání s nativní implementací persistentní aplikační kontejner nabízí tyto výhody:

1. Vede programátora rozdělovat aplikaci do nezávislých funkčních částí, jež mohou být jednoduše nahraditelné.
2. Tyto nezávislé části mohou být uloženy do paměti cache a znovupoužity dalšími požadavky.
3. Persistence objektů odstraňuje zbytečné inicializace v rámci běhu skriptů.

Za účelem ověření konceptu byl vytvořen testovací scénář, jenž zahrnoval inicializaci frameworku, kontroléru a vygenerování stránky s registračním formulářem. Testovaná aplikace je postavena na frameworku, jež uplatňoval principy DI a disponoval DI kontejnerem. Nicméně původní kontejner neumožňoval sdílet objekty mezi jednotlivými PHP skripty. Proto byl nahrazen prototypem persistentního aplikačního kontejneru. Tato úprava si však vyžádala menší úpravy ve startovací rutině frameworku. Z tohoto důvodu abychom předešli vlivu nutných změn, výsledky jsou porovnány mezi originální a modifikovanou verzí aplikace. Po uplatnění persistence na objekty došlo ke zrychlení aplikace přibližně o 17 %. V 98 % požadavků byla doba odezvy zkrácena, zbylé 2 % procenta měly dobu odezvy přibližně o 20 % delší vůči nepersistentnímu kontejneru. Tento vzor se opakoval i na dalších měřeních s různou úrovní konkurence souběžných požadavků. Je nutné mít však na paměti, že podíl změny v rychlosti, je vždy závislý na povaze aplikace.



Obrázek 2. Graf srovnávající doby odezvy aplikace při použití persistentního a nepersistentního kontejneru.

Tabulka 1. Podrobné statistky výsledků testu.

	Nepersistentní kontejner	**Persistentní kontejner**
Celkový čas	15.586 sekund	13.050 sekund
Dokončených požadavků	5 000	5 000
Chybných požadavků	55 (neplatná délka dokumentu)	0
Požadavků za sekundu	320.81	383.15
Průměrná doba vyřízení požadavku	3.117 milisekund	2.610 milisekund
Podíly požadavků dokončených v určitém časovém limitu (v milisekundách)	50 % 13 66 % 13 75 % 18 80 % 22 90 % 25 95 % 27 98 % 30 99 % 32 100 % 45 (nejdelší vyřízení)	50 % 11 66 % 11 75 % 15 80 % 18 90 % 22 95 % 23 98 % 25 99 % 27 100 % 56 (nejdelší vyřízení)

Závěr

Bylo vytvořeno řešení v podobě konceptu persistentního aplikačního kontejneru, které umožnuje globálně kešovat jednotlivé objekty v aplikaci, ale zároveň nezasahovat do zdrojových kódu tříd, čímž lze efektivně eliminovat zbytečné inicializace v rámci běhu jednotlivých PHP skriptů. Tímto způsobem zlepšovat výkon PHP aplikací. Test prokázal, že použitím prototypu persistentního kontejneru bylo dosaženo zrychlení testované aplikace přibližně o 17%. Důležitým faktem však zůstává, že samotné objekty v aplikaci musí splňovat požadavky na použití takzvaných persistentních objektů, tato podmínka byla v rámci testované aplikace splněna pouze částečně, problémem byly právě neserializovatelné objekty, jež bránily v plném využití potenciálu kontejneru. Tento problém je řešitelný, avšak by si vyžadoval úpravy v aplikaci.

Poděkování

Poděkování patří Jakubu Vránovi, českému přednímu vývojáři v PHP, za jeho konzultaci při řešení návrhu.

Literatura

[1.] FOWLER, Martin. Inversion of Control Containers and the Dependency Injection pattern. [online]. [cit. 2015-04-03]. Dostupné na World Wide Web: http://www.martinfowler.com/articles/injection.html.

[2.] POTENCIER, Fabien. Do you need a Dependency Injection Container?.*Fabien.potencier.org* [online]. 2009 [cit. 2015-05-24]. Dostupné na World Wide Web: http://fabien.potencier.org/article/12/do-you-need-a-dependency-injection-container.

[3.] MARTIN, Robert C a Micah MARTIN. *Agile principles, patterns, and practices in C#*. Upper Saddle River, NJ: Prentice Hall, c2007, xxxiii, 732 s. ISBN 978-013-1857-254.

Abstract

Contemporary applications generally become more complex, whereas flexible architecture is more beneficial than the amount of implemented functionality itself. This implicates a necessity of use containers responsible for management of dependencies in object-oriented solutions. The PHP platform hides an opportunity to implement a container which would persists in computer memory, whereby it could provide object sharing for ongoing requests, against the fact that PHP is a scripting language. A concept of the persistent application improves application performance, reusability of components and allows adaptively cache objects in PHP applications.