ООП в PHP: история развития и проблемы

Краткое повествование о том, как формировался PHP, как в нем появилось ООП и о том, какие проблемы в ООП PHP есть в настоящее время (PHP 5.2).

Если история развития ОПП в PHP вам неинтересна, можно сразу перейти к части «Проблемы в ООП PHP».

История развития PHP и ООП в PHP.

Когда-то давно, на заре развития Интернета вовсе не было ни PHP, ни «динамичных» веб-страниц. Сайты были похожи на простые книги, их суть заключалась в банальной передаче информации пользователю. Главным отличием сайтов той эпохи от книг был гипертекст. Возможность делать ссылки с одного документа на другой оказалась невероятно удобной и в то время потрясла мир.

Кстати. Для общего развития автор рекомендует прочитать статьи «Краткий экскурс в историю гипертекста» и «Секреты хорошего гипертекста».

Гипертекст определил сильный толчок в развитии Интернета, и буквально за несколько лет появилась необходимость в динамичных веб-страниц. Сайты начали развиваться в динамичном и интерактивном направлениях: появились поиски по каталогам, формы обратной связи, гостевые книги и т.д. Вместе с этим появилась необходимость в написании программ для сайтов.

Сервера в Интернете в то время повально работали на ОС Unix. Писать исполнимые бинарные файлы под эти операционные системы было сложно. На сайтах требовались абсолютно простые приложения, иногда в несколько строк. Выход нашелся быстро: решили использовать скриптовые языки программирования. Писать их было не сложно, а компилировать вовсе не нужно. Следовательно, использовать такие языки было на много проще.

Наиболее продвинутым и любимым языком среди системных администраторов Unix в то время был Perl. А так как подавляющее большинство серверов работало на Unix, его и стали широко использовать при программировании сайтов.

Многие программисты помнят нашумевший в свое время каталог *cgi-bin* на сайте. В него складывали все скрипты, и только на нем было разрешение на запуск (выполнение) программного кода (как скриптов, так и бинарников) в целях безопасности.

Проблема Perl заключалась в том, что изначально он разрабатывался как вспомогательное средство для системного администрирования. Во всяком случае – это была основная область его применения. Для программирования сайтов он был особенно удобен. Он ограничивал возможности программиста. Это привело к тому, что сообщество веб-программистов начало искать новый язык, более удобный для веб.

Таким образом, в 1994 году появился новый скриптовый язык программирования, созданный специально для генерации HTML-страниц на веб-сервере и работы с базами данных. Это был PHP. Во многом он был схож с Perl, синтаксис заимствовался из языка C++. Так как язык разрабатывался специально для веб и невероятно прост, он оказался очень удобным при программировании сайтов, и стал невероятно популярным. В конце 90-х – начале 00-х годов был бум развития PHP.

По сей день PHP – один из популярнейших скриптовых языков (наряду с JSP, Ruby и языками, используемыми в ASP.NET) благодаря своей простоте, скорости выполнения, богатой

функциональности и распространению исходных кодов на основе лицензии PHP. Сейчас для него разработано огромное количество расширений, библиотек, написано большое количество книг и статей.

Кстати, PHP 3.0 был официально выпущен в июне 1998 года после 9 месяцев публичного тестирования. Это была первая открытая официальная версия PHP. Многие задаются вопросом, куда делись PHP 1 и PHP 2? А никуда. Их просто не было. Были версии PHP/FI и PHP/FI 2.0. PHP 3 был переписан с нуля, поэтому прямой связи с PHP/FI не имел. Однако нумерацию версий решили продолжить с 3-й.

Проблемы в ООП PHP.

Проблемы ООП в PHP следуют из самых его корней. Так как в то время на сайтах использовались простые скрипты, состоящие из относительно небольшого числа строк кода, необходимости в мощном и полнофункциональном ООП для PHP не было. Разработчики написали довольно простой объектный движок, который поддерживал только основы ООП, и то с некоторыми искажениями, которые показались авторам PHP уместными.

Одним из самых очевидных промахов была передача объектов не по ссылке, а копированием. Запись *$a = $b* вела к созданию новой копии объекта, на который указывала *$a*. Эту ошибку исправили в PHP 5, но огромное количестве скриптов, написанных на PHP 4, содержат «костыли» для решения этой проблемы (повальное использование *&*).

В PHP 3 и PHP 4 не было возможности объявлять тип доступа к объектам класса. Это важный недостаток при разработке больших классов и больших систем. Это так же было исправлено в PHP 5, но не достаточно полно. Так как не был введен модификатор доступа при наследовании, остались большие проблемы с доступом к свойствам и методам класса.

Появившийся в PHP 5 деструктор, конечно, был полезен, однако большой погоды не сделал из-за отсутствия необходимости очистки мусора в PHP.

Введение в PHP 5 интерфейсов так же не сыграло большой роли из-за отсутствия жесткой типизации в PHP.

ООП в PHP постоянно дорабатывается и улучшается. В PHP 6 будут решены несколько проблем ООП, но он по-прежнему останется далек от совершенства. Вообще, основная задача PHP 6 – окончательно отказаться от поддержки устаревших стандартов. Это важный ход в истории PHP, но, на взгляд автора, работы в этом направлении ведутся не достаточно активно.

Perl давно перестал конкурировать и значительно отстал от PHP. Писать сайты на Perl в наше время могут только фанаты, энтузиасты и студенты в вузах. Объективных поводов программировать сайты на Perl нет. Однако с начала 00-х годов появилась и активно развивается другая конкурирующая с PHP технология – Microsoft .NET. В настоящее время она составляет очень серьезную конкуренцию PHP и большое число компаний (в т.ч. и многие гиганты) отказались от PHP и перешли на ASP.NET.

Основным языком в ASP.NET стал C#. Этот язык, по мнению автора, – одна из лучших разработок Microsoft. C-подобный синтаксис по-прежнему является одним из самых гибких и удобных. Важным преимуществом перед C++ является организация автоматической сборки мусора и простота написания кода.

Одно из главных преимуществ PHP – отсутствие жесткой типизации – стало одним из главных его недостатков в борьбе с C#. Вторым серьезным недостатком стало ООП. В C# ООП развито на уровне C++. Для крупных проектов этот недостаток PHP крайне серьезен.

Частью технологии .NET является мощный и удобный фреймворк – .NET Framework. В споре с ныне существующими фрэймворками PHP (CakePHP, Symfony, Zend Framework) .NET Framework однозначно выигрывает.

Кстати. Одно из главных достоинств ASP.NET тоже сыграло против него самого. Программирование сайтов на ASP.NET сделано похожим на программирование программ для Windows. В случае простых сайтов это действительно очень удобно. Разработка страниц ведется значительно быстрее, чем на PHP. Но для крупных сайтов такой подход, напротив, создает ряд неудобств. Поэтому, PHP пока держит серьезные позиции и сдавать их не собирается.

Теперь вернемся к ООП в PHP. Так как текущая версия PHP 5, о ней и будем говорить.

Проблема 1. Одна из серьезных проблем ООП в PHP 5 уже была озвучена. Это отсутствие модификатора наследования. Об этом подробнее.

Допустим, создан класс, в котором часть методов объявлена как *public*. При создании нового класса, базовым для которого служит первый созданный класс, эти методы нужно скрыть. Это нужно, когда интерфейс класса-наследника «перекрывает» функционал открытых методов базового класса. Естественно, что при использовании базового класса отдельно возникает необходимость оставить эти методы открытыми, а при использовании его в качестве базового для нового класса, методы должны быть закрыты. В PHP этого механизма нет.

Проблема 2. Отсутствие возможности множественного наследования. Это свойственна не только PHP, но и C# (и некоторым другим языкам). Присутствует оно в C++. От множественного наследования отказались потому, что оно сложно и якобы «непонятно» для программиста. По мнению автора это глупо. Это все равно, что отказаться от полетов в космос, потому что это сложно.

В некоторых ситуациях возникает острая необходимость множественного наследования. Например, когда есть некий условный класс получения (getter) некого X и класс установки (setter) этого же самого X. На более высоком уровне может иметь смысл объединить их в один класс работы с X. Для этого надо написать класс, базовыми для которого будут сразу два класса – getter и setter. Но сделать этого технически мы не можем.

Проблема 3. Конструктор базового класса не вызывается по умолчанию из конструктора наследника. Но если конструктор наследника вовсе не определен, конструктор базового класса все же будет вызван. Это вносит некоторую путаницу и чревато ошибками.

Проблема 4. Следствием отсутствия строгой типизации является отсутствие возможности перегрузки функций. Да, в случае с обычными функциями или методами класса это легко обойти и серьезных затруднений не вызывает. Проблема возникает при необходимости перегрузки конструктора. Тогда приходится выкручиваться и делать «костыли». Это неприятно.

Проблема 5. В PHP 5 есть возможность указать тип аргумента функции (только для классов и массивов). Если переданный аргумент не будет соответствовать требуемому типу, возникнет ошибка и функция вызвана не будет. Это удобно: избавляет от необходимости повально писать лишние проверки.

Но разработчики забыли один важный нюанс, который проявляется серьезными неудобствами в некоторых случаях. Если указан тип объекта, нельзя передать в качестве объекта *NULL*. *NULL* – это по сути обозначение отсутствия объекта. В C++ *NULL* повально используется в местах передачи объекта в функции. В PHP программисты лишены такой возможности. В результате возник казус. Если написать *function f(MyClass $class = null)*, то такая конструкция будет замечательно работать. Но вызов *f(null)* выдаст ошибку. В результате, при наследовании, когда из метода, где есть параметры-объекты со значением по умолчанию *NULL*, вызывается метод класса-родителя с теми же параметрами-объектами со значением по умолчанию *NULL*, программист вынужден писать костыли в виде блоков *IF*.По мнению автора, такая недоработка лишает метод указания типа параметра половины преимуществ.

Кстати, эта проблема возникла из-за отсутствия строгой типизации в PHP. Из плюса в начале развития PHP сейчас этот вопрос перерос в минус.

Проблема 6. Отсутствие нормального определения виртуальных свойств и методов класса (свойств, которые имеют геттеры и сеттеры).

Кстати. По сути, невиртуальные свойства – это вовсе не «свойства» класса, а «поля». Свойством же, как раз, называется поле, определенное не в виде переменной, но в виде геттера и/или сеттера. Это в языках с нормальной реализацией ООП. В PHP любое поле класса называется свойством. А виртуальные свойства входят в понятие перегрузки свойств.

Из-за неудобного способа перегрузки свойств и методов класса, интерпретатор не может точно определить, существует ли свойство или метод в классе, пока непосредственно его не вызовет. Как следствие – отсутствие виртуальных (перегруженных) свойств и методов в выпадающем списке членов класса в PHP-редакторах и IDE. Именно по это причине автор избегает перегрузки во всех случаях, когда это возможно. Использование методов перегрузки (__get, __set, __call) – отличный способ запутать программиста, использующего ваш класс.

Создание перегрузки свойств и методов в таком виде – это ужасный поступок со стороны разработчиков PHP. Ведь нельзя заставлять программиста помнить все методы и свойства класса наизусть. Это просто невозможно. Именно для этого во всех современных средах программирования (IDE) используются выпадающие списки членов класса. Использование методов перегрузки ломает механизм выпадающих списков.

Проблема 7. Открытость классов. Даже защищенные и приватные свойства класса в runtime легко просмотреть и подменить, используя сериализацию. Это открытый путь ко взлому скриптов.

Проблема 8. Смешение статических и нестатических методов и свойств. Уже сейчас статические методы и свойства можно вызвать точно так же, как нестатические. Это может внести путаницу и привести к ошибкам, так как программист не сможет отличить статическое свойство от нестатического, не подсматривая в класс.

В PHP 5.3 собираются расширить возможности запутывания статических и нестатических свойств и методов. Это просто глупо.

Заключение.

ООП в PHP по-прежнему нуждается в серьезной переработке. И коррективы, которые будут внесены в PHP 6, не спасут ситуацию.

PHP можно считать удачным экспериментом по созданию языка с нежесткой типизацией переменных. И на его примере видно, что это плохой путь. При создании простых скриптов такой подход удобен, но чреват ошибками. При создании крупных систем такой подход оказывается и

неудобным, и влекущим множество мелких ошибок. Кроме этого, нестрогая типизация вносит проблемы и в развитие ООП языка.

Автор уверен, что наследником PHP в недалеком будущем (2012-2015 года) станет язык со строгой типизацией. Возможно, это будет даже C#.

P.S. При перепечатке статьи обязательно указание этого абзаца в полном объеме и со ссылками. Автор статьи – Валерий Леонтьев aka **feedbee**. Источник – Персональный блог автора.