Урок 8

Развитие карьеры разработчика. Инструменты и необходимые знания

Обзор тенденций современного программирования и разработки на PHP. Знакомство с полезными инструментами. Обсуждение дальнейшего развития.

Очевидно, что два десятка занятий по основам и инструментам языка не могут дать полную картину и статус профессионального разработчика. Но это необходимый фундамент, на котором будет базироваться дальнейшее изучение языка и смежных систем. О том, что это за знания и инструменты, пойдет речь на этом уроке.

Fullstack-разработка

Мало работодателей на рынке веб-разработки заинтересованы в узких специалистах. Чтобы производить качественный, продаваемый продукт, разработчик не должен ограничивать свою область деятельности написанием скриптов – нужно понимать и работу смежных систем. Во многих компаниях позиция разработчика PHP уже сменилась позицией fullstack-разработчика. Что она в себя включает?

Современному разработчику надо знать язык **SQL**. Уметь составить запрос не только выборки всех полей, но с объединением таблиц, с фильтрами и т.п. Это требование уже к уровню **junior**.

Надо иметь навыки работы с **HTML** и **CSS**. Это не значит, что нужно будет верстать, но ваша ценность как специалиста значительно возрастает, если вы знакомы с этими технологиями, а еще лучше – с новшествами **HTML5** и **CSS3**.

Отдельно стоит упомянуть и frontend-разработку. Современный веб-сайт невозможен без **JavaScript** и асинхронных действий. Вакансии с чистым PHP – редкость, в особенности на уровнях junior и middle. Как правило, требуется разработчик, который умеет верстать и кодировать. Но программистов редко просят переносить макет с картинки в код – этим, как правило, занимаются верстальщики.

Имея представления о **PHP** и **JS**, вы со временем сможете переквалифицироваться в разработчика мобильных приложений – и это еще поднимет вашу «стоимость» на рынке труда.

Операционные системы

Вне зависимости от того, с какой операционной системой вы привыкли работать, в 95% случаев ваша программа будет работать под ОС семейства **Linux**. Веб-сервера на ОС Windows уже сложно встретить, и практически невозможно – на MacOS.

Необходимо изучать команды **Linux**, разбираться в принципах работы ОС хотя бы в общих чертах: вы же будете настраивать свою программу для работы на сервере.

Многие компании строят департамент IT по методологии **DevOps**. Это методология разработки программного обеспечения, нацеленная на активное взаимодействие и интеграцию специалистов по разработке и по информационно-технологическому обслуживанию. Базируется это на идее о тесной взаимозависимости разработки и эксплуатации программного обеспечения. Цель – помогать организациям быстрее создавать и обновлять программные продукты и сервисы. Задача **DevOps** – сделать процесс разработки и поставки программного обеспечения согласованным с эксплуатацией. Часто эти задачи решаются при поддержке автоматических средств.

Базы данных

Знание **SQL** и **MySQL** будет жизненно необходимо. Но может случиться и так, что у вас будет другая база семейства **SQL**: **PostgreeSQL** или **Oracle**. Синтаксисом запросов они не отличаются от **MySQL**, но каждая из них реализует различные механизмы хранения и индексации данных.

Чтобы стать настоящим специалистом в данной области, стоит изучить эти технологии и понять, какой движок БД лучше использовать в каждой конкретной ситуации. На уроках мы разбирали самые

простые принципы построения запросов, но **SQL** – это огромный стек технологий, который невозможно рассмотреть в рамках даже целого курса.

Помимо баз данных **SQL** существуют и другие, **NoSQL**-хранилища. Они далеко не всегда выступают альтернативой SQL, но могут взаимодополнять друг друга в рамках одного проекта.

Самая популярная среди баз данных NoSQL – MongoDB. В объектно-ориентированной базе данных набором данных является не строка, состоящая из ячеек, а целый документ со своим набором свойств. Даже в рамках одной условной «таблицы» могут находиться данные разного формата и объема. Существенно усложняется выборка таких данных, но в качестве плюса имеем удобное хранилище для «нестандартных» объектов. Например, товар в магазине может иметь индивидуальный набор самых разных свойств. Подобный механизм можно реализовать и через MySQL, но придется строить несколько JOIN-запросов, что при большом объеме данных сильно повлияет на скорость выборки. С Mongo эту проблему решить гораздо проще.

К другим NoSQL-решениям можно отнести **key-value хранилища**. Это базы данных, которые представляют собой таблицу, где хранится ключ и его значение. Любое значение можно получить по ключу. При этом значением может быть не только строка, но и число, массив, и даже целый объект.

Подобные хранилища чаще всего используются как вспомогательный кэш: когда надо быстро передать небольшой объем данных между двумя процессами. В одном случае мы можем воспользоваться уже привычным SQL и проиграть в производительности, а в другом – сохранить данные в оперативной памяти.

В качестве примера разберем такую задачу: на сайте запускается специальный скрипт, который откуда-то вытягивает каталог товаров. Размер каталога нам неизвестен – только его структура. Извлекать данные необходимо раз в минуту. Важно избежать ситуации, когда импорт каталога еще не завершен, а уже запускается следующий процесс импорта.

Чтобы такая ситуация не возникла, мы при запуске импорта должны проверить, запущен ли сейчас еще один процесс. Самый «дорогой» путь – сохранить какой-то ключик в MySQL и по завершении работы программы его удалить его. Другим «дорогим» способом является создание файла-маркера. Есть вероятность, что сервер будет перезагружен во время исполнения и ключ не удалится, и программа больше не запустится, т. к. будет считать, что она еще выполняется.

Если сохранить такой маркер в оперативной памяти, мы можем быть уверены, что контролируем его и он пропадет из памяти в случае перезагрузки.

Для решения данной задачи можно воспользоваться двумя наиболее популярными **key-value** базами данных, работающих с оперативной памятью: **MemoryCache** и **Redis**.

MemoryCache – достаточно простое решение, уже имеющее модуль для работы с **PHP**. Эта база позволяет хранить какой-либо объект в оперативной памяти в течение ограниченного времени. Но у этой базы есть существенные ограничения: если необходимо хранить в памяти большие объекты, нужно разбивать их на несколько фрагментов и собирать по необходимости.

Еще одно решение – использовать **Redis**. Принцип работы тот же, что и у **MemoryCache**, но эта база позволяет выполнять поиск непосредственно внутри хранящихся в памяти объектов. Она может сохранять редко используемые объекты на устройствах ПЗУ и «прогревать» кэш, т.е. готовить его к активному взаимодействию, ускоряя работу приложения после простоя.

MemoryCache. Практика.

Решим нашу задачу с помощью **MemoryCache**. Проверьте, установлено ли у вас расширение **php-memcache** на веб-сервере. Сервер **memcache** по умолчанию работает на порте **11211**. Получить

и загрузить в него данные можно через протокол **telnet**, а можно и с помощью php-модуля, чем мы и воспользуемся.

Создадим класс **InstantCache**, который будет оберткой вокруг стандартных методов **memcache**. Это нужно для:

- установки префиксов для ключей, чтобы объекты других проектов не пересекались с нашими;
- доступа к объекту **memcache** из любого участка кода.

```
<?php
class InstantCache
{
   const PREFIX = 'cyberstars_';
   protected static $instance = null;
   protected static function Connect()
   {
    if (self::$instance) return false;
    $instance = new \MemCache();
    $instance->connect('localhost', 11211);
    self::$instance = $instance;
   }
   public static function get($key)
   {
    if (!self::$instance) {
     self::Connect();
    }
    $key = self::PREFIX . $key;
    return self::$instance->get($key);
}
public static function set($key, $value, $time = 60)
{
   if (!self::$instance) {
    self::Connect();
   }
   $key = self::PREFIX . $key;
   return self::$instance->set($key, $value, false, $time);
}
public static function remove($key)
{
   if (!self::$instance) {
     self::Connect();
   }
   $key = self::PREFIX . $key;

   return self::$instance->delete($key);
 }
}
?>
```

Будем пользоваться тремя методами: **get**, **set** и **remove**. Напишем базовый класс команды (им можно будет заметить существующий):

```
Abstract class BaseCommand
{
   protected $locker = '';
   public function handle()
   {
      $this->unlock();
      if (!$this->isLocked()) {
         $this->lock();
      $this->exec();
     $this->unlock();
   }
}
protected function exec()
{
}
protected function lock($time = 3600)
{
   if (!$this->isLocked()) {
     InstantCache::set($this->locker, true, $time);
   }
}
protected function unlock()
{
   if ($this->isLocked())
    {
     InstantCache::remove($this->locker);
    }
}
protected function isLocked()
{
   return InstantCache::get($this->locker) ? true : false;
}
}
?>
```

Разберем логику метода **handle**. Проверяем наличие в **memcache** ключа **$this->locker** (его нужно будет заполнять обязательно). Если он существует, прекращаем выполнение программы. Если нет – добавляем этот ключ в оперативную память, выполняем саму команду, и по ее завершении удаляем ключ из памяти. Нужно будет только унаследовать этот класс в команде, задав **$this->locker** и перегрузив метод **exec()** своей логикой.

Таким образом можно защититься от срабатывания команды, если ее работа не была завершена. Теперь можно добавить наш скрипт в **cron** (или другой планировщик задач).

Менеджеры очередей

Зачастую на крупных проектах применяется такое решение, как менеджеры очередей. Примером такого менеджера может быть **RabbitMQ**. Менеджер очередей – это интерфейс между различными компонентами приложения. Его задача состоит в том, чтобы принять какие-то данные (например, сообщение из чата) и отправить ответ, что пакет доставлен и обрабатывается. После того, как другой компонент забрал эти данные, обработал и передал снова менеджеру очередей, последний может сообщить первому отправителю, что данные обработаны.

Зачем это нужно? Когда работа идет с огромным комплексом задач, время ожидания ответа может быть долгим. Например, пользователь хочет получить банковскую выписку за год. Чтобы не показывать ему «крутилку» ожидания, можно сообщить, что запрос принят, но выписка будет сформирована позже.

На сервере эта команда встанет в очередь и будет выполнена с требуемыми затратами времени. После этого пользователю отправится ответ в виде email, личного сообщения или push-уведомления. Пользователю не всегда принципиально важно получить ответ от сервера в момент запроса. Но ответ должен доходить до него и не неделю (хотя и такое бывает!). Самое важное – сообщить пользователю, что данные обрабатываются и не реализовывать постоянный опрос сервера (который под большой нагрузкой может «тупить»).

Фреймворки

Каждый раз, когда начинается разработка нового проекта, вы пишете одно и то же: ядро, которые принимает запросы и отдает контент. После этого, в зависимости от требований проекта, вы дополняете это ядро различными функциями.

Чтобы не выполнять одни и те же заученные действия, есть смысл воспользоваться готовым решением – PHP-фреймворком. В рамках курса мы начали с того, что подготовили основу нашего сайта – небольшой фреймворк, хоть и не успели сделать его его по-настоящему гибким.

Фреймворки избавляют от самого интересного для начинающего программиста – написания ядра, и кидают в рутину: проектирование базы данных, моделей и контроллеров. Но это дает возможность быстро запустить проект практически любой сложности.

Преимущества фреймворков

- Для популярных решений много документации;
- ПО на популярных фреймворках дешевле поддерживать за счет большего числа специалистов, работающих с ними;
- Программисту не надо думать, что происходит в ядре – все задокументировано.

Разумеется, на первых порах можно и нужно писать свои «велосипеды» – только так можно научиться находить правильные решения для своих задач и эффективно использовать инструменты. Но чтобы быть в тренде, просто необходимо хотя бы бегло ознакомиться с тем, как устроены наиболее популярные современные фреймворки: **Symfony**, **YII** и **Laravel**.

Каждый из фреймворков обладает своими плюсами и минусами. Например, **Symfony** является наиболее мощным, но при этом крайне громоздким решением с высоким порогом вхождения, в то время как **Yii**, имея порог вхождения наравне с «голым» PHP, наделен более простым функционалом.

Контроль версий

Для работы в команде наличие системы контроля версий кода является уже недостаточным, но необходимым условием. Многие крупные компании используют в разработке Git.

Git – распределенная система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux.

Любая система контроля версий позволяет:

- хранить несколько версий одного и того же документа;
- при необходимости возвращаться к более ранним версиям;
- определять, кто и когда сделал то или иное изменение;
- и многое другое.

Часто бывает, что над одним проектом одновременно работают несколько человек. Если два человека изменяют один и тот же файл, один из них может случайно отменить изменения, сделанные другим. Системы управления версиями отслеживают такие конфликты и предлагают средства их решения. Большинство систем может автоматически объединить (слить) изменения, сделанные разными разработчиками. Однако такое автоматическое объединение изменений обычно возможно только для текстовых файлов и при условии, что изменялись непересекающиеся части этого файла. Такое ограничение связано с тем, что большинство систем управления версиями ориентированы на поддержку процесса разработки программного обеспечения, а исходные коды программ хранятся в текстовых файлах. Если автоматическое объединение выполнить не удалось, система может предложить решить проблему вручную.

Менеджеры зависимостей

Фреймворк – это не просто ядро, которое выполняет базовые функции. Он может содержать множество дополнительных модулей: работу с почтой, со специфическими базами данных, другие утилиты. Чтобы не тащить эти компоненты в git (некоторые могут весить десятки мегабайт), но иметь возможность развернуть такое же окружение на другой системе, можно воспользоваться пакетными менеджерами. В конфигурации проекта можно указать, какие именно версии компонентов необходимо доставить на сервер, и установить их или обновить с помощью одной команды.

Но следует соблюдать осторожность: есть вероятность, что пакет, от которого зависит ваш проект, может быть удален из публичного репозитория.

Карьера

После обучения закономерно начать зарабатывать деньги, используя полученные знания. Но начинающему разработчику сделать это без портфолио проблематично.

Получив основные сведения по инструментам, рекомендуется потренироваться с ними. Для начала – доделайте домашние задания.

Придумайте свой проект. Пусть он не будет уникальным, но постарайтесь использовать в нем максимум из того, что вы узнали. Напишите, например, блог или чат.

Работодатель

На самом деле, работодателю не очень интересно, что вы знаете. Он искренне заинтересован только в одном: можете ли вы решить его задачи и сколько это будет стоить. Когда компании требуется программист, им нужен не человек, который умеет писать программы, а человек, который умеет решать задачи с помощью написания программ. Чем дешевле будет ваше решение (причем во всех аспектах: разработка и поддержка) – тем больше выгоды для компаний.

Поэтому именно ваш технологический кругозор позволит вам решать задачи работодателя, затратив минимум его (и своих) ресурсов. Не пишите код ради того, чтобы просто попрограммировать – от вас этого не ждут. Если вы можете решить задачу с помощью простого shell-скрипта, а не громоздкого решения – это ваше преимущество. Если ваше решение можете адекватно поддерживать только вы – это большой минус.

Пригодится умение пользоваться поиском: знать все невозможно, а уметь быстро найти информацию – вполне.

Собеседование

Работодатель принимает решение о приглашении вас на интервью, исходя из вашего соответствия требованиям должности (количество страшных аббревиатур в резюме). Будьте готовы к тому, что у вас поинтересуются, как именно вы использовали ту или иную технологию.

Обязательно будут вопросы по инструментам. Если уж мы говорим о языке PHP – будьте готовы ответить на самые простые: основные принципы ООП, функции PHP, синтаксис.

Такие вопросы могут показаться глупыми, но именно ответы на них помогают работодателю понять, насколько глубоко кандидат знает свою область.

На начальных позициях от вас не будут требовать глубоких знаний, но будет очень интересен ход ваших мыслей. Поэтому, даже если не знаете точного ответа, поразмышляйте вслух. Обсудите ваше решение.

Если ваши ответы удовлетворят интервьюера, вам наверняка предложат тестовое задание. Как правило, нужно будет выполнить его в рамках собеседования, но бывает и так, что дают на дом. Даже если кто-то из ваших знакомых поможет вам его выполнить, вас обязательно спросят, почему вы решили выполнить его именно так.

В хороших компаниях не жалеют ресурсов на покрытие кода тестами. Лучше сделайте это, чтобы продемонстрировать, что вы понимаете, зачем это нужно. Оформляйте код в соответствии со стандартами, оставляйте комментарии в коде. Обязательно проверьте ваше решение на всех возможных вариантах, предусмотрите, чтобы приложение корректно реагировало на неадекватные входные данные.

Первые дни на новой работе

Новая работа – это всегда большой стресс, а новая работа в новой сфере – вдвойне. Готовьте себя к тому, что вас нагрузят новыми подходами к разработке, вам нужно будет быстро «въехать» в код нового проекта. К сожалению, не все компании придерживаются грамотной организации хода

разработки: проектирование – разработка – тестирование – продакшн. Кто-то допускает отсутствие хорошей аналитики, кому-то не нужны тесты, а кто-то даже допускает корректировки «боевого» кода. Впрочем, это тоже хороший опыт!

Итоги

Вы закончили курс «PHP. Профессиональный уровень». Мы постарались собрать в нем самый актуальный материал и донести до вас самое основное, что необходимо вам как веб-разработчику.

Методики и ПО постоянно меняются, поэтому в любом случае необходимо поддерживать актуальность знаний самостоятельно, не забывая подкреплять их практикой, будь то собственный проект или коммерческая разработка.