1

INTRODUCTION

- Choosing the right programming language
- Clarifying terminology
- Introducing Helpful Robot
- Getting up and running

CHOOSING THE RIGHT PROGRAMMING LANGUAGE

You're going to learn many different programming languages throughout your career. Since you're reading a book about PHP, you've probably already tried to build a website. And, you've probably encountered HTML, CSS, and JavaScript along the way.

As developers, we often get to choose which programming languages we'll use to solve our problems. Sometimes our choices depend on what we're most comfortable with. Sometimes our choices depend on what the programming languages we're deciding between, are good at doing.

Nowhere are these decisions more apparent than the subject of asynchronous PHP programming. Let's consider an example. Imagine your boss asks you to build a real-time news feed, to cover a live event. Your boss doesn't care which programming language you use, so long as thousands of visitors can receive instant updates, without refreshing their browsers.

How would you build such a thing? Perhaps you'd head over to a forum of programmers, and ask which programming language you should use. That's not a bad idea. They're likely to tell you which programming languages they think will be better suited to the task. You'll have to figure out how much of their choice depends on the strengths of the language and how much of it depends on their comfort with the language.

These days, a common answer is to use something like NodeJS. It's a JavaScript platform, which is event-driven and has a massive community. There are some event-driven design patterns which have a mature footing within the language, and many people have developed libraries in these design patterns.

That's not a bad thing. It's your professional duty to pick the right tool for the job. But every tool has its trade-offs. If you already have a team of PHP developers or an existing PHP codebase, adding a new JavaScript platform to the mix may be costly. The developers will need time to adjust to the new programming languages and environment. You'll need time to re-write the parts of your code to interact with the JavaScript code you're about to write.

How about building the news feed in PHP? You may ask this of the forum members, and would likely be surprised by the answers you hear. PHP is not known for its ability to multitask. It rose to popularity through a systematic approach of embedding itself inside popular web servers. It was designed to be an easy replacement for C and Perl, in a time when web development was the domain of very few developers.

Traditional PHP scripts thrive on the implicit flow of HTTP requests and responses. They're meant to be short-lived and to dump every variable and resource the moment the response is sent to a browser. They're not designed for persistent connections, or to act as their own web server.

That's not to say PHP can't be used to do any of these things, but popular tradition has molded the community into a pattern of thinking that has been hard for me to shake.

CHOOSING THE RIGHT PROGRAMMING LANGUAGE

I have a simple goal. I want to help a new generation of PHP developers discover PHP can do just as much as platforms like NodeJS. We need to cast off the old ways of thinking and learn how to build simple, elegant, asynchronous PHP applications.

There are a few reasons why this is a good idea. As we'll see, there are practical reasons, like allowing more people to use our sites and services at the same time. These gains save on the cost of new hardware and make full use of the hardware which we already have.

There are also long-term reasons for why we might want to build these kinds of applications. The language already includes basic tools which allow for this kind of architecture. We're going to see what those tools look like, but it's important to realize they're still very basic.

We can choose to ignore parts of PHP that allow asynchronous architecture, or we can understand how they work. Many people have done the latter, building libraries and extensions to further compliment this toolset.

If you needed a database and had to install an extension to use it, that wouldn't be a difficult choice. If you needed higher concurrency and wanted a faster request-response cycle, wouldn't it be just as simple to install an extension or use a built-in function?

CLARIFYING TERMINOLOGY

There are a few terms we need to understand before we dive into code. The first is that traditional PHP programming is **synchronous**. That is, each program is essentially a list of instructions, executed one after the other.

This is the simplest and, depending on how to look at it, the slowest way of building applications. It's simple because the order is predictable. Once you understand what each instruction does, you know exactly how the script will execute.

But there's another way of building applications that environments like NodeJS have made popular. It's called reactive, event-based, or **asynchronous** programming. Scripts are still written as a list of instructions, but the order in which they are executed is no longer as predictable.

Asynchronous programming adds to this list of instructions, the idea of events. A database query is no longer just an expected execution time of `120ms`, but when complete it can trigger an event. Further instructions can be executed and completed in the time it takes a database query to complete because they happen in a kind of cooperative-processing execution state.

It's a kind of cooperative-processing because single-threaded, single-process environments can't perform multiple tasks at the same time. It's still more efficient since synchronous processing forces the processor to wait until the completion of things like network requests and file system operations, before returning to the real purpose of the script.

Asynchronous execution works around these waiting periods since the processor is allowed to do work while the "waiting operations" are periodically polled for updates. We've seen this time and time about, in the browser. We could, for instance, make all Ajax requests synchronous. But the browser wouldn't be able to respond to user interaction or even scroll while waiting for the network request to complete.

Instead, we can use event-based JS to *tell us* when the network request is finished so we can respond accordingly. And, in the meantime, we'd have been able to do other things with the processor.

CLARIFYING TERMINOLOGY

To do many things at the same time, we need a multi-threaded or multi-process environment. This is called **parallel execution**, because multiple tasks can be run in a truly parallel state.

Not only is this possible using built-in PHP functions, but it's also made easy using libraries and extensions. If you needed to communicate with an IMAP server, you wouldn't build the socket client library from scratch, would you? It's far easier to install an extension for that. The same can be said of databases, message queues, cache servers, etc.

In the same way, we can install Pthreads, PCNTL, and Amp. These extensions and libraries smooth over the differences in server hardware and software, to the point where we can use multi-threading and multi-process scripts.

> If you don't know what these are, don't worry. There are many chapters dedicated to the individual libraries and extensions mentioned. I'm not even linking to them here because learning exactly what they are, or how they differ, is just a distraction at this point.

There are some immediate benefits, as already explained through synchronous vs. asynchronous Ajax requests. Being able to do more things while "waiting" lets us serve more HTTP requests and perform more "business logic" than before. A document conversion service can convert more documents; a machine learning API can analyze more data.

It also lets us move slow processes out of places where the biggest concern is how responsive the application is to users. Why write to a database or send an email, forcing the user to wait for a browser response? We could just as well push these expensive tasks to another thread, process, or server.

NodeJS, Ruby, Python, and Go all have these abstractions built-in. PHP may not initially have been designed as a general purpose language, but that's how we're using it. PHP may not have been designed with multi-threading and multi-process built in, but that's how we can use it.

INTRODUCING HELPFUL ROBOT

A few years ago, I started to work on a few SilverStripe framework modules. I was employed by SilverStripe and noticed something interesting. There were about 2,000 modules tagged specifically to work with the framework, and though many of them did interesting things, very few of them were what some consider to be a high standard.

There were some obvious omissions, like missing tests and documentation. Then, there were less obvious (or agreed upon) things missing; such as PSR conformance, licenses, and codes of conduct.

One day I started working on a collection of scripts to inspect these modules and suggest changes. I started with a script which would format the source code to conform to PSR-2. It would store alongside the module's name and path, whether or not the module conformed to PSR-2. It would also get a pull request ready, and provide me with a link so I could review and submit pull requests as needed.

As time went by, I added more and more scripts to this helper application.

> I wasn't asking people to opt into Helpful Robot's suggestions. I figured, since I was manually reviewing every pull request before submitting it, I was just being a very active community contributor. The GitHub team and terms of service agreed with me.

There were many of these pull requests floating around, and module maintainers frequently maintained multiple SilverStripe modules. I didn't want to associate my real name with a large number of pull requests (in case module maintainers became irritated with new activity on dormant modules). Thus, I created a new Github account and named it Helpful Robot.

Over the space of six months, Helpful Robot submitted thousands of pull requests. The percentage of merged pull requests is currently around 46%. That's a fantastic number of contributions to community modules, and something I'm very proud of.

Despite the effort, Helpful Robot has always remained a relatively small collection of scripts. I've rebuilt the scripts a few times, but I've longed to make it into something more.

In this book, we're going to remake Helpful Robot into a continuous integration service to be reckoned with. It will retain its core inspection and suggestion functionality, but this time it'll have a face. A charming, robotic face.

GETTING UP AND RUNNING

Nothing ages a book faster than a list of installation instructions for every operating system the reader could possibly have. I'm not going to do that here.

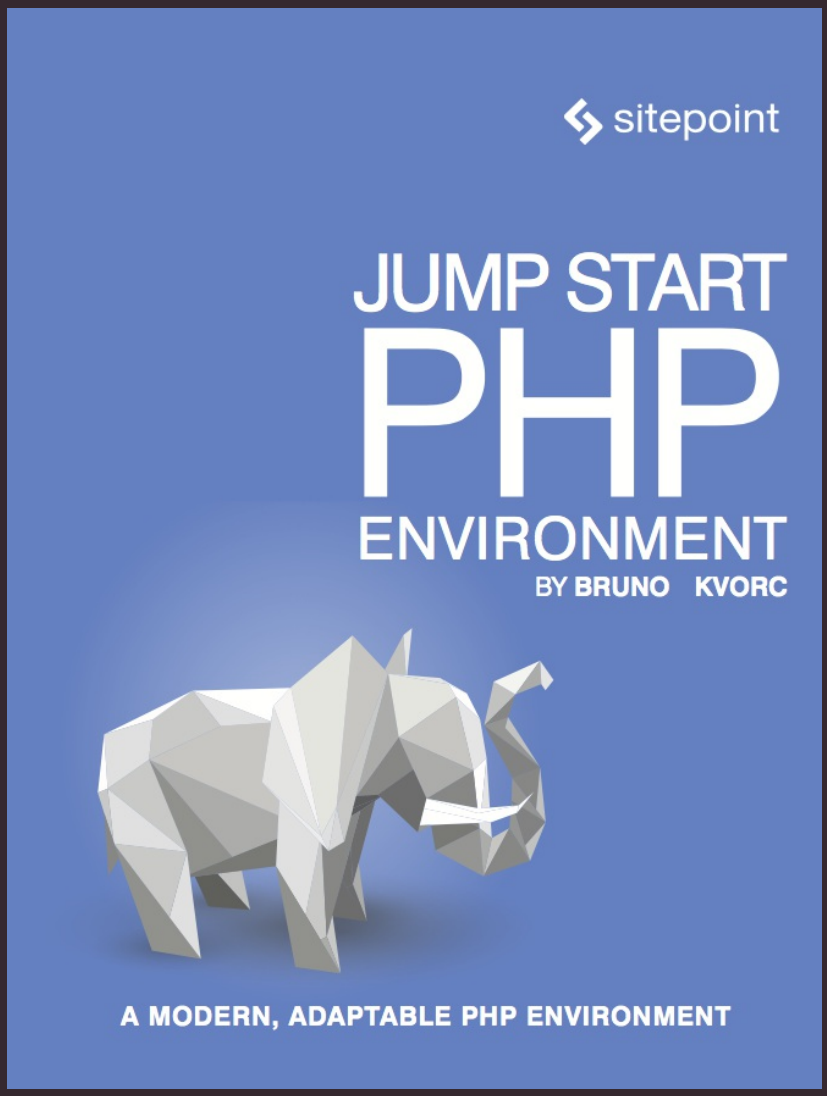
> I'm also assuming, since you're reading this book, you have some experience in PHP. You probably already have a working development environment.

There are a couple of books you could refer to, for specific setup information. The first is written by this book's technical reviewer: Bruno Škvorc, Jump Start PHP Environment. It covers many topics, from Integrated Developer Environments to Vagrant to Git.

Bruno's book does a great job of explaining the basics and getting you started with a working development environment. If you'd like details about the finer aspects of Vagrant configuration, you should also definitely check out Erika Heidi's Vagrant Cookbook.

Later on, we'll begin handling HTTP requests and making database queries. We'll deploy Helpful Robot to a server, so others can access its API and interface. For this, we'll need a virtual private server, preferably one geared to support asynchronous architecture.

I've recently found Platform.sh to be a good fit for asynchronous PHP applications. We'll see why in later chapters. In the meantime, feel free to create an account there. They offer free trials and have good documentation over at docs.platform.sh.


sitepoint
JUMP START
PHP
ENVIRONMENT
BY BRUNO KVORC
A MODERN, ADAPTABLE PHP ENVIRONMENT