Event-Driven Architecture - Part 1

In this lesson, which is part one of the event-driven architecture section, we will cover concepts like blocking and non-blocking.

We'll cover the following

- What is blocking?
- What is non-blocking?

When writing modern Web 2.0 applications, chances are you have come across terms like *reactive programming* and *event-driven architecture* and concepts like *blocking* & *non-blocking*.

What are they? Should I be aware of them?

You might have also noticed that tech like *NodeJS*, *Play*, *Tornado*, and *Akka.io (http://Akka.io)* are gaining more popularity in the developer circles for modern application development in comparison to traditional tech.

Why is that the reason for that? Is it just that we are bored of working on the traditional tech like *Java*, *PHP*, etc. and are attracted towards the shiny new stuff, or are there technical reasons behind this?

In this lesson, you will go through each and every concept step by step to realize the demands of modern software application development.

So, without any further ado, let's get on with it.

Alright, at this point in the course, we know what *persistent connections* are, what *asynchronous behavior* is, and why do we need it. We can't

really write real-time apps without implementing them.

Let's start with *blocking*. What is it?

What is blocking?#

In web applications *blocking* means the flow of execution is stopped while waiting for a process to complete. Until the process completes it cannot move on. Let's say we have a block of code of ten lines within a function and every line triggers another external function executing a specific task.

Naturally, when the flow of execution enters the main function it will start executing the code from the top, from the first line. It will run the first line of code and will call the external function.

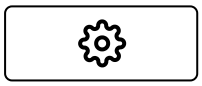
At this point in time until the external function returns the response, the flow is blocked. The flow won't move further. It just waits for the response, unless we add *asynchronous behavior* to it by annotating it and moving the task to a separate thread. However, that's not what happens in the regular scenario, like in regular *CRUD-based* apps. Right?

So, this behavior is known as *blocking* because the flow of execution is blocked.

What is non-blocking?#

Now, we have come to the *non-blocking* approach. In this approach, the flow doesn't wait for the first function that is called to return the response. It just moves on to execute the next lines of code. This approach is a little not-so-consistent as opposed to the blocking approach because a function might not return anything or throw an error. Still, the code in

the sequence up next is executed.



The *non-blocking* approach facilitates *Input-Output (IO)* intensive operations. Besides the disk and other hardware-based operations, communication and network operations also fall under *IO* operations.

We will continue this discussion in part two of the event-driven architecture lesson. You will have an insight into what events are what event-driven architecture is, and the technologies used to implement it.

← Back

Stream Processing Quiz

Next →

Event-Driven Architecture - Part 2

Mark as Completed

Report an Issue