Rust

- Futures and Tokio

Futures and Tokio

> This article was originally published at https://gist.github.com/joepie91/bc2d29fab43b63d16f59e1bd20fd7b6e. It may be out of date.

Event loops

If you're not familiar with the concept of an 'event loop' yet, watch this video first. While this video is about the event loop in JavaScript, most of the concepts apply to event loops in general, and watching it will help you understand Tokio and Futures better as well.

Concepts

- **Futures:** Think of a `Future` like an asynchronous `Result`; it represents some sort of result (a value or an error) that will eventually exist, but doesn't yet. It has many of the same combinators as a `Result` does, the difference being that they are executed at a *later* point in time, not immediately. Aside from representing a future result, a `Future` also contains the logic that is necessary to obtain it. A `Future` will 'complete' (either successfully or with an error) precisely once.
- **Streams:** Think of a `Stream` like an asynchronous `Iterator`; like a `Future`, it represents some sort of data that will be obtained at a later point in time, but *unlike* a `Future` it can produce *multiple* results over time. It has many of the same combinators as an `Iterator` does. Essentially, a `Stream` is the "multiple results over time instead of one" counterpart to a `Future`.
- **Executor:** An `Executor` is a thing that, when you pass a `Future` or `Stream` into it, is responsible for 1) turning the logic stored in the `Future`/`Stream` into an internal task, 2) scheduling the work for that task, and 3) wiring up the Future's state to any underlying resources. You don't usually implement these yourself, but use a pre-made `Executor` from some third-party library. The exact scheduling is left up to the implementation of the `Executor`.
- **`.wait()`:** A method on a `Future` that will block the current thread until the `Future` has completed, and that then returns the result of that `Future`. This is an example of an `Executor`, although not a particularly useful one; it won't allow you to do work concurrently.
- **Tokio reactor core:** This is also an `Executor`, provided by the Tokio library. It's probably what you'll be using when you use Tokio. The frontpage of the Tokio website provides an example on how to use it.

- **`futures_cpupool`:** Yet another `Executor`; this one schedules the work across a pool of threads.