SUMMARY ON ARTICLE : “async/.await”

In Rust programming language the most predictable feature is finally put in, named as “async/.await”. Here I gave short summary on asynchronous programming in rust , here we go.

We understand async function by taking example of simple application that take some data from internet by using our new function’s ability ‘asyn’. Before going on coding we discuss about basic concept of asynchronous programming in rust .

**Async in rust:**

In rust async means run multiple program in one time or in single thread. It is useful for IO bound task which consumes time. For getting result of asynchronous programming we must .await it . In rust awaitable value known as “future” value.

**Rusty strangeness:**

Asynchoronous programming in rust is different from other programming language like java script, C# etc. Here is the main point of asynchoronus programming that we need to understand.

**An Async func execution:**

Async func does not execute immediately . We need .await or an executer to start it. Lets take an example.

**use async\_std::task;**

**// ^ we need this for task spawning**

**async fn negate\_async(n: i32) -> i32 {**

**println!("Negating {}", n);**

**task::sleep(std::time::Duration::from\_secs(5)).await;**

**println!("Finished sleeping for {}!", n);**

**n \* -1**

**}**

**async fn f() -> i32 {**

**let neg = negate\_async(1);**

**// ... nothing happens yet**

**let neg\_task = task::spawn(negate\_async(2));**

**// ^ this task /is/ started**

**task::sleep(std::time::Duration::from\_secs(1)).await;**

**// we sleep for effect.**

**neg.await + neg\_task.await**

**// ^ this starts the first task `neg`**

**// and waits for both tasks to finish**

**}**

**The explanation of the above code are as follow:**

1. In first line,We import external library “async\_std::task” because standard library does not provide executer.
2. The async function “negate\_async” take signed integer as input sleep for 5 sec and return negative value.
3. In Async function “f”: Let neg create future of negat\_async but does not execute. Let neg\_task uses task::spwan function to start executing future of negat\_async and assigned to neg \_task . Now sleep for a second which can see in output . now add neg and neg\_task . We start executing future and wait for completion of neg\_task.

The Output of above Task is:

Negating 2

# <- there's a 1 second pause here

Negating 1

Finished sleeping for 2!

Finished sleeping for 1!

As we see neg\_task is executed by task::spwan help while neg is not start executing until it is awaited.

**External linrary to use async/.await:**

As we discussed above that we need to use external library for async/.await because the standard library do not have executer. Executer executing future ,canvasing them and return the result whenever we need it. The most famous executer are “asyn-std” and “tokio”.

**A Small Example Of Async.**

Lets take pokemon simple application that take data and print in console. For this we need 1.39 version of rust and cargo available>

**Step no. 1.**

**Creating the application.**

cargo new async-basics

**Step no. 2:**

**Dependencies**

Using asyn-std for spawning tasks, and surf to take data from the API and add them to Cargo.toml file. Your file look like this.

[package]

name = "async-basics"

version = "0.1.0"

authors = ["Your Name <your.email@provider.tld>"]

edition = "2018"

[dependencies]

async-std = "1"

surf = "1"

**Step no. 3:**

**Fetch data:**

Modify main.rs file and make it simple as possible.

use **async\_std::**task;

use surf;

// fetch data from a url and return the results as a string.

// if an error occurs, return the error.

async fn fetch(url: &str) -> Result<String, **surf::**Exception> {

**surf::**get(url).recv\_string().await

}

// execute the fetch function and print the results

async fn execute() {

match fetch("https://pokeapi.co/api/v2/move/surf").await {

**Ok**(s) => **println!**("Fetched results: {:#?}", s),

**Err**(e) => **println!**("Got an error: {:?}", e),

};

}

fn main() {

**task::**block\_on(execute());

// ^ start the future and wait for it to finish

}

Divide code into few parts for understanding:

**Use statement:**

In this importing the crate which we use in cargo.toml file, surf and async.

Fetch:

It wrapped surf::get function which return either playload as a String or Exception if any error occur:

**Execute:**

This function called fetch to the end point for the move surf, wait for the result matched if matched output print in console or else print error.

**Main:**

We use task::block\_on synchronous counterpart to start task::spawn asynchronous operator because we cant use .await directly in main that’s why use asynchronous operator.

**Step no. 4:**

**Can be Extended;**

You can add multiple task in this and clear your concepts ore about async/.await.