Hack `postgres` Source Code: Vol I

Chapter 1 : C & Rust

1.17 Collections in Rust - Vector. Go Away C, Its Not Yours!!!

Rust has some built in collections (in other words.. data structures) to ease the burden of handling data.

So, what are those collections? Phew.. Data structures ;-)?

`Vector`. I can think of `Vector` now and it is same as array, you know.

Then, How come does it more powerful thatn array?

`Vector` is same as array but not exactly. Because, it grows or shrinks automatically. But, in array, we set fixed size to array and no other element is allowed it if that size is exhausted.

Ok. Then, but, how exactly `Vector` is same as array?

`Vector` stores same type of element as array does. And also, it stores those elements in continuous memory locations. It doesn't store elements here and there in memory. I think you know this one.. but.. let me tell you.. `Vector` data is organized in stack and heap memory in RAM which is same as array's.

Oh!! Interesting... Tell me some more details of this `Vector`. He.. He.. your `Vector`. Do you know what I mean? ;-)

(grinning) It is not mine. It belongs to Rust which is developed by bunch of super cool geeks in firefox team who got vexed with c/c++.

Whatever.. tell me more .. dude.. it is intriguing..

Hmm... ha.. we can declare rust in two ways. One is using `Vector` macro and another is with a kind of style that `Java` uses more.

Show me those styles...

Where?? we don't have any white board here but only a pen in my pocket..

Show me on my hand..

Hmm.. Ok.. girl..

It is like.. (On hand) `let java_style_vector: Vec<i32> = Vec::new();` and `let macro_style_vector: Vec<i32> = vec![];`

So, does writing like `Vec<i32>` make `Vector` to story only `i32` type elements?

Yeah.. kind of.. actually, `i32` in angular brackets makes that possible and `Vec` is like notation and we can use any type between those angular brackets of `Vec` to enfore only those types in `Vector`.

Oh.. Got it. I think, you initialized immutable vector. But, how about adding elements to that vector?

Yeah. They are immutable vectors. Vectors are just like other types. If we want to add elements after vector initialization, you need to initialize it as mutable using `mut`.

Ok. Let's say, I initialized vector as mutable. Now, tell me, how to add elements to that vector.

There is method called `push` in vector. We can use it to add elements to it..

Why not `add` and `insert` name? but `push` name..

I don't know.. Rust developers used `push` so am I. When I get to meet them, I will ask them why they named it like that. ;-)

He.. He.. tell me some complex stuff in this vector.

You want complex!!! Then, let me give you details of vector with references.

Ok. you know.. I like references because they are somewhat simpler than pointers to me. go on..

Hmm. If you create a reference to vector element, you are not allowed to add any more elements to it further..

Really? Why?

Because, for example, if you initialize vector with some values like `let v: Vec<i32> = vec![1,2,3];` , rust looks for a free space in heap and stores all these elements in continuos memory locations available in that free space. Then, if you create a reference to an element in vector like `let ref:&i32 = &vec[0];` , rust gets memory location address of first element in vector and assign it to `ref` variable. Later, if I want to add an element to `v` like `v.push(4)` , rust throws an error. Because... Because...

Why? tell me..

Because.. rust doesn't want to take a chance.

What do you mean by taking a chance in this case?

Means.. how can I put it in simple way? Ok.. just assume a case in which rust goes to heap memory and finds free space to store only those threee elements - `1,2,3` . Does rust store those in that area or does rust looks for another memory area which is big enough to store remaining elements that might come into vector `v` later?

Hm.. in future.. elements may or may not be added to vector. So, rust just store them in that tiny free space in heap memory..

Correct. But, what if we add another element to `v` again like `v.push(4)`.

Rust needs to looks for another free space in heap because it has to store all the elements in continuous memory locations..

You got it, babe. But, what if rust copies those elements to new memory area in heap, Is there any impact to our reference variable `ref` which stored memory location of old memory area in heap for a element in `v`?

That `ref` would be in a state of poiting to unused memory location where there is no value.

Go on.

Ohhhhh... I got it now. Because of this kind of cases which may or may not happen.. rust is not allowing addition of values after initialization of reference variable to vector element.

Yep..

Ok.. I gotta go.. man.. We will meet again in lunch.

By the way.. Its nice meeting you in this college canteen.

Its nice meeting you too.