CnosDB

Use Rust To Make A TSDB

rust 入门基础 （六）

Lecturer: Rose

Date: 2022.06.14

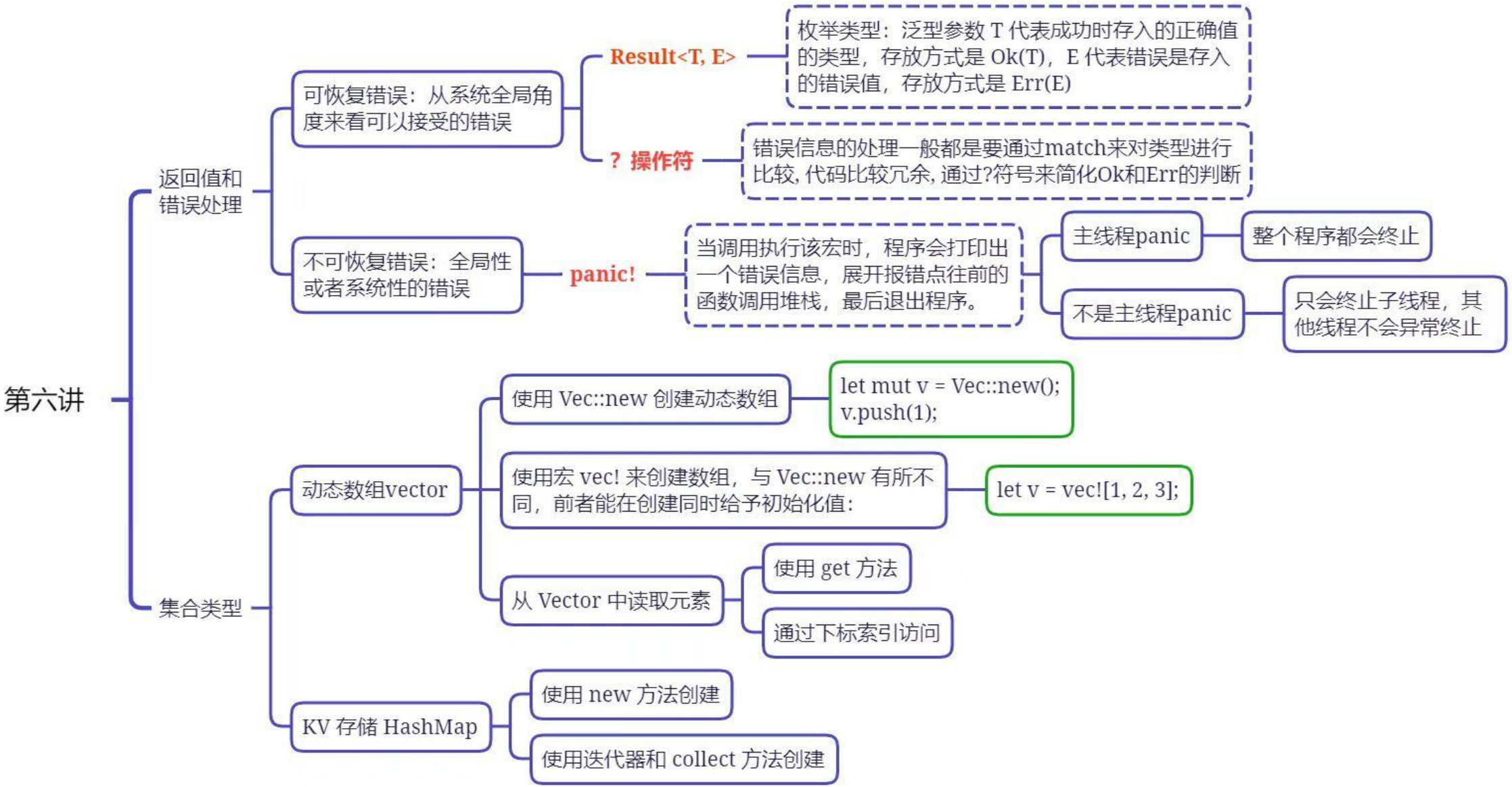
Welcome to follow the GitHub repo

欢迎关注我们的代码仓库

https://github.com/cnosdb/cnosdb

GitHub

rust 入门基础　回顾


第六讲
返回值和错误处理
可恢复错误：从系统全局角度来看可以接受的错误
Result<T, E>
枚举类型：泛型参数 T 代表成功时存入的正确值的类型，存放方式是 Ok(T)，E 代表错误是存入的错误值，存放方式是 Err(E)
? 操作符
错误信息的处理一般都是要通过match来对类型进行比较，代码比较冗余，通过?符号来简化Ok和Err的判断
不可恢复错误：全局性或者系统性的错误
panic!
当调用执行该宏时，程序会打印出一个错误信息，展开报错点往前的函数调用堆栈，最后退出程序。
主线程panic
整个程序都会终止
不是主线程panic
只会终止子线程，其他线程不会异常终止
集合类型
动态数组vector
使用 Vec::new 创建动态数组
let mut v = Vec::new();
v.push(1);
使用宏 vec! 来创建数组，与 Vec::new 有所不同，前者能在创建同时给予初始化值：
let v = vec![1, 2, 3];
从 Vector 中读取元素
使用 get 方法
通过下标索引访问
KV 存储 HashMap
使用 new 方法创建
使用迭代器和 collect 方法创建

rust 入门基础

CnosDB

指针是个通用概念，它表示内存地址这种类型，其引用或“指向”其他数据。Rust中的指针是“第一类公民”（first-class values），可以将它们移动或复制，存储到数据结构中并从函数中返回

Rust提供了多种类型的指针

- 引用（Reference），共享引用&，可变引用&mut
- 原生指针,又叫裸指针（Raw Pointer），*const和*mut
- 智能指针（Smart Pointer），Box，Rc

rust 入门基础

CnosDB

Rust 可以划分为 Safe Rust 和 Unsafe Rust 。

- 引用　主要应用于 Safe Rust。

在Safe Rust 中，编译器会对引用进行借用检查，以保证内存安全和类型安全。

- 原生指针 主要用于 Unsafe Rust。

原生指针不在 Safe Rust 的控制范围内，需要编程人员自己保证安全。

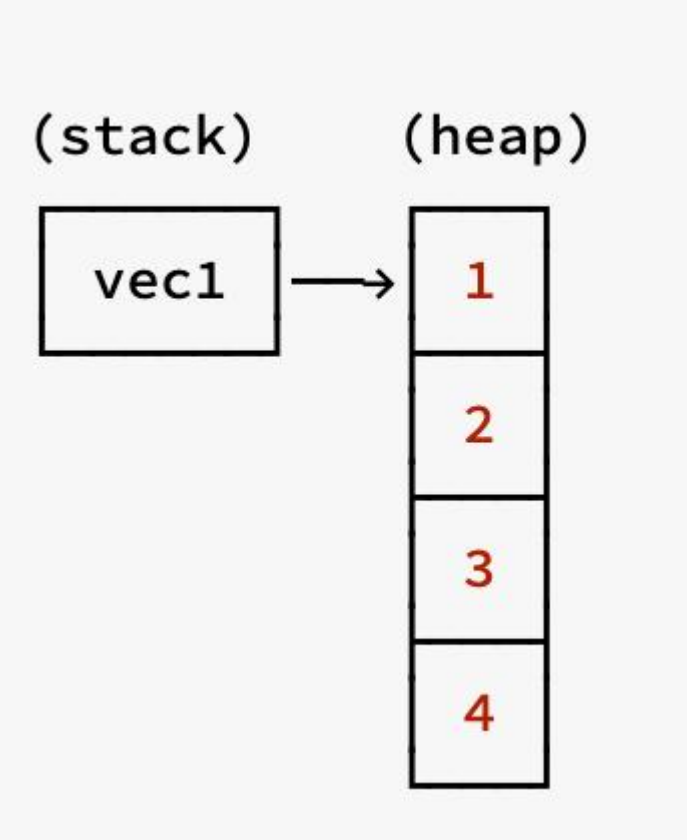
rust 入门基础　智能指针—Box<T>

CnosDB

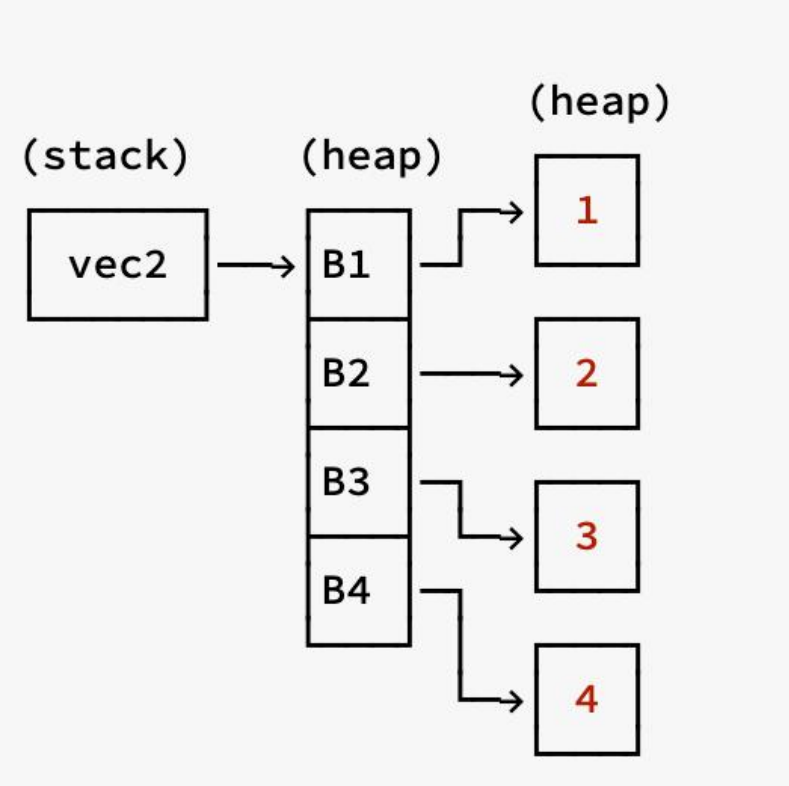
Box<T>

- Rust 中的值默认被分配到 栈内存，通过 Box<T> 在堆内存中分配。
- Box<T> 是指向类型为T的堆内存分配值的智能指针。
- 当 Box<T> 超出作用域范围时，将调用其析构函数，销毁内部对象，并自动释放内存。
- 可以通过解引用操作符 * 来获取Box<T>中的 T
- Box<T> 的行为像引用，并可以自动释放内存，所以称为智能指针。

rust 入门基础



vec<i32>



vec<Box<i32>>

rust 入门基础　Deref 和 Drop

CnosDB

智能指针往往是基于结构体实现，它与我们自定义的结构体最大的区别在于它实现了 Deref 和 Drop 特征

Deref 可以让智能指针像引用那样工作，这样你就可以写出同时支持智能指针和引用的代码，例如 *T

Drop 允许你指定智能指针超出作用域后自动执行的代码，回收内存资源

例如做一些数据清除等收尾工作

CnosDB

Q&A

Welcome to follow the GitHub repo

欢迎关注我们的代码仓库

https://github.com/cnosdb/cnosdb

GitHub