14智能指针

2020年3月9日 12:24

经常使用到的智能指针：
Box<T>：用于在堆上分配值.
Rc<T>：一个引用计数类型,其数据可以有多个所有者.不能使用在多线程中.
Ref<T>和RefMut<T>,通过RefCell<T>访问,一个在运行时而不是在编译时执行借用规则的类型.

1：Box<T>使用的场景
(1):当有一个编译时未知大小的类型,而又想要在需要确切大小的上下文中使用这个类型值的时候
Rust需要在编译时知道类型占用多少空间.一种无法在编译时知道大小的类型是递归类型,
其值的一部分可以是相同类型的另一个值.这种值的嵌套理论上可以无限的进行下去,所以Rust
不知道递归类型需要多少空间.不过Box有一个已知的大小.
(2):当有大量数据并希望在确保数据不被拷贝的情况下转移所有权的时候
(3):当希望拥有一个值,并只关心它的类型是否实现了特定的trait而不是具体类型的时候,比如trait对象.
2：Rc<T>和引用的区别：
(1):Rc<T> 用于当我们希望在堆上分配一些内存供程序的多个部分读取,
而且无法在编译时确定程序的哪一部分会最后结束使用它的时候。
(2)：Rc<T>允许所有权移动,最后一个负责drop,引用则不能.
注意：Rc<T>是分配在堆上的.
3：RefCell<T>：
类似于 Rc<T>，RefCell<T> 只能用于单线程场景。如果尝试在多线程上下文中使用RefCell<T>，会得到一个编译错误。
Rc<T> 允许相同数据有多个所有者；Box<T> 和 RefCell<T> 有单一所有者。
Box<T> 允许在编译时执行不可变或可变借用检查；Rc<T>仅允许在编译时执行不可变借用检查；RefCell<T> 允许在运行时执行不可变或可变借用检查。
因为 RefCell<T> 允许在运行时执行可变借用检查，所以我们可以在即便 RefCell<T> 自身是不可变的情况下修改其内部的值。
当创建不可变和可变引用时，我们分别使用 & 和 &mut 语法。
对于 RefCell<T> 来说，则是 borrow 和 borrow_mut 方法，这属于 RefCell<T> 安全 API 的一部分。
borrow 方法返回 Ref 类型的智能指针，
borrow_mut 方法返回 RefMut 类型的智能指针。
这两个类型都实现了 Deref，所以可以当作常规引用对待。

RefCell<T> 记录当前有多少个活动的 Ref<T> 和 RefMut<T> 智能指针。
每次调用 borrow，RefCell<T> 将活动的不可变借用计数加一。当 Ref 值离开作用域时，不可变借用计数减一。
就像编译时借用规则一样，RefCell<T> 在任何时候只允许有多个不可变借用或一个可变借用。
如果我们尝试违反这些规则，相比引用时的编译时错误，RefCell<T> 的实现会在运行时 panic!。

内部可变性：
RefCell是单线程,多线程的.对应的多线程的就是Mutex

引用计数：
Rc是单线程的,对应的多线程的就是原子引用计数Arc