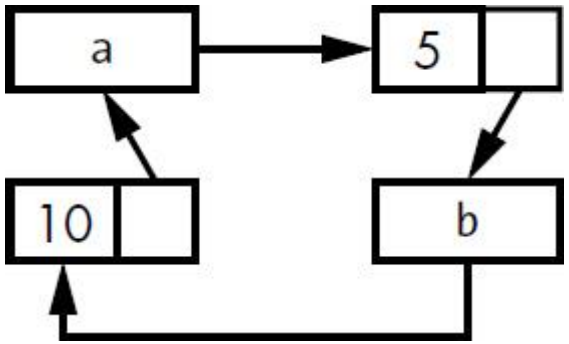
Rust 编程语言入门

杨旭，微软 MVP
Rust、Go 开发者

15.6 循环引用可导致内存泄漏

Rust 可能发生内存泄漏

- Rust 的内存安全机制可以保证很难发生内存泄漏，但不是不可能。
- 例如使用 Rc<T> 和 RefCell<T> 就可能创造出循环引用，从而发生内存泄漏：
  - 每个项的引用数量不会变成 0，值也不会被处理掉。
- （例子）



防止内存泄漏的解决办法

- 依靠开发者来保证，不能依靠 Rust
- 重新组织数据结构：一些引用来表达所有权，一些引用不表达所有权
  - 循环引用中的一部分具有所有权关系，另一部分不涉及所有权关系
  - 而只有所有权关系才影响值的清理

防止循环引用
把 Rc<T> 换成 Weak<T>

- Rc::clone 为 Rc<T> 实例的 strong_count 加 1， Rc<T> 的实例只有在 strong_count 为 0 的时候才会被清理
- Rc<T> 实例通过调用 Rc::downgrade 方法可以创建值的 Weak Reference（弱引用）
  - 返回类型是 Weak<T>（智能指针）
  - 调用 Rc::downgrade 会为 weak_count 加 1。
- Rc<T> 使用 weak_count 来追踪存在多少 Weak<T>。
- weak_count 不为 0 并不影响 Rc<T> 实例的清理

Strong vs Weak

- Strong Reference（强引用）是关于如何分享 Rc<T> 实例的所有权
- Weak Reference（弱引用）并不表达上述意思
- 使用 Weak Reference 并不会创建循环引用：
  - 当 Strong Reference 数量为 0 的时候，Weak Reference 会自动断开
- 在使用 Weak<T> 前，需保证它指向的值仍然存在：
  - 在 Weak<T> 实例上调用 upgrade 方法，返回 Option<Rc<T>>
- （例子）

再见