»

Rust Patterns: Enums Instead Of Booleans

07 May 2019

As I've been writing Rust code more, I've noticed how few boolean types I'm using in my code. Instead, I'm using Rust's powerful enums in 90% of cases where I would have reached for a boolean in another language.

A few reasons why I think this is:

1. Binary state can easily be represented using enums, with two variants.
2. Binary states can easily be promoted to tri-states with enums. Booleans can only have two variants ever.
3. Because of Rust's amazing exhaustive pattern matching, it's safe to add more variants in the future and ensure all cases are handled.
4. The code intention is much more readable, more self documenting, and easier to grok.

Consider this function signature, with a boolean to indicate why the block was rotated:

```
fn rotate_block(
    &self,
    current_block: &mut BlockMut,
    timed_out: bool
) -> Result<(), io::Error> {
    ...
    if timed_out {
        println!("Rotated block due to timeout");
    }
    Ok(())
}
```

And the caller:

```
self.rotate_block(&mut current_block, true)?;
```

If the code above is far away from the definition of `rotate_block`, it's hard to figure out what the last boolean `true` is supposed to mean.

Here's the refactored version:

```
#[derive(Debug)]
enum BlockRotateReason {
    Full,
    Timeout,
}

fn rotate_block(
    &self,
    current_block: &mut BlockMut,
    reason: BlockRotateReason
) -> Result<(), io::Error> {
    ...
    match reason {
       BlockRotateReason::Timeout => println!("Rotated block due to timeout"),
       BlockRotateReason::Full => println!("Rotated block due to being full"),
    }
    Ok(())
}
```

The caller changes to:

```
self.rotate_block(&mut current_block, BlockRotateReason::Timeout)?;
```

From the call-site, this is much better. It's clearer what that final function argument means: The block was rotated because of a timeout.

The question I asked myself: *Why did I reach for booleans more in other languages, rather than enums?*. The big win for Rust here is: **Exaustive, pattern matched enums add a layer of safety**. If you fail to cover a variant of an enum in calling code, the compiler will complain.

Yes, C, Java, and other languages have enums, but they offer weaker guarantees than Rust. With these weaker semantics, sometimes it's easier to reach for a boolean first.

---

about the author

Blake Smith is a Principal Software Engineer at Sprout Social.