Generic Enum Type in Rust

Adrian Kosmaczewski

2021-05-14

I continue my exploration of Rust through a simple implementation of the Active Record design pattern.

Last week I told the story of how I had created a simple `Property<T>` type, and a collection thereof. But of course a real object has many properties, and of various different types at once; so such a `PropertyMap<T>` was not exactly what I needed, unless all of your properties are the same type.

To solve that issue, in my original C++ implementation from 2008 I used the Poco::Any type, itself based on the original Boost library of the same name, created by Kevlin Henney.

I quickly found out that Rust had an Any trait, but after some fiddling I could not make it work the way I wanted.

That had to do more with my lack of knowledge of Rust than anything else; see, Rust's `Any` is a trait, not a struct, so the semantics in my head were, of course, twisted, and I could not find a solution.

Thankfully DuckDuckGo was kind enough to provide me with the answer, in the form of a blog post by Simone Vittori published last year. The idea is simple, and will sound familiar to those using Swift, by the way: use an `enum`.

```
#[derive(Clone, Debug)]
pub enum AnyProperty {
    Str(Property<&'static str>),
    Int(Property<i32>),
    Float(Property<f64>),
    Bool(Property<bool>),
    DateTime(Property<DateTime<Utc>>),
}
```

Now I can hold many of these in the same bag:

```
#[derive(Clone, Debug)]
pub struct AnyPropertyMap {
    properties: HashMap<String, AnyProperty>,
}
```

The whole API is very simple to use now:

```
#[test]
fn create_any_property_map() {
    let a = AnyProperty::from_int("int", 22);
    let b = AnyProperty::from_str("str", "value");
    let c = AnyProperty::from_float("float", 1.234);
    let d = AnyProperty::from_bool("bool", true);
    let e = AnyProperty::from_datetime("date", Utc::now());
    let m = AnyPropertyMap::from(vec![a, b, c, d, e]);
    assert_eq!(m.len(), 5);
}
```

And thanks to pattern matching, translating such properties to strings becomes very simple and straighforward. This will look familiar to Swift developers; the influence of Rust in Swift is here more visible than anywhere else, in my opinion.

```
pub fn get_value(&self) -> String {
    return match self {
        AnyProperty::Str(prop) => format!("'{}'", prop.value),
        AnyProperty::Int(prop) => format!("{}", prop.value),
        AnyProperty::Float(prop) => format!("{}", prop.value),
        AnyProperty::Bool(prop) => {
            let val = if prop.value { "TRUE" } else { "FALSE" };
            format!("{}", val)
        }
        AnyProperty::DateTime(prop) => format!("'{}'", prop.value.to_rfc3339()),
    };
}
```

To be able to handle date and time information, I chose the chrono crate, which provides a very handy and useful DateTime struct to work with.

I started working on the actual ActiveRecord pattern implementation, and I have published the code as usual in my GitLab account. Feel free to clone and play with it. The code is already bundled with as many unit tests as I could fit, and more will come. The usual `cargo test` command will execute them for your testing pleasure.

The final objective of this exercise is to achieve an API that will look more or less like this:

```
let mut o = Object::create();
o.set_name("Milk");
o.set_price(1.23);
o.set_valid(true);
o.save();

o.set_price(2.56);
```

```
o.set_valid(false);
o.save();
```

And at the end, a SQLite file will be generated with those objects stored in a table. I am planning to explore Rust's macro system and syntax in order to generate most of the boilerplate code required for this library to be useful.

I know that many consider Active Record an antipattern, but I do not pretend this library to become a production thing anytime soon; of course, the Internet being what it is, I added a BSD 3 license to the bundle, just in case.

Upcoming Explorations

Another thing I started exploring this week was how to create web applications with Rust; in particular I built a simple web application using Actix with the Askama template library, and then wrapping the whole thing inside a Docker container image.

But I will leave that for a future post. In short, I loved it; the final code is ultra fast, in the shape of a very small self-contained binary, perfectly suitable for a deployment in Kubernetes. I rewrote a very simple Flask application in a few hours. More soon!