perplexity

how return self in trait

To return `Self` in a trait, you can directly use `Self` as the return type in the trait definition, provided that `Self` is `Sized`[1]. The `Sized` trait bound indicates that the size of the type is known at compile time[2] [3].

```
trait T {
    fn f(&self) -> Self;
}

struct S;

impl T for S {
    fn f(&self) -> Self {
        S
    }
}
```

However, when returning `Option<Self>`, the compiler might require an explicit `Sized` bound because `Option<T>` requires `T` to be `Sized`[2] [3]. This is because `Option<T>` owns its data, and all type parameters to a concrete type definition are by default constrained as `Sized`[2].

```
trait X where Self: Sized {
    fn parent(&self) -> Option<Self>;
}

#[derive(Clone)]
struct Bar;

impl X for Bar {
    fn parent(&self) -> Option<Self> {
        return Some(self.clone());
    }
}
```

If you encounter an error stating that the size for values of type `Self` cannot be known at compilation time, you can add a `where Self: Sized` bound to the trait definition[2] [3]. This ensures that the trait is only implemented for types whose size is known at compile time[2].

```
trait MyTrait where Self: Sized {
    fn foo(&self) -> Option<Self>;
}
```

Alternatively, you can use `impl Trait` as the return type for a function within a trait definition. The impls that implement this trait return "some type that implements `Trait`", and users of the trait can only rely on that[4].

```
trait IntoIntIterator {
    fn into_int_iter(self) -> impl Iterator<Item = u32>;
}

impl IntoIntIterator for Vec<u32> {
    fn into_int_iter(self) -> impl Iterator<Item = u32> {
        self.into_iter()
    }
}
```

⁂

1. https://users.rust-lang.org/t/return-self-in-trait/31400
2. https://stackoverflow.com/questions/54465400/why-does-returning-self-in-trait-work-but-returning-optionself-requires
3. https://users.rust-lang.org/t/returning-option-self-in-a-trait/28081
4. https://rust-lang.github.io/impl-trait-initiative/explainer/rpit_trait.html