Rust - 应用错误处理的实践思路

作者：plus7wist
原文链接：https://ld246.com/article/1592795294902
来源网站：链滴

软件开发一方面要 fail fast，另一方面又讨厌 exception，我认为应该辩证的看待这个矛盾。不论是出异常，还是用其他的错误处理方式，最终都体现在错误用户界面上。最终界面，是运行时接住异常生的，或者是某种框架接住了异常产生的，都是可行的办法，要看应用的需求如何。不过总之在相互直的逻辑聚集的地方，就需要错误处理聚集；在逻辑不垂直的地方，错误倾向于交给上层处理。

在比较简单的系统里，这种「垂直聚集」只发生在一个地方，就是主函数里。我用 anyhow 包来解释种思路。它的文档在：https://docs.rs/anyhow。

要完成一个逻辑独立的任务，可能产生的错误有两种，一种是使用的包的逻辑产生的错误，这可能有多种，但按照约定，它们多实现了 std::error::Error，所以可以用 ? 运算符把它们都转换成 anyhow::Error；另一种是根据任务本身产生的错误，可以使用 anyhow::anyhow 宏来产生错误，或者用 anyhow::bail 宏直接在错误处退出。这说明它返回的错误是 anyhow::Error，而整体返回值是 Result<T, anyhow::Error> 或者 anyhow::Result<T>。用一段伪代码来解释这个格式：

```
fn create_some_client(config: ClientConfig) -> anyhow::Result<Client> {
  use std::fs::File;
  // 文件错误
  let data_file = File::open(config.data_file_path)?;

  // 反序列化错误
  let data: ClientData = serde_json::from_reader(data_file)?;

  use anyhow::bail;
  if data.go_east && data.go_west {
    // 任务逻辑错误
    bail!("Cannot go to both the east and the west")
  }

  let client = create_client_with_data(config, data)?;

  Ok(client)
}
```

对于这个创建客户端的任务来说，无论是文件错误、反序列化错误、逻辑错误，还是其他错误，都不让调用这个任务的用户操心。恰恰相反，只需要知道这个错误是创建客户端的错误即可。这种场景下标识错误的任务就交给了使用者：

```
use anyhow::Context;

mod client;
use client::*;

fn main() -> anyhow::Result<()> {
  let config = load_client_config()
    .context("Failed to load client config")?;

  let client = create_client(&config)
    .context("Failed to create client")?;

  use_client(&client).context("Use Client")?;
  Ok(())
}
```

这就是 anyhow::Context 的使用场景。

如果不是这样简单的场景，我们可以选择继续使用 anyhow::Error，也就是放弃错误的类型帮助，那就需要一种向下转型的方式来帮助「上层的上层」区分上层是什么错误。

```
match root_cause.downcast_ref::<DataStoreError>() {
    Some(DataStoreError::Censored(_)) => Ok(Poll::Ready(REDACTED_CONTENT)),
    None => Err(error),
}
```

这就跟一些动态类型语言的异常没有区别了。可能有人喜欢有人讨厌。

不然的话，我们不可避免的要创造错误类型，这就是一种新的需求了：可以使用 thiserror 来完成它它的文档在：https://docs.rs/thiserror。它精巧的设计了宏，多数情况下都会使构建错误类型的痛苦得微不足道。

```
use thiserror::Error;

#[derive(Error, Debug)]
pub enum DataStoreError {
    #[error("data store disconnected")]
    Disconnect(#[from] io::Error),
    #[error("the data for key `{0}` is not available")]
    Redaction(String),
    #[error("invalid header (expected {expected:?}, found {found:?})")]
    InvalidHeader {
        expected: String,
        found: String,
    },
    #[error("unknown data store error")]
    Unknown,
}
```