并在结尾处添加了一个返回值 Ok(())。现在这段代码可以通过编译了：

src/main.rs

```
use std::error::Error;
use std::fs::File;

fn main() -> Result<(), Box<dyn Error>> {
    let greeting_file = File::open("hello.txt")?;

    Ok(())
}
```

示例 9-12：将 main 函数的返回类型修改为 Result<(), E>，使?运算符可以被用在 Result 值上

这里的 Box<dyn Error>被称作 trait 对象，我们将在第 17 章的“使用 trait 对象存储不同类型的值”一节中讨论它。现在，你可以简单地将 Box<dyn Error> 理解为“任何可能的错误类型”。由于 main 函数使用了 Box<dyn Error>作为错误类型，它允许任意的 Err 值提前返回，所以我们可以在 Result 值的后面使用?运算符。尽管这个 main 函数只会返回 std::io::Error 这一种错误类型，但通过签名中指定的 Box<dyn Error>，我们可以合法地在 main 函数中返回其他错误类型。

假如 main 函数的返回类型是 Result<(), E>，那么可执行文件会在 main 函数返回 Ok(())时以 0 值退出，而在 main 函数返回 Err 值时以非 0 值退出。使用 C 语言编写的可执行文件会在它们退出时返回一个整数：成功运行并退出的程序返回 0，产生了错误的程序则会返回一个非 0 整数。为了与这种惯例相兼容，Rust 同样会在可执行文件退出时返回整数。

main 函数可以返回任何实现了 std::process::Termination trait 的类型，它包含了一个 report 函数来指定 ExitCode。你可以参考标准库文档来学习如何为自己的类型实现 Termination trait。

至此，我们已经讨论了足够多关于调用 panic!或返回 Result 的内容。让我们再来看一看它们各自的适用场景吧。