Systemnahe Programmierung in Rust

- “The Book” / Error Handling / Kap. 9 -

Hubert Högl

Hochschule Augsburg / Informatik

2023-11-02 22:42:41

Error Handling

→ https://doc.rust-lang.org/book/ch09-00-error-handling.html

Fehler sind …

- *recoverable*

  `Result<T, E>`

  oder

- *unrecoverable*

  `panic!();`

  `panic("Nichts geht mehr");`

panic!()

Zwei mögliche Verhaltensweisen

- *stack unwinding*

  `RUST_BACKTRACE=1 cargo run`

- *abort*

Beispiel für Auslöser zur Laufzeit

```
fn main() {
    let v = vec![1, 2, 3];
    v[99];
}
```

Result<T, E>

```
enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

`Ok()` und `Err()` auch ohne `Result::` Präfix (Prelude).

```
use std::fs::File;

fn main() {
    let greeting_file_result = File::open("hello.txt");
}
```

`open("hello.txt")` liefert zurück ..

- Erfolg: `Ok(T)`, `T` ist vom Typ `std::fs::File`
- Fehler: `Err(E)`, `E` ist vom Typ `std::io::Error`

Fehler behandeln mit `match`

```rust
use std::fs::File;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => panic!("Problem opening the file: {:?}", error),
    };
}
```

Verschiedene Fehler

Verschachtelte `match` Anweisungen eher schlecht zu lesen

```
use std::fs::File;
use std::io::ErrorKind;

fn main() {
    let greeting_file_result = File::open("hello.txt");

    let greeting_file = match greeting_file_result {
        Ok(file) => file,
        Err(error) => match error.kind() {
            ErrorKind::NotFound => match File::create("hello.txt") {
                Ok(fc) => fc,
                Err(e) => panic!("Problem creating the file: {:?}", e),
            },
            other_error => {
                panic!("Problem opening the file: {:?}", other_error);
            }
        },
    };
```

Bessere Lösung mit unwrap_or_else()

```rust
use std::fs::File;
use std::io::ErrorKind;

fn main() {
    let greeting_file = File::open("hello.txt").unwrap_or_else(|error| {
        if error.kind() == ErrorKind::NotFound {
            File::create("hello.txt").unwrap_or_else(|error| {
                panic!("Problem creating the file: {:?}", error);
            })
        } else {
            panic!("Problem opening the file: {:?}", error);
        }
    });
}
```

Shortcuts mit unwrap() und expect()

```rust
use std::fs::File;

fn main() {
    let greeting_file = File::open("hello.txt").unwrap();
}
```

```rust
fn main() {
    let greeting_file = File::open("hello.txt")
        .expect("hello.txt should be included in this project");
}
```

Fehler weiterleiten

```rust
use std::fs::File;
use std::io::{self, Read};

fn read_username_from_file() -> Result<String, io::Error> {
    let username_file_result = File::open("hello.txt");

    let mut username_file = match username_file_result {
        Ok(file) => file,
        Err(e) => return Err(e),
    };

    let mut username = String::new();

    match username_file.read_to_string(&mut username) {
        Ok(_) => Ok(username),
        Err(e) => Err(e),
    }
}
```

Kürzer mit ?

```rust
use std::fs::File;
use std::io::{self, Read};

fn read_username_from_file() -> Result<String, io::Error> {
    let mut username_file = File::open("hello.txt")?;
    let mut username = String::new();
    username_file.read_to_string(&mut username)?;
    Ok(username)
}
```

Noch kürzer

```rust
use std::fs::File;
use std::io::{self, Read};

fn read_username_from_file() -> Result<String, io::Error> {
    let mut username = String::new();

    File::open("hello.txt")?.read_to_string(&mut username)?;

    Ok(username)
}
```

Es geht immer noch kürzer ...

```rust
use std::fs;
use std::io;

fn read_username_from_file() -> Result<String, io::Error> {
    fs::read_to_string("hello.txt")
}
```

Typen passen nicht

```rust
use std::fs::File;

fn main() {
    let greeting_file = File::open("hello.txt")?;
}
```

Möglichkeiten:

- Returnwert der Funktion ändern
- `match` einbauen oder Methoden auf `Return<T, E>` wählen z.B. `unwrap_or_else()`

? geht auch für Option

```rust
fn last_char_of_first_line(text: &str) -> Option<char> {
    text.lines().next()?.chars().last()
}
```

Gerne mal ausprobieren, was die Funktion macht (Testfälle)

main()

Trait object `Box<dyn Error>` passt auf jeden Fehler.

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

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

    Ok(())
}
```

Exit-Code des Programms ist `0` bei `Ok(())` und `1` bei `Err`.

To panic! or Not to panic!

Panic bei

- Beispielen
- Prototypen
- Tests