09-grepr

目标：实现 grep 命令

- -i | --ignore-case：忽略大小写
- -v | --invert-match：查找不满足模型的行
- -c | --count：返回行数
- -r | --recursive：递归查找目录下的所有文件

实现

- 解析参数

Rust

```rust
#[derive(Debug)]
pub struct Config {
    pattern: Regex,
    files: Vec<String>,
    recursive: bool,
    count: bool,
    invert_match: bool,
}

let pattern = RegexBuilder::new(pattern)
    .case_insensitive(matches.is_present("insensitive"))
    .build()
    .map_err(|_| format!("Invalid pattern \"{}\"", pattern))?;
```

- The `RegexBuilder::new` method will create a new regular expression.
- The `RegexBuilder::case_insensitive` method will cause the regex to disregard case in comparisons when the insensitive flag is present.（忽略大小写）
- The `RegexBuilder::build` method will compile the regex.
- If build returns an error, use `Result::map_err` to create an error message stating that the given pattern is invalid.

- 查找文件

Rust

```rust

fn find_files(paths: &[String], recursive: bool) -> Vec<MyResult<String>> {
    let mut results = vec![];

    for path in paths {
        match path.as_str() {
            "-" => results.push(Ok(path.to_string())),
            _ => match fs::metadata(path) {
                Ok(metadata) => {
                    if metadata.is_dir() {
                        if recursive {
                            for entry in WalkDir::new(path)
                                .into_iter()
                                .flatten()
                                .filter(|e| e.file_type().is_file())
                            {
                                results.push(Ok(entry
                                    .path()
                                    .display()
                                    .to_string()));
                            }
                        } else {
                            results.push(Err(From::from(format!(
                                "{} is a directory",
                                path
                            ))));
                        }
                    } else if metadata.is_file() {
                        results.push(Ok(path.to_string()));
                    }
                }
                Err(e) => {
                    results.push(Err(From::from(format!("{}: {}", path, e))))
                }
            },
        }
    }

    results
}
```

- `flatten` 作用是打平，可以看下面的例子：

Rust

```rust
let data = vec![vec![1, 2, 3, 4], vec![5, 6]];
let flattened = data.into_iter().flatten().collect::<Vec<u8>>();
assert_eq!(flattened, &[1, 2, 3, 4, 5, 6]);
```

- 匹配行

Rust

```rust
fn find_lines<T: BufRead>(
    mut file: T,
    pattern: &Regex,
    invert_match: bool,
) -> MyResult<Vec<String>> {
    let mut matches = vec![];
    let mut line = String::new();

    loop {
        let bytes = file.read_line(&mut line)?;
        if bytes == 0 {
            break;
        }
        if pattern.is_match(&line) ^ invert_match {
            matches.push(mem::take(&mut line));
        }
        line.clear();
    }

    Ok(matches)
}
```

- `^`：异或
- Use `std::mem::take` to take ownership of the line. I could have used `clone` to copy the string and add it to the matches, but take avoids an unnecessary copy.

Rust

```rust
use std::mem;

let mut v: Vec<i32> = vec![1, 2];

let old_v = mem::take(&mut v);
assert_eq!(vec![1, 2], old_v);
assert!(v.is_empty());
```