src\main.rs

use std::{fs, path::PathBuf};

use clap::Parser;

use docx\_rs::{Docx, DocxError, IndentLevel, NumberingId, Paragraph, Run};

use glob::glob;

//TODO Doc model

//TODO Doc generation

#[derive(Parser, Debug)]

#[command(author, version, about, long\_about = None, arg\_required\_else\_help = true)]

struct Args {

/// Расширения файлов. Пример code\_report\_rs rs js

file\_extensions: Vec<String>,

///Использовать расширения файлов по умолчанию rs, go cs toml java html

///Вы так же можете добавить свои расширения файлов[FILE\_EXTENSIONS]...

///Пример: code\_report\_rs --use\_defaut\_values shtml css

#[arg(short, long, default\_value\_t = false)]

use\_defaut\_values: bool,

}

fn main() {

let args = Args::parse();

let mut file\_extensions = vec![

String::from("rs"),

String::from("go"),

String::from("cs"),

String::from("toml"),

String::from("java"),

String::from("html"),

];

if !args.file\_extensions.is\_empty() && !args.use\_defaut\_values {

file\_extensions = args.file\_extensions;

} else if args.use\_defaut\_values {

for extension in args.file\_extensions {

file\_extensions.push(extension);

}

file\_extensions.dedup();

}

let mut doc = Docx::new();

for file\_extension in file\_extensions {

for entry in glob(&("./\*\*/\*\*/\*\*/\*\*/\*\*/\*\*/\*\*/\*.".to\_owned() + &file\_extension))

.expect("Failed to read glob pattern")

{

match entry {

Ok(path) => {

println!("{:?}", path.to\_str().unwrap());

println!("{:?}", doc.gen\_body(path));

}

Err(e) => println!("{:?}", e),

}

}

}

let path = std::path::Path::new("./numbering.docx");

let file = fs::File::create(path).unwrap\_or(fs::File::open(path).unwrap());

doc.to\_owned().build().pack(file).unwrap();

}

trait GenFile {

fn gen\_body(&mut self, input\_path: PathBuf) -> Result<(), DocxError>;

}

impl GenFile for Docx {

fn gen\_body(&mut self, input\_path: PathBuf) -> Result<(), DocxError> {

\*self = self.to\_owned().add\_paragraph(

Paragraph::new()

.add\_run(

Run::new()

.add\_text(input\_path.as\_path().to\_str().unwrap())

.size(16 \* 2),

)

.numbering(NumberingId::new(2), IndentLevel::new(0))

.size(16 \* 2),

);

let lines: Vec<String> = fs::read\_to\_string(input\_path)

.unwrap()

.split("\n")

.map(str::to\_string)

.collect();

for line in lines {

\*self = self

.to\_owned()

.add\_paragraph(Paragraph::new().add\_run(Run::new().add\_text(line)));

}

Ok(())

}

}

// fn gen\_file(input\_path: PathBuf, doc: &mut Docx) -> Result<(), DocxError> {

// \*doc = doc.to\_owned().add\_paragraph(

// Paragraph::new()

// .add\_run(

// Run::new()

// .add\_text(input\_path.as\_path().to\_str().unwrap())

// .size(16 \* 2),

// )

// .numbering(NumberingId::new(2), IndentLevel::new(0))

// .size(16 \* 2),

// );

// let lines: Vec<String> = fs::read\_to\_string(input\_path)

// .unwrap()

// .split("\n")

// .map(str::to\_string)

// .collect();

// for line in lines {

// \*doc = doc

// .to\_owned()

// .add\_paragraph(Paragraph::new().add\_run(Run::new().add\_text(line)));

// }

// Ok(())

// }

target\debug\build\thiserror-56ceb42a81137db3\out\probe.rs

#![feature(provide\_any)]

use std::any::{Demand, Provider};

fn \_f<'a, P: Provider>(p: &'a P, demand: &mut Demand<'a>) {

p.provide(demand);

}

Cargo.toml

[package]

name = "code\_report\_rs"

version = "0.1.0"

edition = "2021"

authors = ["Dan Hludencov <https://github.com/Chu-4hun>"]

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]

clap = { version = "4.0.32", features = ["cargo", "derive"] }

docx-rs = "0.4.6"

glob = "0.3.1"

rand = "0.8.5"