Rust de abajo para arriba

Slides y ejemplos en:
https://github.com/seppo0010/rust-abajo-arriba

@seppo0010 / @seppo0011

Interrumpan si tienen preguntas,
si algo no se entiende,
o ante cualquier comentario que quieran hacer.

Posta.

¿Por qué Rust?

- Bajo nivel (performance comparable con C++)
- Seguro (sin race conditions, segfaults, etc)
- Seguro II (te obliga a estructurar tu código, te empuja a manejar errores)
- Sin Garbage Collector
- Lindo

mov64 r1, 0x2172656a
@badboy_

Rust made me a better C programmer.

20:47 - 13 ene. 2017

Backend

Servicios y Async I/O: https://github.com/tokio-rs

Parser: https://github.com/Geal/nom

Web frameworks: http://rocket.rs/ http://ironframework.io/

Are we web yet? http://www.arewewebyet.org/

Microservicio que recibe un string y lo da vuelta

hola => aloh

Hello world => dlrow olleH

Un protocolo simple: leer de a una línea y responder de a un línea

Microservicio - revstring - Cargo.toml

Cargo.toml

```
[package]
name = "example-01-microservice-revstring"
version = "0.1.0"
authors = ["root"]

[dependencies]
tokio-line = { git = "https://github.com/tokio-rs/tokio-line" }
service-fn = { git = "https://github.com/tokio-rs/service-fn" }
```

Microservicio - revstring - src/main.rs

src/main.rs

```
extern crate tokio_line as line;
extern crate service_fn;

fn main() {
	let addr = "127.0.0.1:12345".parse().unwrap();
	line::service::serve(
		addr,
		|| {
			Ok(service_fn::service_fn(|msg: String| {
				Ok(msg.chars().rev().collect::<String>())
			}))
	});
}
```

Microservicio - revstring - example

```
$ cargo build
...
    Finished debug [unoptimized + debuginfo] target(s) in 18.88 secs
$ ./target/debug/example-01-microservice-revstring &
$ echo hello world |nc 127.0.0.1 12345
dlrow olleh
$ echo dabale arroz a la zorra el @abad |nc 127.0.0.1 12345
daba@ le arroz al a zorra elabad
```

Microservicio que hace bcrypt por nosotros

Guarda “costo” (complejidad para hashear)

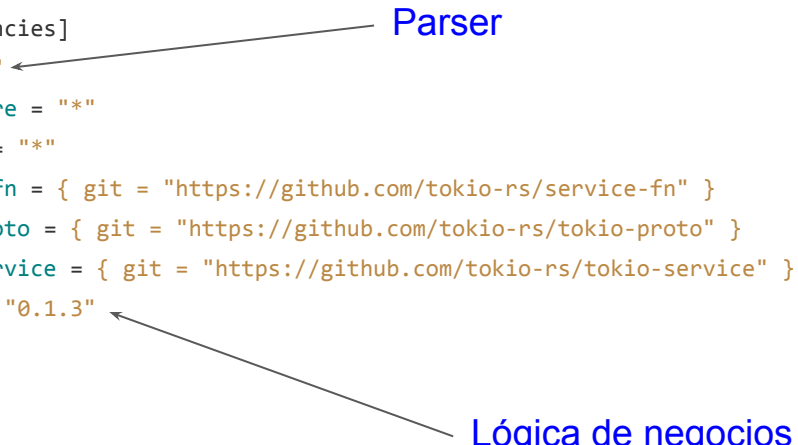
Un método para crear un hash

Un método para verificar un hash

Microservicio - bcrypt - Cargo.toml

```
[package]
name = "example-02-microservice-baas"
version = "0.1.0"
authors = ["root"]

[dependencies]
nom = "*"
tokio-core = "*"
futures = "*"
service-fn = { git = "https://github.com/tokio-rs/service-fn" }
tokio-proto = { git = "https://github.com/tokio-rs/tokio-proto" }
tokio-service = { git = "https://github.com/tokio-rs/tokio-service" }
bcrypt = "0.1.3"
```



Microservicio - bcrypt - src/protocol.rs

```
#[derive(Debug, PartialEq)]
pub enum BaasProtocol {
     SetCost(u32),
     Hash(String),
     Verify(String, String),
}
```

Microservicio - bcrypt - src/protocol.rs

```
named!(cost<BaasProtocol>, do_parse!(
     tag!("cost") >>
     cost: digit >>
     newline >>
     (BaasProtocol::SetCost(str::from_utf8(cost).unwrap().parse().unwrap()))
));
```

cost12\n

Microservicio - bcrypt - src/protocol.rs

```
named!(hash<BaasProtocol>, do_parse!(
     tag!("hash") >>
     s: alphanumeric >>
     newline >>
     (BaasProtocol::Hash(String::from_utf8(s.to_vec()).unwrap()))
));
```

hashhunter2\n

Microservicio - bcrypt - src/protocol.rs

```
named!(verify<BaasProtocol>, do_parse!(
     tag!("verify") >>
     hash: alphanumeric >>
     space >>
     verify: not_line_ending >>
     newline >>
     (BaasProtocol::Verify(
          String::from_utf8(hash.to_vec()).unwrap(),
          String::from_utf8(verify.to_vec()).unwrap()
     ))
));
```

verifyhunter2 $myhash$lolololololol\n

Microservicio - bcrypt - src/protocol.rs

```
named!(parse<BaasProtocol>, alt!(cost | hash | verify));
```

cost12\n
hashhunter2\n
verifyhunter2 $myhash$lolololololol\n

Microservicio - bcrypt - src/transport.rs

```
let parsed = match BaasProtocol::parse(&*self.read_buffer) {
        nom::IResult::Done(read, res) => Some((self.read_buffer.len() - read.len() - 1, res)),
        nom::IResult::Incomplete(_) => None,
        nom::IResult::Error(e) => return Err(io::Error::new(io::ErrorKind::Other, e)),
};
if let Some((n, res)) = parsed {
        let tail = self.read_buffer.split_off(n+1);
        let mut line = mem::replace(&mut self.read_buffer, tail);
        line.truncate(n);
        return Ok(Async::Ready(Some(res)));
}
match self.inner.read_to_end(&mut self.read_buffer) {
        Ok(0) => return Ok(Async::Ready(None)),
        Ok(_) => {},
        Err(e) => {
                if e.kind() == io::ErrorKind::WouldBlock {
                        return Ok(Async::NotReady);
                }
                return Err(e);
        }
}
```

Hay un comando válido, todavía no hay un comando, o hay fruta

Si hay un comando, sacarlo del buffer, y devolverlo

Seguir leyendo

Microservicio - bcrypt - src/main.rs

```
let addr = "127.0.0.1:12345".parse().unwrap();
let size = Arc::new(Mutex::new(13));
service::serve(addr, move || {
	let size = size.clone();
	Ok(service_fn::service_fn(move |msg: protocol::BaasProtocol| {
		Ok(match msg {
			protocol::BaasProtocol::SetCost(s) => {
				let mut data = size.lock().unwrap();
				*data = s;
				"OK".to_owned()
			},
			protocol::BaasProtocol::Hash(s) => bcrypt::hash(&*s, *size.lock().unwrap()).unwrap(),
			protocol::BaasProtocol::Verify(s, h) => {
				match bcrypt::verify(&*s, &*h).unwrap() {
					true => "valid",
					false => "invalid"
				}.to_owned()
			},
		});
	}))
});
```

Microservicio - bcrypt

<boilerplate>

Microservicio - bcrypt - example

```
$ cargo run &
...
     Finished debug [unoptimized + debuginfo] target(s) in 0.0 secs
     Running `target/debug/example-02-microservice-baas`
$ echo "cost7" | nc 127.0.0.1 12345
OK
$ echo "hashhello" | nc 127.0.0.1 12345
$2y$07$3F9At7aoqqiMCpuxXpXEkelbWjw950P.0G83fxKcqpaZWY671cn1u
$ echo 'verifyhello $2y$07$3F9At7aoqqiMCpuxXpXEkelbWjw950P.0G83fxKcqpaZWY671cn1u' | nc 127.0.0.1 12345
valid
$ echo 'verifyworld $2y$07$3F9At7aoqqiMCpuxXpXEkelbWjw950P.0G83fxKcqpaZWY671cn1u' | nc 127.0.0.1 12345
invalid
```

Servidor HTTP usando tokio

Hay otros frameworks específicos para HTTP que seguro son más adecuados si es lo único que van a hacer.

Pero como ya venimos viendo tokio…

Vamos a ver tres iteraciones: un hello world, un servidor que sirve el pwd via http (como SimpleHTTPServer de python) y uno que además escribe

Servidor http - Cargo.toml

```
[package]
name = "example-03-webserver"
version = "0.1.0"
authors = ["root"]

[dependencies]
futures = "0.1.1"
tokio-proto = { git = "https://github.com/tokio-rs/tokio-proto" }
tokio-service = { git = "https://github.com/tokio-rs/tokio-service" }
tokio-minihttp = { git = "https://github.com/tokio-rs/tokio-minihttp" }
```

Servidor http - src/main.rs

```
struct HelloWorld;

impl Service for HelloWorld {
    type Request = Request;
    type Response = Response;
    type Error = io::Error;
    type Future = future::Ok<Response, io::Error>;

    fn call(&mut self, request: Request) -> Self::Future {
        let mut resp = Response::new();
        resp.body("Hello, world!");
        future::ok(resp)
    }
}

fn main() {
    let addr = "0.0.0.0:8080".parse().unwrap();
    TcpServer::new(Http, addr)
        .serve(|| Ok(HelloWorld));
}
```

Servidor http - example

```
$ cargo run &
    Finished debug [unoptimized + debuginfo] target(s) in 1.99 secs
     Running `target/debug/example-03-webserver`

$ curl -v localhost:8080
*   Trying 127.0.0.1...
* Connected to localhost (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> Host: localhost:8080
> User-Agent: curl/7.47.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: Example
< Content-Length: 13
< Date: Tue, 03 Jan 2017 14:30:40 GMT
<
* Connection #0 to host localhost left intact
Hello, world!
```

Servidor http 2 - src/main.rs

```
fn call(&mut self, request: Request) -> Self::Future {
    let mut resp = Response::new();
    match File::open(&request.path()[1..]) {
        Ok(ref mut f) => {
            let mut s = String::new();
            if let Err(_e) = f.read_to_string(&mut s) {
                resp.status_code(500, "Internal Server Error")
            } else {
               resp.body(&*s)
            }
        },
        Err(ref e) =>  match e.kind() {
            io::ErrorKind::NotFound => resp.status_code(404, "Not Found"),
            _ => resp.status_code(500, "Internal Server Error"),
        }
    };
    future::ok(resp)
}
```

Servidor http 2 - src/main.rs

```
fn call(&mut self, request: Request) -> Self::Future {
    let mut resp = Response::new();
    match File::open(&request.path()[1..]) {
        Ok(ref mut f) => {
            let mut s = String::new();
            if let Err(_e) = f.read_to_string(&mut s) {
                resp.status_code(500, "Internal Server Error")
            } else {
               resp.body(&*s)
            }
        },
        Err(ref e) =>  match e.kind() {
            io::ErrorKind::NotFound => resp.status_code(404, "Not Found"),
            _ => resp.status_code(500, "Internal Server Error"),
        }
    };
    future::ok(resp)
}
```

TODO: fs async

TODO: no copiar memoria

Servidor http 2 - example

```
$ cargo run
    Finished debug [unoptimized + debuginfo] target(s) in 0.0 secs
     Running `target/debug/example-03-webserver`

$ curl localhost:8080/Cargo.toml
[package]
name = "example-03-webserver"
version = "0.1.0"
authors = ["root"]

[dependencies]
futures = "0.1.1"
tokio-proto = { git = "https://github.com/tokio-rs/tokio-proto" }
tokio-service = { git = "https://github.com/tokio-rs/tokio-service" }
tokio-minihttp = { git = "https://github.com/tokio-rs/tokio-minihttp" }
$ curl localhost:8080/404
$
```

Servidor http 3 - src/main.rs

```
    fn call(&mut self, request: Request) -> Self::Future {
        let mut resp = Response::new();
        let r = match request.method() {
            "POST" => self.write(request).map(|_|
"OK".to_owned()),
            _ => self.read(request),
        };
        match r {
            Ok(e) => { resp.body(&*e); },
            Err((status, message)) => { resp.status_code(status,
message); },
        }
        future::ok(resp)
    }
```

```
    fn write(&mut self, request: Request) -> Result<(), (u32,
&str)> {
        let p = request.path();
        let path = Path::new(&p[1..]);
        if let Some(parent) = path.parent() {
            if parent.is_file() {
                return Err((400, "Parent path is a file"))
            }
            if ! parent.is_dir() {
                if let Err(_) = create_dir_all(parent) {
                    return Err((500, "Failed to create
directory"))
                }
            }
        }
        File::create(path).and_then(|mut f| {
            f.write(request.body().as_slice()).map(|_| ())
        }).map_err(|_| (500, "Internal Server Error"))
    }
```

Servidor http 3 - example

```
$ curl localhost:8080/test
$ curl localhost:8080/test  --data "hello world"
OK
$ localhost:8080/test
hello world
$ cat test
hello world
$ curl -v 127.0.0.1:8080/test/test --data "test"
(...)
< HTTP/1.1 400 Parent path is a file
(...)
```

Frontend

Compilar Rust a Javascript: https://users.rust-lang.org/t/compiling-to-the-web-with-rust-and-emscripten/7627

Rust webplatform: https://github.com/tcr/rust-webplatform

Quasar: https://github.com/anowell/quasar

Hello emscripten

```
$ echo 'fn main() { println!("Hello, Emscripten!"); }' > hello.rs
$ node hello.js
Hello, Emscripten!
```

```
$ ls -lh hello.js
-rw-r--r-- 1 root root 1.1M Jan  3 18:31 hello.js
$ uglify -s hello.js -o hello.min.js
(...)
Done, without errors.
$ ls -hl hello.min.js
-rw-r--r-- 1 root root 389K Jan  3 18:44 hello.min.js
$ gzip hello.min.js
$ ls -lh hello.min.js.gz
-rw-r--r-- 1 root root 118K Jan  3 18:44 hello.min.js.gz
```

Sí, soy root, porque estoy dentro de un container y no me creé un usuario ¯_(ツ)_/¯

Hello web - Cargo.toml

```
[package]
name = "example-04-webclient"
version = "0.1.0"
authors = ["root"]

[dependencies]
webplatform = { "git" = "https://github.com/tcr/rust-webplatform" }
```

Hello web - src/main.rs

```
extern crate webplatform;

fn main() {
    let document = webplatform::init();
    let body = document.element_query("body").unwrap();
    body.html_set("<h1>HELLO FROM RUST</h1> <button>CLICK ME</button>");
    let button = document.element_query("button").unwrap();
    button.on("click", |_| webplatform::alert("WITNESS ME"));
}
```

index.html

```
<body><script src="target/asmjs-unknown-emscripten/debug/example-04-webclient.js"></script></body>
```

Hello web - example

```
$ cargo build --target=asmjs-unknown-emscripten
    Updating git repository `https://github.com/tcr/rust-webplatform`
    Updating registry `https://github.com/rust-lang/crates.io-index`
   Compiling libc v0.2.18
   Compiling webplatform v0.4.2 (https://github.com/tcr/rust-webplatform#ebbc86bf)
   Compiling example-04-webclient v0.1.0 (file:///root/rust-abajo-arriba/example-04-webclient)
    Finished debug [unoptimized + debuginfo] target(s) in 8.19 secs
```

SPA - Cargo.toml

```
[package]
name = "example-05-fileeditor"
version = "0.1.0"
authors = ["root"]

[dependencies]
webplatform = { "git" = "https://github.com/seppo0010/rust-webplatform", rev = "unstable" }
```

SPA - src/main.rs - 0/3

WARNING

WARNING

El siguiente código es feo, muy feo, y no representa buenas prácticas ni el espíritu de Rust.

Por favor, entiéndalo como una prueba de concepto y nada más.

WARNING

WARNING

SPA - src/main.rs - 1/3

```
let document_rc = Rc::new(webplatform::init());
let document = document_rc.clone();
let body = document.element_query("body").unwrap();
body.html_set("
    <select>
        <option></option>
        <option value=\"Cargo.toml\">Cargo.toml</option>
        <option value=\"Cargo.lock\">Cargo.lock</option>
        <option value=\"src/main.rs\">src/main.rs</option>
    </select>
    <div>
        <h1>Editing <span></span></h1>
        <textarea cols=\"80\" rows=\"30\"></textarea>
        <button>Save</button>
    </div>
");
let select_rc = Rc::new(document.element_query("select").unwrap());
let div_rc = Rc::new(document.element_query("div").unwrap());
let textarea_rc = Rc::new(document.element_query("textarea").unwrap());
div_rc.clone().style_set_str("display", "none");
let filename_rc = Rc::new(RefCell::new(None));
```

SPA - src/main.rs - 2/3

```
let span = document.element_query("span").unwrap();
let textarea = textarea_rc.clone();
let filename = filename_rc.clone();
let select = select_rc.clone();
let div = div_rc.clone();
select_rc.clone().on("change", move |_| {
        let f = select.prop_get_str("value");
        span.html_set(&*f);
        div.style_set_str("display", if f == "" { "none" } else { "block" });
        textarea.style_set_str("display", "none");
        if f != "" {
            let textarea = textarea.clone();
            webplatform::ajax_get(&*document, &*f, move |xhr| {
                textarea.style_set_str("display", "block");
                textarea.prop_set_str("value", &*xhr.response_text().unwrap());
            });
            *filename.borrow_mut() = Some(f);
        } else {
            *filename.borrow_mut() = None;
        }
});
```

SPA - src/main.rs - 3/3

```
let textarea = textarea_rc.clone();
let document = document_rc.clone();
let filename = filename_rc.clone();
let select = select_rc.clone();
let div = div_rc.clone();
let button = document.element_query("button").unwrap();
button.on("click", move |_| {
    let value = textarea.prop_get_str("value");
    if let Some(ref s) = *filename.borrow() {
        select.prop_set_str("value", "");
        select.prop_set_str("disabled", "");
        let div = div.clone();
        let select = select.clone();
        webplatform::ajax_post(&*document, s, Some(&*value), move |_| {
            div.style_set_str("display", "none");
            select.prop_del("disabled");
        });
    }
});

webplatform::spin();
```

SPA - live demo

(¿alguien me toma una apuesta de 100 pe a que no funciona bien?)

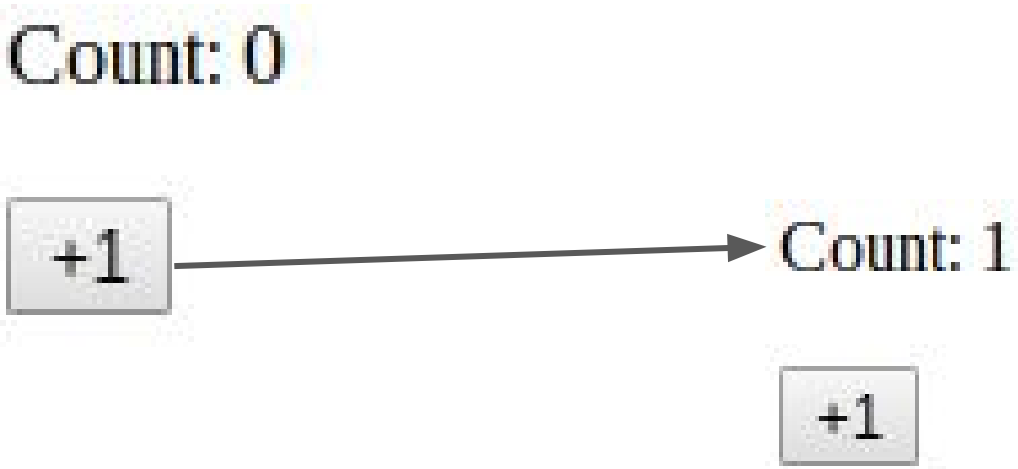
quasar - un framework web estilo react

```
use quasar::*;

struct CounterData {
   count: u32,
}
impl Renderable for CounterData {
   fn render(&self, _props: Properties) -> String {
       (html! { p { "Count: " (self.count) } button { "+1" } }).into_string()
   }
}

pub fn init(app: &QuasarApp) {
   let component = CounterData { count: 0 };
   app.bind("#counter", component)
       .on(EventType::Click, |mut evt| {
           evt.binding.data_mut().count += 1;
       });
}
```

quasar - un framework web estilo react



Otras ideas

Procesar datos y pasar mensajes via WebWorkers [wasm]

Desde Rust usar react, compilar a `React.createElement(...)`

Una base de datos para el browser, offline-first, sincronizandose con un backend, sirviendo queries localmente y notificando de cambios [wasm]

Gracias

Slides y ejemplos en:
https://github.com/seppo0010/rust-abajo-arriba

@seppo0010 / @seppo0011