

Rust in the Web

Eric Kunze



1 Historie

2 Warum Rust?

3 Was macht Rust sicher?

4 Are we Web yet?

5 Iron

6 Demo



- 2006 persönliches Projekt von Graydon Hoare
- ab 2009 Projekt bei Mozilla
- 15. Mai 2015 Veröffentlichung der Version 1.0



- 2006 persönliches Projekt von Graydon Hoare
- ab 2009 Projekt bei Mozilla
- 15. Mai 2015 Veröffentlichung der Version 1.0

- Entwicklung einer neuen Browserenging → Servo



- 2006 persönliches Projekt von Graydon Hoare
- ab 2009 Projekt bei Mozilla
- 15. Mai 2015 Veröffentlichung der Version 1.0

- Entwicklung einer neuen Browserenging → Servo
 - Warum nicht C++ oder Java?



Rust: Safe System Programming



Rust: Safe System Programming

- Performance



Rust: Safe System Programming

- Performance
- Kontrolle
 - minmale Runtime
 - keine Garbage Collection



Rust: Safe System Programming

- Performance
- Kontrolle
 - minmale Runtime
 - keine Garbage Collection
- Sicherheit



Rust: Safe System Programming

- Performance
- Kontrolle
 - minmale Runtime
 - keine Garbage Collection
- Sicherheit
- Features von höheren und funktionalen Programmiersprachen
 - Pattern Matching
 - Traits
 - Closures





■ Ownership

- jede Ressource hat zu einem Zeitpunkt **genau einen** Besitzer
- Ressourcen können den Besitzer wechseln



■ Ownership

- jede Ressource hat zu einem Zeitpunkt **genau einen** Besitzer
- Ressourcen können den Besitzer wechseln

```
1 fn foo() {  
2     let mut y: Vec<i32>  
3     = Vec::new();  
4  
5     y.push(4);  
6     bar(y);  
7     y.push(5); // Compiler Error  
8 }
```

```
1 fn bar(x: Vec<i32>) {  
2     ...  
3 }
```



- Borrowing
 - Ownership kann verliehen werden



- Borrowing
 - Ownership kann verliehen werden
- shared borrow
 - die Referenz ist **unveränderlich**
 - die Ressource kann mehrfach verliehen werden



■ Borrowing

- Ownership kann verliehen werden

■ shared borrow

- die Referenz ist **unveränderlich**
- die Ressource kann mehrfach verliehen werden

```
1 fn foo() {  
2     let mut y: Vec<i32>  
3     = Vec::new();  
4  
5     y.push(4);  
6     bar(&y);  
7     y.push(5); // Ok  
8 }
```

```
1 fn bar(x: &Vec<i32>){  
2     ...  
3     b = a + x[0]; // Ok  
4  
5     x.push(1); // Compiler Error  
6  
7 }
```




- mutable borrow
 - zu einem Zeitpunkt darf **nur eine** Referenz existieren
 - der Besitzer dieser Referenz kann die Ressource **verändern**



■ mutable borrow

- zu einem Zeitpunkt darf **nur eine** Referenz existieren
- der Besitzer dieser Referenz kann die Ressource **verändern**

```
1 fn foo() {  
2     let mut y  
3     = vec![1,2,3,4,5];  
4     let mut x: Vec<i32>  
5     = Vec::new();  
6  
7     bar(&y, &mut x); // Ok  
8     bar(&y, &mut y);  
9     // Compiler Error  
10 }
```

```
1 fn bar(y: &Vec<i32>,  
2       x: &mut Vec<i32>){  
3  
4     for v in y{  
5         x.push(*v);  
6     }  
7 }
```





Vorteile

- Sicherheit und Geschwindigkeit
- Tools
 - Cargo
 - rustdoc
- Open Source Projekte
crates.io



Vorteile

- Sicherheit und Geschwindigkeit
- Tools
 - Cargo
 - rustdoc
- Open Source Projekte
crates.io

Nachteile

- sehr junge Sprache
- häufige Änderungen
- Dokumentation
- (Kompilierzeit)



■ HTTP-Server

- Hyper
- tiny-http

■ HTTP client

- Hyper
- curl-rust

■ Database drivers

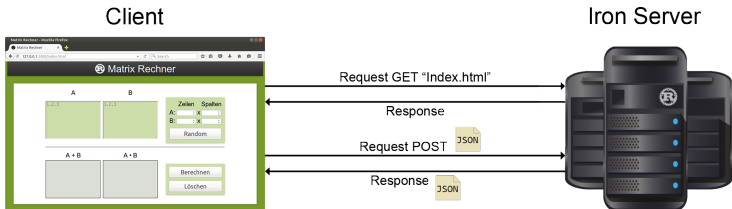
- rust-postgres
- rusqlite
- redis-rs

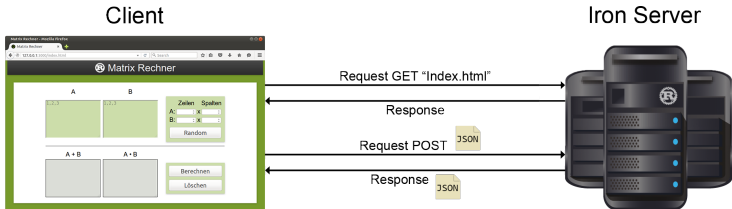
■ Frameworks

- Iron
- rustful
- Nickel



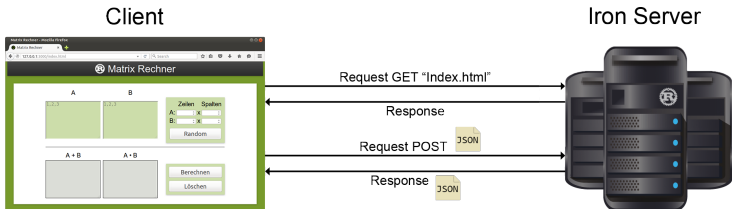
- Webframework
- basiert auf Hyper
- hochgradig Nebenläufig
- Basisframework ist leicht erweiterbar
- bietet Infrastruktur, um das Framework an individuelle Bedürfnisse anzupassen





Request Body

```
1 {  
2   "mat_a": {  
3     "rows": 3,  
4     "cols": 3,  
5     "elem": ["1", "2", ...] },  
6   "mat_b": { ... }  
7 }
```



Request Body

```
1 {  
2   "mat_a": {  
3     "rows": 3,  
4     "cols": 3,  
5     "elem": ["1", "2", ...] },  
6   "mat_b": { ... }  
7 }
```

Response Body

```
1 {  
2   "message": "Fehler",  
3   "mat_a": { ... },  
4   "mat_b": { ... }  
5 }
```

