

# Rust 世界の二つのモナド

Rust でも **do** 式をして  
プログラムを直感的に記述する件について

konn  
関数型まつり 2025  
2025-06-14



Slides available: <https://u.konn-san.com/qdo>



# 自己紹介

- ・ <sup>いし い ひろ み</sup>石井 大海 / konn (Twitter: @mr\_konn)
- ・ 現職：株式会社 Jij ソフトウェア開発チーム（2024/11～）
  - ・ **Rust** で数理最適化の埋込ドメイン特化言語 (EDSL) を開発しています
    - ・ **スポンサーブース出展中！** 興味のある方もない方もぜひ！
- ・ 前職では **Haskell** で数値計算向け **EDSL** を開発
  - ・ EDSL 開発に縁があるね
- ・ 今日の話題は EDSL の設計に欠かせない **do 記法**のお話です



# 本題



# Rust 世界の二つのモナド

Rust でも `do` 式をして

プログラムを直感的に記述する件について



# Rust 世界の二つのモナド

Rust でも **do 式** をして

プログラムを直感的に記述する件について



🤔 モナドとか do 式とかって

Haskell が使ってるやつ？



 **Rust** と関係あるの？

**二つ**ってなに？



お答えしましょう！



# 今回の内容

## 1. `qualified_do`: Rust でも (Qualified)do する件について

- ・ Haskell の **QualifiedDo 記法**を移植した `qualified_do` クレートを紹介
- ・ **Rust でも do 式は便利そ〜**と思ってもらう

## 2. do 式でわかる かもしれない！モナド

- ・ 「**do式が使える構造**」として**モナド**を捉えなおす
- ・ **Functor** や **Applicative** など「**do式の機能をどこまで使えるか**」という観点で説明
- ・ **GATs** を使って **Rust** で Functor / Applicative / Monad を定式化するトリックも紹介

## 3. Rust 世界の二つのモナド ～資源は大切に～

- ・ Rust や Linear Haskell など**資源の使い方に厳しい世界**では、**モナド階層が二つに分岐**することを紹介
- ・ Rust と Linear Haskell では**型システムの違い**によって**何がモナドになれるかに差異**が生まれることを紹介



# おしながき

1. `qualified_do`: Rust でも (Qualified)do する件について
2. `do` 式でわかる かもしれない ! モナド
3. Rust 世界の二つのモナド ～資源は大切に～



`qualified_do:`

Rust でも (Qualified)do する件について



qualified\_do クレート



# おしながき

1. `qualified_do`: Rust でも (Qualified)do する件について



do 式でわかるかもしれない！モナド



# おしながき

## 2. do 式でわかるかもしれない！モナド



Rust 世界の二つのモナド

～資源は大切に～



# おしながき

## 3. Rust 世界の二つのモナド ～資源は大切に～



まとめ



# 今回の内容

- ・ **モナド = 無制限のdo式を使える構造**

- ・ Functor = `do { x ← mx ; pure (f x) }` の形のdo 式が使える
- ・ Applicative = `do { x1 ← mx1 ; ... ; xn ← mxn ; pure (f x1 ... xn) }` の形のdo式 (mxi: 互いに独立) が使える
- ・ Rust でも Generalised Associated Types (GATs) を使って「実装」にタグづけするトリックで実装可能
- ・ Rust や Linear Haskell など線型・アファイン型のある世界では **Monad の階層が二つに分裂**する
  - ・ **Data Functor**: 渡された関数を**複数回**呼び出し得る
  - ・ **Control Functor**: 渡された関数は **(Rust ならば高々) 一回しか**使われない
    - ・ Linear Haskell では Maybe (Option) は Control Monad になり得ないが、Rust では **(Affine) Control Monad** になる！
- ・ **QualifiedDo 記法**：複数の「モナドっぽい」クラスに do 式を使えるようにする Haskell の言語拡張
  - ・ ApplicativeDo 言語拡張などと組み合わせると Functor, Applicative っぽいものたちも切り替えてつあかえる
  - ・ QualifiedDo + ApplicativeDo +  $\alpha$  を Rust で使うための proc macro qdo! を提供する qualified\_do クレート創りました



# 参考文献

- ・ Linear Haskell の論文
- ・ Shake before Building の論文
- ・ Tale of Two Functors の記事
- ・ ApplicativeDo や QualifiedDo のリファレンス
- ・ わたしの Zenn



御清聴

ありがとう

ございました



# Any Questions?

- ・ **モナド = 無制限のdo式を使える構造**

- ・ Functor = `do { x ← mx ; pure (f x) }` の形のdo 式が使える
- ・ Applicative = `do { x1 ← mx1 ; ... ; xn ← mxn ; pure (f x1 ... xn) }` の形のdo式 (mxi: 互いに独立) が使える
- ・ Rust でも Generalised Associated Types (GATs) を使って「実装」にタグづけするトリックで実装可能
- ・ Rust や Linear Haskell など線型・アファイン型のある世界では **Monad の階層が二つに分裂**する
  - ・ **Data Functor**: 渡された関数を**複数回**呼び出し得る
  - ・ **Control Functor**: 渡された関数は **(Rust ならば高々) 一回しか**使われない
    - ・ Linear Haskell では Maybe (Option) は Control Monad になり得ないが、Rust では **(Affine) Control Monad** になる！
- ・ **QualifiedDo 記法**：複数の「モナドっぽい」クラスに do 式を使えるようにする Haskell の言語拡張
  - ・ ApplicativeDo 言語拡張などと組み合わせると Functor, Applicative っぽいものたちも切り替えてつあかえる
  - ・ QualifiedDo + ApplicativeDo +  $\alpha$  を Rust で使うための proc macro qdo! を提供する qualified\_do クレート創りました