

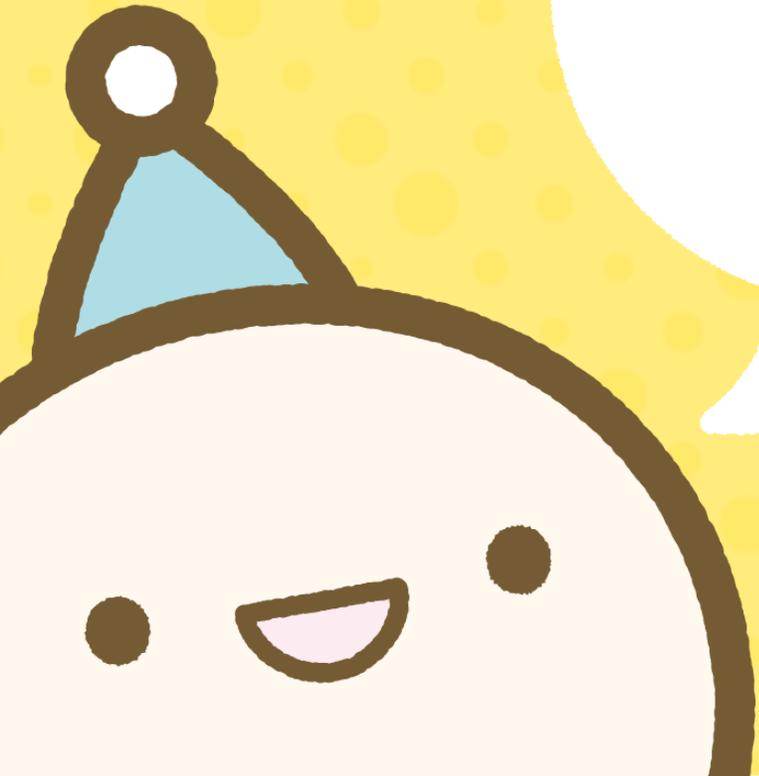
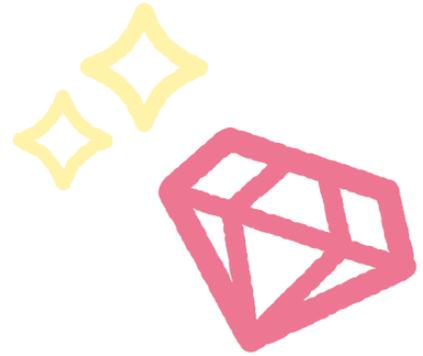
# リアルタイムアドバイスツールRuboSenseiによる学習体験の向上

- RubyWorld Conference 2023
- 五十嵐邦明
- ガーネットテック373株式会社 <https://garnettech373.com>

# 自己紹介

- 五十嵐邦明
- ガーネットテック373株式会社 <https://garnettech373.com>
- 技術顧問(育成、レガシー対応、コード健康診断などなど)のお仕事募集中
- 著書: 「ゼロからわかるRuby超入門」 「RubyとRailsの学習ガイド」 「Railsの教科書」  
「パーフェクトRails」 「Railsの練習帳」
- プログラミングスクール「フィヨルドブートキャンプ」顧問
- 島根県Ruby合宿 アシスタント講師
- 謝辞: この講演を応援して送り出してくれた妻と子に感謝します

Rubbo  
Sensei



# リアルタイムに支援するコーディング環境RuboSensei

- 私がつくっている開発体験向上ツールRuboSenseiについて今日は話します

# リアルタイムに支援するコーディング環境RuboSensei

- 背景: まつもとゆきひろさんが提示するRubyの向かうべき方向の1つ
  - **Rubyの開発体験の向上**
- 初学者向けにエディタ上でリアルタイムに支援するコーディング環境RuboSensei
  - <https://github.com/igaiga/rubocop-sensei>
- コードを書いているときにより良いRubyコードの書き方をリアルタイムにアドバイス・置換できるコーディング環境
- コードの意味、より良い書き方を教えてくれる
- わかりやすい文章で正しい内容を伝える
- AIはつかっていない

## 本講演で話すこと

- エディタ上でリアルタイムに支援するコーディング環境RuboSenseiについて
- Rubyでコードを書くときのAIなどのソフトウェアによる支援の現状をまとめ、比較
  - GitHub Copilot, GitHub Copilot chat, ChatGPT
- 今後の方向性の考察と自由研究
- 講演後、聴講した方々が各種支援を受けてコーディング可能になることを目標とします

# RuboSensei デモ

- 動画ファイル: <https://github.com/igaiga/rubocop-sensei/blob/v0.1.6/demo.gif>

# リアルタイムに支援するコーディング環境RuboSensei

- 初学者向けにエディタ上でリアルタイムに支援するコーディング環境RuboSensei
  - <https://github.com/igaiga/rubocop-sensei>
- コードの意味、より良い書き方を教えてくれる
- RuboSenseiで役立ちたい場面
  - より良い書き方の提示
  - 初学者が初見で読みづらいコードの解説
  - 新しめの文法の解説

# RuboSenseiで実装したアドバイス

- elsifはcaseで書き換えよう
- unless else はやめてifに書き換えよう
- このeachはmapで書けるかも
- method(&:method) で書けるかも
- method(&:method) の説明
- ナンバーパラメータの説明
- パターンマッチcase inの説明
- 1行パターンマッチinの説明
- 1行パターンマッチ=>の説明
- ぼっち演算子の説明
- ||= の説明
- !!variable の説明

# elsifはcaseで書き換えよう

```
1  if true
2    x = 1
3  elsif false
```

`elsif`は`case`で書き換え可能です。

N個の分岐から1つ選ぶ「N者択一」のときはcaseをつかうと読みやすくなります。

たとえば、elsifをつかった次のようなコードがあるとき、

```
if x == 1
elsif y == 2
else
end
```

caseをつかうと次のように書き換えられます。

```
case
when x == 1
```

## このeachメソッドはmapメソッドで書けるかも

この`each`メソッドは`map`メソッドで置き換えられるかもしれません。  
rubocop(Lecture/PreferMap)

問題の表示 (⌘F8)    クイック フィックス... (⌘.)

```
[1,2,3].each do |n|  
  array << n * 2  
end
```

# !!variable の説明

```
def enable
```

```
  !!foo
```

```
en
```

`!!` はオブジェクトをtrueまたはfalseに変換する書き方です。  
 真偽反転の!が2回つかわれていると解釈すれば意味を取りやすいです。  
 nilまたはfalseのときはfalseへ、それ以外はtrueへ変換します。

!!は必ずtrueまたはfalseに変換するので、任意のオブジェクトやnilをメソッドから返したくないときによくつかわれます。

メソッド名の末尾が?で終わるメソッドはtrue/falseを返す慣習になっているので、!!が便利につかえる場面が多いです。

一方で、ifで判定するときなど、メソッドの戻り値としてつかわないときは!!をつけずにそのままつかうことが多いです。

```
  rubocop(Lecture/ExplainBangBang)
```

# コーディング中にリアルタイムアドバイス表示する仕組みの検討

- 構成案: アドバイスプログラム - LSP - エディタ
- LSP(Language Server Protocol)
  - エディタを制御するためのプロトコル
  - LSPがエディタ間の差を吸収してくれる
  - LSPに対応しているエディタ、たとえばVSCodeやEmacsで利用可能になる
- この構成でも良いが、RuboCopのエコシステムに乗ると楽ができる
  - RuboCop: Rubyコード向け静的解析検査ツールとその検査ルール集

## RuboCopを利用したときの構成

- 構成: RuboSensei - RuboCop - LSP - エディタ
- RuboCopにはLSP機能が標準でついていて、リアルタイムに検査結果をエディタで表示
- アドバイス対象のコード判定と表示文章をRuboCopカスタムCop(検査ルール)で実装
  -  実行や表示などLSPより後ろはRuboCopがやってくれる
- RuboCopのオートコレクト機能をつかってコード置換も可能

# RuboSenseiのつかい方 1

- <https://github.com/igaiga/rubocop-sensei> READMEに書いてあります
- Railsアプリでのつかい方は以下です
- Gemfile に以下を追加

```
group :development do
  gem "rubocop", require: false
  gem "rubocop-sensei", require: false
end
```

- bundle installを実行
  - \$ bundle install

## RuboSenseiのつかい方 2

- .rubocop.yml を以下の内容で作成または追記
  - RuboSenseiの各種アドバイスはLectureカテゴリに入っているので有効化
    - (LectureはRuboCop標準ではつかわれていない独自カテゴリ)

```
require:  
  - rubocop-sensei
```

```
Lecture:  
  Enabled: true
```

- もしも、RuboSenseiのアドバイスだけに限定して、RuboCopの他の表示を消すときは以下を追加してください

```
AllCops:  
  DisabledByDefault: true
```

## RuboSenseiのつかい方 3

- VSCodeをインストール
  - <https://code.visualstudio.com>
- VSCode拡張 vscode-rubocop をVSCodeへインストールして有効化
  - <https://marketplace.visualstudio.com/items?itemName=rubocop.vscode-rubocop>
- rubocopコマンドが実行可能になっていれば、自動実行して結果を表示します
  - 手動でLSPを起動するコマンドを実行する必要はありません

# RuboSenseiの実装解説

- RuboCopではコードを静的解析したASTをつかって、対象コードを判別します
- 要素技術
  - 静的解析 構文解析 Parser gem: コードを解析してASTを得ます(後述)
  - 静的解析 型解析 TypeProf: 一部の解析に型推論をつかっています
  - RuboCop: カスタムCop(判定ルール)としてアドバイス表示を実装
  - LSP(Language Server Protocol): エディタと情報をやりとりする仕様
  - VSCode: LSPに対応したエディタ
- 今日は要点だけ説明します
- RuboCopカスタムCopの作り方については以下のページに書きました
  - Railsの練習帳: RuboCop カスタムCopの作り方
  - [https://zenn.dev/igaiga/books/rails-practice-note/viewer/rails\\_rubocop\\_custom\\_cop](https://zenn.dev/igaiga/books/rails-practice-note/viewer/rails_rubocop_custom_cop)

## RuboSensei実装例: アドバイス対象コード `!!foo`

```
def enable?  
  !!foo  
end
```

- このRubyコードの `!!foo` に対してアドバイスを表示
- `!!foo` に対するアドバイス
  - 真偽値を反転するnot演算を行う!を2回つかっているコード
  - 真偽値true, falseに変換したいときにつかう
    - fooに代入されたオブジェクトがfalseかnilのときはfalseを返す
    - それ以外のオブジェクトではtrueを返す
  - 末尾が?で終わるメソッド（真偽値を返す）などでよくつかわれる

# !!foo の意味をアドバイスするRuboSenseiのコード

```
class ExplainBangBang < Base
  MSG = <<~STRING
    `!!`はオブジェクトをtrueまたはfalseに変換する書き方です。 # 略
  STRING

  def on_send(node)
    if node.send_type? && node.method_name == :!
      if node.children.any?{|n| n.respond_to?(:send_type?) &&
        n.send_type? &&
        n.method_name == :! }

        add_offense(node)
      return
    end
  end
end
end
```

- [https://github.com/igaiga/rubocop-sensei/blob/main/lib/rubocop/cop/lecture/explain\\_bang\\_bang.rb](https://github.com/igaiga/rubocop-sensei/blob/main/lib/rubocop/cop/lecture/explain_bang_bang.rb)

# AST (Abstract Syntax Tree: 抽象構文木) をつかって対象コード判定

- ASTはRubyコード実行時につくられる中間表現
- Rubyコード直接判定と比べてAST判定では楽になることが多い
  - スペース有無や個数を考えなくて良い
  - 文字列の書き方の種類を考えなくて良い
    - シングルクォート、ダブルクォート、ヒアドキュメント など
- Rubyコード

```
!!foo
```

- AST

```
s(:send,  
  s(:send,  
    s(:lvar, :foo), :!), :!)
```

# RuboCopカスタムCopを実装して判定する流れ

- Parser Gemの支援
  - RuboCopは標準でParser Gemをつかうようになっている
  - 対象コードを静的解析してASTを生成
  - コールバックメソッド: `on_send`
    - AST中にメソッド呼び出しである `:send` が出てきたら呼び出してくれる
    - 解析結果ASTを`on_send`メソッドの引数(`node`)に渡してくれる
    - ほかにいろいろ便利なコールバックメソッドがある

# RuboCopカスタムCopを実装する方法

- RuboCopカスタムCopをつくるときの主な作業
  - 表示するメッセージを用意する
  - ASTを調べてアドバイス表示したいコードかどうかを判定する
  - 判定の結果、表示するときはadd\_offenseメソッドを呼び出す

```
class ExplainBangBang < Base
  # 表示するメッセージ
  MSG = "`!!`はオブジェクトをtrueまたはfalseに変換する書き方です。(略)"
  def on_send(node) # メソッド呼び出しのときに呼ばれる
    if # ここでASTを判定
      add_offense(node) # メッセージを表示する
      return
    end
  end
end
```

# Parser Gem が生成するAST

- コード

```
!!foo
```

- AST
  - on\_sendメソッドの引数nodeに代入されて渡ってくる

```
s(:send,  
  s(:send,  
    s(:lvar, :foo), :!), :!)
```

- 判定コード

```
def on_send(node)  
  # ここでASTを見て判定する  
  add_offense(node) # アドバイス対象のときadd_offenseメソッドを呼ぶ  
end
```

## ASTが対象コードかを判定するコード

```
s(:send,  
  s(:send,  
    s(:lvar, :foo), :!), :!)
```

- ASTを観察して判定条件を考える
- `(:send, xxx, :!)` の中にもう1つ `(:send, xxx, :!)` が存在する
- `(:send, xxx, :!)` 判定 `node.send_type? && node.method_name == :!`
- 中にあるか判定 `node.children.any?{|n| #ここで(:send, xxx, :!)か判定 }`

## 対象コードかを判定するコード

- できあがった判定コード

```
def on_send(node)
  if node.send_type? && node.method_name == :!
    if node.children.any?{|n| n.respond_to?(:send_type?) &&
                             n.send_type? &&
                             n.method_name == :! }
      add_offense(node)
    end
  end
end
```

# 一般的なRuboCopカスタムCopとRuboSenseiの差異

- 理解のためのアドバイスか、問題点の指摘か
  - RuboSenseiでは学習者が理解しやすいアドバイスを(問題点以外でも)出力する
  - 一般的なRuboCop Copは問題のあるコードを指摘する
- 難易度
  - RuboSenseiでは初学者も理解しやすい難易度のアドバイスを目指す
  - 一般的なRuboCop Copのメッセージを初学者が理解するのは難しい
  - RubyKaigi 2022 講演 "[The Better RuboCop World to enjoy Ruby](#)" でも問題提起
- 有効設定、無効設定
  - RuboSenseiでは学習者が学んだらアドバイスをオフにしたい
    - Lectureカテゴリを用意  RuboSenseiのアドバイスだけON/OFF可能に
  - 一般的なRuboCop Copはコード品質向上を目的として永続的に有効にする

# リアルタイムに支援するコーディング環境RuboSensei まとめ

- 初学者向けにエディタ上でリアルタイムに支援するコーディング環境RuboSensei
  - <https://github.com/igaiga/rubocop-sensei>
- コードの意味を解説したり、より良い書き方を教えてくれる
- わかりやすい文章で、正しい内容を伝える
- RuboCopエコシステムを利用してエディタ上でリアルタイムにアドバイス表示
- ASTを調べてアドバイス対象コードかどうかを判定
- (現在は)AIはつかっていない
- 課題: 便利と感じるレベルにするには相当数のアドバイスを実装する必要がある

## AI支援の現状とそれぞれの特徴

- Rubyをつかってコードを書くときの以下のAI支援の現状をまとめ、特徴を比較する
- ChatGPT
- GitHub Copilot
- GitHub Copilot chat

# ChatGPT

- ブラウザやAPIでチャット形式で質問したりコードを提案してもらったりできる
- テンプレート的なソースコードの叩き台をもらうのに便利
- クラス名、変数名などの名前付けの相談に便利
- 返答の真偽を実装者が判断する必要がある
- <https://chat.openai.com>
  - ユーザー登録して利用可能

# GitHub Copilot

- VSCode上でコードを提案、挿入してくれる
  - コメントを書くとその内容に沿ったコードを提案
  - コードを途中まで書くとその先のコードを提案
- 生成されたコードは良い確率で意図通りになっている
- ただし、生成されたコードにバグがないか、意図通りかを実装者が判断する必要がある
- つかい方
  - <https://docs.github.com/ja/copilot/getting-started-with-github-copilot>
  - 料金: <https://docs.github.com/ja/billing/managing-billing-for-github-copilot/about-billing-for-github-copilot>
    - 10 USD/月 or 100USD/年

# GitHub Copilot chat

- VSCode上でチャット形式で質問したりコードを提案してもらったりできる
- 広範囲の質問に答えられる
- 返答は多くのケースで良い回答をもらえる
- 「その」などの指示語が出てきたときに、何を指すかを読み解く必要はある
- 返答の真偽を実装者が判断する必要がある
  - Google検索で調べた結果の妥当性を判断することと本質的には同じ作業
- 揚げ足取りになってしまいますが、次のページから注意が必要な例を説明します
- つかい方
  - <https://docs.github.com/ja/copilot/github-copilot-chat/using-github-copilot-chat>
  - 「GitHub Copilotのアクティブなサブスクリプションが必要」

# GitHub Copilot chat 注意が必要な回答例



igaiga  
LIKE検索について教えて

GitHub Copilot

LIKE演算子は、SQLクエリで文字列の部分一致検索を行うために使用されます。LIKE演算子を使用すると、指定した文字列が他の文字列の中に含まれている場合に、その文字列を検索できます。

例えば、titleカラムに対してLIKE演算子を使用して、search変数に格納された文字列を含む書籍を検索する場合、以下のようにクエリを記述できます。

```
where("title LIKE ?", "%#{search}%")
```

上記のクエリでは、?に対応する部分に%#{search}%を埋め込んでいます。%は、0文字以上の任意の文字列を表します。つまり、%#{search}%は、search変数に格納された文字列が、titleカラムのどこかに含まれている場合にマッチします。

- ActiveRecordのLIKE検索について質問したときの答え

```
where("title LIKE ?", "%#{search}%")
```

- 動作はするが、LIKE節に関するサニタイズがなく、SQLインジェクションの可能性あり
  - 通常のwhereではプレースホルダをつかえばOKだが、LIKE節ではサニタイズが必要
- 詳しい解説: [https://zenn.dev/igaiga/books/rails-practice-note/viewer/ar\\_sql\\_injection](https://zenn.dev/igaiga/books/rails-practice-note/viewer/ar_sql_injection)

# GitHub Copilot chat 注意が必要な回答例 つづき1

- SQLインジェクションを回避するサニタイズについて尋ねると正しく教えてくれる



igaiga

サニタイズはいらないの？



GitHub Copilot

`sanitize_sql_like`メソッドは、SQLクエリ内の特殊文字をエスケープするために使用されます。このメソッドを使用することで、ユーザーが入力した文字列に含まれる特殊文字が、SQLクエリの構文に影響を与えることを防ぐことができます。

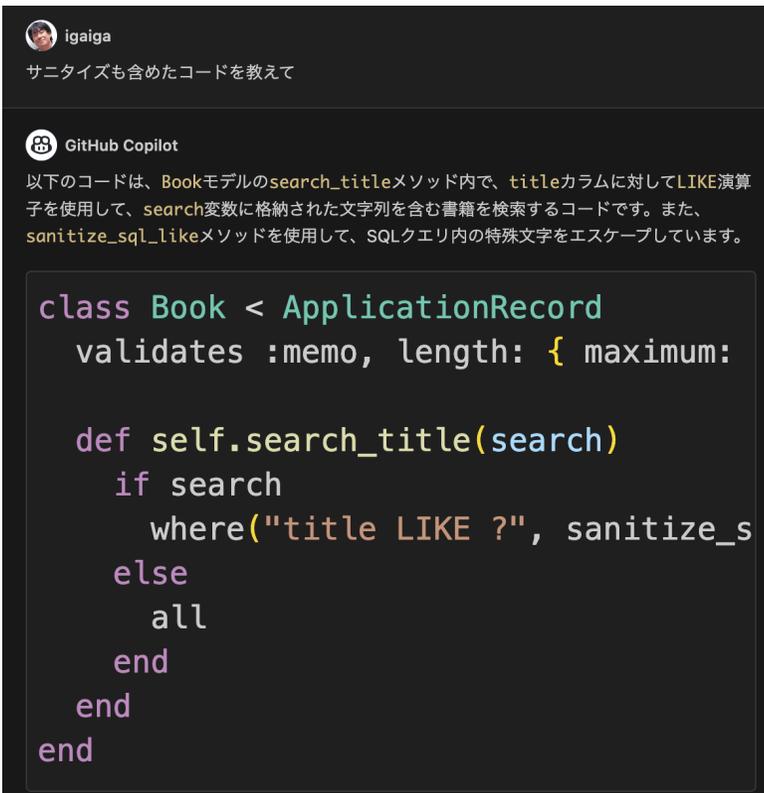
例えば、`search`変数に格納された文字列に`%`が含まれている場合、`sanitize_sql_like`メソッドを使用しないと、SQLクエリの構文が崩れてしまいます。そのため、`sanitize_sql_like`メソッドを使用して、特殊文字をエスケープすることが推奨されます。

したがって、この場合には`sanitize_sql_like`メソッドを使用することをお勧めします。

# GitHub Copilot chat 注意が必要な回答例 つづき2

- 反映したコードを欲しいと伝えると正しいコードを教えてくれる

```
where("title LIKE ?", sanitize_sql_like("%#{search}%"))
```



Igaiga  
サニタイズも含めたコードを教えてください

GitHub Copilot  
以下のコードは、Bookモデルのsearch\_titleメソッド内で、titleカラムに対してLIKE演算子を使用して、search変数に格納された文字列を含む書籍を検索するコードです。また、sanitize\_sql\_likeメソッドを使用して、SQLクエリ内の特殊文字をエスケープしています。

```
class Book < ApplicationRecord
  validates :memo, length: { maximum:

  def self.search_title(search)
    if search
      where("title LIKE ?", sanitize_s
    else
      all
    end
  end
end
```

# RuboSenseiとAI支援の比較とまとめ

- RuboSensei
  - 情報の正しさは(それなりに)担保されている
  - 学び方を設計することができる
  - 支援を受けられる範囲は狭い
  - 課題: 便利と感じるレベルにするには相当数のアドバイスを実装する必要がある
- AI支援
  - 幅広い範囲で支援を受けられる
    - 支援可能範囲を実装ではなく学習で広げることができる
  - 質問時に動的な会話を繰り返せるメリットがある
  - (現状では)正しさを判別する能力が利用者に必要

## RuboSenseiの今後

- RuboSenseiでのアドバイスの拡充
  - アドバイスの実装を増やしていく
  - アドバイスの実装をAIに任せられたら楽(方法模索中)
- 直接LSPをつかってエディタ上での表現力を上げてみたい
  - LSPとエディタUIの対応表みたいなものが欲しい
  - LSPに詳しい方はぜひ教えてください

# どんな世界をつくりたいか

- 書籍、資料とコーディング環境の距離を縮めたい
  - 例: 「このコードはこの本のこの部分で説明されています」と教えてくれる
    - 書籍「Ruby超入門」「パーフェクトRails」「Railsの練習帳」など
  - 例: 「もっと学びたいときはこの資料が役立ちます」と教えてくれる
    - 書籍「RubyとRailsの学習ガイド」など
- 目指したい良い学びとは
  - 書いたコードに対して学ぶと良いことがあることを知れる
  - それを学ぶための正しさが担保された、現在の能力で理解できる資料を読める
  - 深掘りしたいときに学ぶ資料、質問できる人やコミュニティへアクセスできる
- 地道な設計実装計画: RuboSensei
- 夢のある計画: AIに自分の資料を学習させて質問対応
  - LLM(大規模言語モデル)+RAG(検索強化済み文章生成)などのアプローチ (後述)
  - AIに学習させやすい書籍資料の形を模索、ライセンス面でAI学習材料許諾

## 自由研究: LLM(大規模言語モデル)+RAG(検索強化済み文章生成)

- 自分の資料を学習させて回答させる実験
- LlamaIndex <https://www.llamaindex.ai>
  - 自分で用意したデータを学習させてChatGPT APIで回答させるサービス
  - 学習コード(Python):  
<https://gist.github.com/igaiga/9a38e557aef1e0f1925a48958ef2ec44>
- 「Railsの練習帳」を学習させて前述の「LIKE検索について教えて」を質問する

# 自由研究: LlamaIndexで自分のデータを学習させて答えさせた結果

- 「Railsの練習帳」を入力して前述の「LIKE検索について教えて」を質問する
  -  下記の（SQLインジェクションについて考慮された）良い答えが返ってきた

LIKE演算子は、通常の場合には位置指定ハンドラや名前付きハンドラを使用することでサニタイズ（悪意のあるコードを無害な形に変換すること）が行われますが、LIKE演算子の場合には自動的にサニタイズが行われません。具体的には、WHERE句のサニタイズはされるものの、LIKE句のサニタイズは行われないう動作になります。

Railsでは、LIKE句のサニタイズにはActiveRecord::Base.sanitize\_sql\_likeメソッドを使用します。このメソッドでは、LIKE句で使用される特殊文字の%と\_がそれぞれ\%と\\_に置き換えられます。

したがって、LIKE検索を安全に行うためには、パラメータをActiveRecord::Base.sanitize\_sql\_likeメソッドでサニタイズしてからクエリに組み込む必要があります。例えば、Package.where("code like ?", "%" + ActiveRecord::Base.sanitize\_sql\_like(params[:str]) + "%")のように記述します。

# 「Railsの練習帳」を機械学習の学習データとして活用してもらいたい

- 「Railsの練習帳」の内容は非商用の範囲において、LLMなど機械学習への入力データとして利用可能です。
- 商用利用の場合はご相談ください。できるだけ柔軟に対応できればと考えております。
- <https://zenn.dev/igaiga/books/rails-practice-note/viewer/preface>

## LLMなど機械学習への入力データとしての利用について

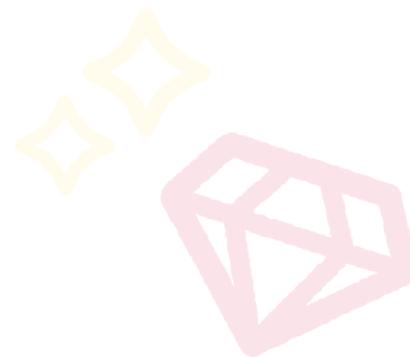
「Railsの練習帳」の内容は非商用の範囲において、LLMなど機械学習への入力データとして利用可能です。

商用利用の場合はご相談ください。できるだけ柔軟に対応できればと考えております。

- 個人で権利を自由につかえる書籍は学習データとして利用してもらいやすい
- 商業誌での文章は権利者への調整が必要になるかもしれない
- 機械学習へ入力可否はライセンスで明記すべきなのか

ご清聴ありがとうございました

Rubbo  
Sensei



Rubbo  
Sensei

