

最高の DX - Nuxt Typed Router と Pinia Colada で実現する次世 代 Vue/Nuxt 開発

Twitter Share

発表中、もしよろしければツイートしてください。

自己紹介



ナイトウコウスケ

Composition API 生まれ script setup 育ち

株式会社メイツ

株式会社メイツ

- EdTech 企業
- Web アプリ
- Vue / Nuxt



カジュアル面談



株式会社メイツ

フォロー中

<https://mates-edu.co.jp> 東京都

ホーム 私たちについて メンバー ストーリー 募集

教育を、 アップデートする。

教育とテクノロジーを融合させ、
教育が永続的に進化する仕組みを創る。



ホーム

ミッション

メンバー

働き方やメンバーに
興味がある方へ

仕事に興味がある方へ

会社情報

教育をアップデートし、子どもたちに最適な教育を提供する

教育ICTを活用し、子供たち一人ひとりに最適な教育が届くよう、我々は教育のアップデートを
目指しています。

クイズ なんの数字？

57 44 54

35 42

Vue Fes
Japan 2025

ビューフェス
ジャパン2025

OCTOBER 25, 2025

2025年10月25日



Vue Fes Japan 2025 に関する数

スポンサー企業

57

個人スポンサー

44

スタッフ

54

ボランティア

35

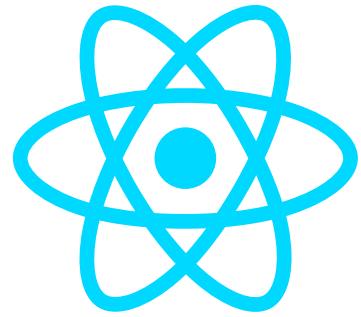
登壇者

42

※ <https://vuefes.jp/2025/sponsors> ※ 同じ人をめっちゃ重複して数えている ※ ざっくり数え

感謝





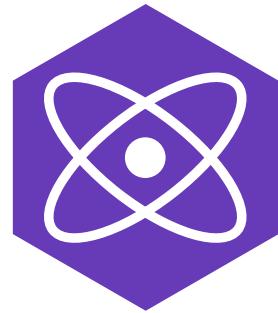












懇親会の話のタネに

懇親会の話のタネになれば嬉しいです

- 「ナイトウの発表見ました！」
- 「Pinia Colada って…」
- 「データフェッチってどうしますか？」
- 「Nuxt Typed Router って良さそうですね！」
- 「React / Next.js ではどんな感じ？」

エコシステム

Vue の未来

"The limits of my language mean the limits of my world."

「人間は、いま自分が話している言葉の範囲でしか、世界を考えることができない。」

— Ludwig Wittgenstein, 1922

たまには外の世界に出て知らない人の話を聞こう！

「どうせ技術的負債になるから」 っていうのは設計とか実装とか上手くなる機会を逃している



SignalとObservable—新たなデータモデルを解きほぐす - lacolaco / TS...



後で見る

共有

TsKaigi 2025

レバレジーズトラック

セッション

DAY1

13:40 ~ 14:10

SignalとObservable—新たなデータモデルを解きほぐす



lacolaco

「ライブラリの話かな」 って思っちゃうけど、 抽象化すれば抽象的に見れば その言語がどのような課題を有しているかわかる

ライブラリを見ることで 言語の未来を 想像することができる

<https://youtu.be/OUsXjtkLYu0>

ということで

タイトル再表示

最高の DX - Nuxt Typed Router と Pinia Colada で実現する次世代 Vue/Nuxt 開発

「最高の」という表現は主観的で誇張的である可能性があります。より客観的な評価を示すことを検討してください。
(ai-writing/no-ai-hype-expressions)

DX

Developer Experience = 開発者体験

開発プロセス全体で 開発者が感じる体験の総体

DX = なんかいいな

最高の DX

開発者の認知負荷を最小化 本質的な問題解決に集中

課題

開発って大変

Web アプリ開発(フロントエンド)

- ルーティング
- データフェッチ
- フォーム
- エラーハンドリング
- 認証・認可
- パフォーマンス
- 状態管理
- アクセシビリティ
- 国際化(i18n)
- テスト
- SEO 対策
- セキュリティ
- UI/UX
- ビルド・デプロイ
- モニタリング

切っても切れない

のに

ルーティングを手打ち(TS 使ってるので)

```
router.push(`/user/${userId}`);
router.push("/user/profile");
router.push("/user/settings");
```

データフェッチングのボイラープレート

```
const question = ref("");
const answer = ref("Questions usually contain a question mark. ;-)");
const loading = ref(false);

watch(question, async (newQuestion, oldQuestion) => {
  if (newQuestion.includes("?")) {
    loading.value = true;
    answer.value = "Thinking...";
    try {
      const res = await fetch("https://yesno.wtf/api");
      answer.value = (await res.json()).answer;
    } catch (error) {
      answer.value = "Error! Could not reach the API. " + error;
    } finally {
      loading.value = false;
    }
  }
});
```

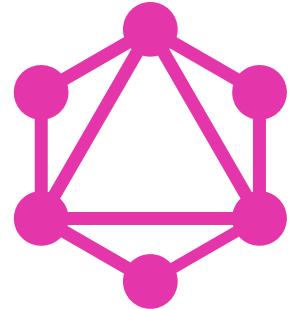
from <https://vuejs.org/guide/essentials/watchers>

Nuxt Typed Router と Pinia Colada

型安全なルーティング 宣言的データフェッチング

どこで
出会ったか

技術スタック



Nuxt Typed Router



公式ドキュメント

<https://nuxt-typed-router.vercel.app/>

Nuxt Typed Router とは

型安全なルーティング Nuxt モジュール

nuxt-typed-router

Overview Repositories 54 Projects Packages Stars 579



victorgarciaesgi / README.md

Hi 🎉

I'm Victor, a French developer. currently working as Senior Front engineer at [Malt](#)

My stack is primarily based on Typescript, Nuxt and Vue.

I love to make everything type-safe ✅.

All my projects can be seen at <https://vicflix.dev/>

Pinned

[regle](#) (Public)

Headless form validation library for Vue.js

TypeScript ⭐ 291 🏷 9

[nuxt-typed-router](#) (Public)

Provide autocompletion and typecheck to Nuxt router

TypeScript ⭐ 411 🏷 15

[vue-chart-3](#) (Public)

A simple wrapper around Chart.js 3 for Vue 2 & 3

TypeScript ⭐ 316 🏷 12

[vicflix](#) (Public)

My portfolio taking inspiration of Netflix UI

Vue ⭐ 13 🏷 4

[dewightjack/vue-types](#) (Public)

Vue3 Prop Types definitions

TypeScript ⭐ 580 🏷 34

[apertureless/vue-charts](#) (Public)

Vue.js wrapper for Chart.js

TypeScript ⭐ 5.7k 🏷 829

124 followers · 89 following

<https://www.malt.fr/>

Paris, France

06:57 - 7h behind

victorgarciaparis13@gmail.com

<https://vicflix.dev/>

@desnoth_dev

@vicflix.dev

in/victor-garcia-1793b2a9

Achievements



866 contributions in the last year



Learn how we count contributions

Less More

2025

2024

2023

2022

2021

ファイル構造から自動型生成

Nuxt のファイルベースルーティング

```
pages/
  index.vue      → /
  users/
    index.vue    → /users
    [id].vue     → /users/:id
  login/
    index.vue   → /login
```

生成される型

```
// @typed-router エイリアスから利用可能
import { RoutesNamesList, RoutesParamsRecord } from "@typed-router";

// ルート名のユニオン型
type RoutesNamesList = "index" | "users" | "users-id" | "login";

// パラメータの型辞書
type RoutesParamsRecord = {
  index: never;
  users: never;
  "users-id": { id: string };
  login: never;
};
```

型安全なナビゲーション

```
// RoutesNamedLocations - router.push() の型安全化
type RoutesNamedLocations =
  | { name: "index" }
  | { name: "users" }
  | { name: "users-id"; params: { id: string } }
  | { name: "login" };

// ルート名に応じて params が自動的に切り替わる
router.push({ name: "users-id", params: { id: "123" } }); // ✓ OK
router.push({ name: "users-id" }); // ✗ エラー: params が必須
router.push({ name: "login", params: {} }); // ✗ エラー: params 不要
```

型による絞り込み

```
const route = useRoute();

// route.name で分岐すると params の型が絞り込まれる
if (route.name === "users-id") {
  console.log(route.params.id); // string 型
}

// ルート名を指定してアサーションも可能
const route = useRoute("login");
console.log(route.name); // "login" 型
```

生成される routesNames

```
const routeName = routesNames.usersId; // "users-id"

// 実際に生成されるオブジェクト
export const routesNames = {
  index: "index" as const,
  users: "users" as const,
  usersId: "users-id" as const,
  login: "login" as const,
};
```

TypeScript の恩恵

IDE の補完が効く typo に気づく

快適な開発体験

```
router.push({ name: "user-id" }); // Error! ✗  
router.push({ name: "user-id", params: { bar: "baz" } }); // Error! ✗  
router.push("/user"); // Error! ✗  
  
const id = "ey7878";  
router.push(`/user/${id}`); // Good! ✓  
router.push({ name: "user-id", params: { id } }); // Good! ✓  
  
router.push(`/user/${id}/baguette`); // Error! ✗
```

従来のルーティング問題

```
const router = useRouter();

const navigateToUser = (userId) => {
  router.push(`/usr/${userId}`); // ✗ typo: actual "user"
};
```

⚠ typo の危険性

従来のルーティング問題

```
// /users/123/posts/456/comments?sort=latest
import { useRoute } from "vue-router";

const route = useRoute();

const currentUserId = route.params.id;
const currentTab = route.query.tab;
```

💡 パラメータの型が曖昧

従来のルーティング問題

🤔 < ユーザー詳細画面だからきっと `user-detail` だよね

🤔 < ユーザー ID だからパラメータはもちろん `userId` 👍

```
router.push({  
  name: "user-detail", // ✗ 実際は user-id  
  params: { userId }, // ✗ 実際は id  
});
```

😭 IDE の補完が効かない

こうなつたら嬉しい

理想的なルーティング開発体験

- typo したらすぐ気づく
- パラメータの型が明確
- リファクタリングが安全
- IDE の補完が効く

Nuxt Typed Routerによる解決

typo を防ぐ & IDE 補完が効く

```
import { useRouter } from "@typed-router";
const router = useRouter();

// ✓ IDE が自動補完候補を表示
router.push({
  name: "articles-slug", // Tab で候補が出る
  params: { slug: "hello-world" }, // 必須パラメータを提案
});

// ✗ typo や存在しないルートはコンパイルエラー
router.push({ name: "usr-id" }); // TypeScript Error!
```

型が明確

```
import { useRoute } from "@typed-router";

const route = useRoute();

// 条件分岐で型が自動的に絞り込まれる
if (route.name === "articles-slug") {
  const slug = route.params.slug; // string型として推論
  console.log(slug.length); // ✅ stringメソッドが補完
}
```

体験してみてください➡️

Nuxt Typed Router で DX 向上

IDE の補完が効く

- ルート名やパラメータ名が自動補完される
- typo を未然に防ぐ

静的解析でエラー検知

- 存在しないルートへの遷移を検出
- リファクタリング時の修正漏れを防ぐ

レビューコスト減

- 型システムが正しさを保証
- レビュアーは型エラーではなくロジックに集中できる
- (最近は AI がそれっぽいのを提出してくる)

型安全なコンポーネント

<NuxtLink /> をラップしたコンポーネント

```
<script setup lang="ts" generic="T extends RoutesNamesList, P extends string">
import type { RoutesNamesList, NuxtRoute } from "@typed-router";

const { to } = defineProps<{
  to: NuxtRoute<T, P>;
}>();
</script>

<template>
  <NuxtLink :to><slot /></NuxtLink>
</template>

<script setup lang="ts" generic="T extends RoutesNamesList, P extends string">
import type { RoutesNamesList, NuxtRoute } from "@typed-router";

// 外部リンク
const { to, external } = defineProps<
  { to: NuxtRoute<T, P>; external?: false } | { to: string; external: true }
>();
</script>

<template>
  <NuxtLink :to :external><slot /></NuxtLink>
</template>
```

Script Setup Generics とは

Vue 3.3+

<script setup> に直接ジェネリック型パラメータを定義できる

```
<script setup lang="ts" generic="T extends SomeType">
// T は型パラメータとして利用可能
</script>
```

コンポーネント単位で型安全な props を実現

Genericsによる型安全なコンポーネント

```
<script setup lang="ts" generic="T extends RoutesNamesList, P extends string">
  import type { RoutesNamesList, NuxtRoute } from "@typed-router";

  const props = defineProps<{
    to: NuxtRoute<T, P>;
  }>();
</script>
```

- T : ルート名の型 (e.g. 'index' | 'about' | 'articles-slug')
- P : パス文字列の型
- NuxtRoute<T, P> : 型安全なルート定義

Nuxt Typed Router のここが好き

Out-of-Box I18n Support

@nuxtjs/i18n

useLocalePath() , useLocaleRoute() , <NuxtLinkLocale>

```
const localePath = useLocalePath();

// Error ✗
navigateTo(localePath({ name: "user-id" }, "fr"));

// Good ✓
navigateTo(localePath({ name: "user-id", params: { id: 1 } }, "fr"));
```

```
const localeRoute = useLocaleRoute();

// Error ✗
const route = localeRoute({ name: "user-id" }, "fr");

// Good ✓
const route = localeRoute({ name: "user-id", params: { id: 1 } }, "fr");
if (route) navigateTo(route fullPath);
```

```
<template>
<NuxtLinkLocale :to="{ name: 'users' }" locale="fr"> />
```

え、まって

Nuxt Typed Pages は？

Nuxt Typed Router と Nuxt Typed Pages は別物

3rd party module VS Official feature

安定 VS 実験的機能

Nuxt Typed Router は 3rd party module

↳ 安定

Nuxt Typed Pages は公式機能

 実験的機能

Nuxt Typed Pages

- `unplugin-vue-router` をベースにした型安全ルーティング
- Nuxt Typed Router と同様の機能を Nuxt 本体で提供

```
// nuxt.config.ts
export default defineNuxtConfig({
  experimental: {
    typedPages: true,
  },
});
```

Nuxt 3.5+

unplugin-vue-router

型付きファイルベースルーティング

for Vue 3

Nuxt Typed Pages の基盤技術

Not Only Nuxt

unplugin-vue-router と Nuxt Typed Router

This project idea came from trying [to type the router directly using Typescript](#), finding out it's not fast enough to be pleasant to use and, ending up using build-based tools, taking some inspiration from other projects like:

- [Nuxt](#) - The Vue.js Framework
- [vite-plugin-pages](#) - Framework agnostic file based routing
- [Typed Router for Nuxt](#) - A module to add typed routing to Nuxt

It's currently included as [an experimental setting in Nuxt](#).

unplugin-vue-router は Nuxt Typed Router からインスピレーションを得て作られました

unplugin-vue-router

Overview Repositories 333 Projects Packages Stars 1.2k Sponsoring 3



Eduardo San Martin
Morote
posva

Unfollow Sponsor

Member of the @vuejs core team
Speaker, trainer. From 🇪🇸, lives in 🇫🇷

6.4k followers · 61 following

Freelance
Paris
02:45 - 7h behind
<https://esm.dev>
@posva
@posva@m.webtoilis
@esm.dev

posva / README.md

Keeping the Vue Ecosystem enjoyable ⚡
www · X · RuleKit · Mastering Pinia · sponsor

Pinned

- unplugin-vue-router (Public)
Next Generation file based typed routing for Vue Router
TypeScript 2.2k 108
- vuejs/pinia (Public)
Intuitive, type safe, light and flexible Store for Vue using the composition api with DevTools support
TypeScript 14.2k 1.2k
- pinia-colada (Public)
The smart data fetching layer for Vue
TypeScript 1.8k 58
- vuejs/router (Public)
The official router for Vue.js
TypeScript 4.4k 1.3k
- vuejs/vuefire (Public)
Firebase bindings for Vue.js
TypeScript 3.9k 343
- cating (Public)
Insanely fast image printing in your terminal
C 1.5k 61

2,502 contributions in the last year



Less More

Oct Nov Dec Jan Feb Mar Apr May Jun Jul Aug Sep Oct
Mon Wed Fri

Learn how we count contributions

2025 2024 2023 2022 2021

by posva Eduardo San Martin Morote

posva's OSS

- Vue Router / unplugin-vue-router
- Pinia / Pinia Colada
- VueFire
- mande
- vue-promised

Pinia Colada



piña colada

ラムをベースにパイナップルジュースとココナッツミルクを碎いた氷と一緒にシェイクして作るロングドリンク。黄白色で甘みが強い。



Pinia Colada

公式ドキュメント

<https://pinia-colada.esm.dev/>

Pinia Colada とは

Vue 用の非同期状態管理ライブラリ

Vue 用のデータフェッキングライブラリ

特徵

Automatic Caching

client-side cache dedupe



Async State

あらゆる非同期状態の処理

 Plugins

強力なプラグインシステム

✨ Optimistic Updates

樂観的更新も簡単に実現



Sensible Defaults

健全なデフォルト設定 完全なカスタマイズ性



Out-of-the-Box Plugins

データフェッチング用の composable

🔑 TypeScript Support

TypeScript 型サポート

 Small Bundle Size

約 2kb のベースライン fully tree-shakeable



Zero Dependencies

依存関係なし(Pinia 以外)



SSR サポート



Data Loaders

Vue Router Data Loaders

インストール

ni @pinia/colada

! Pinia も必要

"peerDependencies": { "pinia": "^2.2.6 || ^3.0.0", "vue": "^3.5.17" }

Vue プラグインを設定

```
import { createApp } from "vue";
import App from "./App.vue";
import { createPinia } from "pinia";
import { PiniaColada } from "@pinia/colada";

const app = createApp(App);
const pinia = createPinia();
app.use(pinia);
app.use(PiniaColada, {
  /** Options */
});
```

もちろん Nuxt module も

```
nlx nuxi@latest module add @pinia/colada-nuxt

// nuxt.config.ts
export default defineNuxtConfig({
  modules: ["@pinia/nuxt", "@pinia/colada-nuxt"],
});
```

BEFORE と AFTER

```
// BEFORE 😞

// 状態
const isLoading = ref(false);
const isPending = ref(true);
const error = shallowRef(null);
const data = shallowRef(null);

// データフェッチ
async function refresh() {
  isLoading.value = true; // 手動更新
  try {
    data.value = await fetchUsers();
    error.value = null; // 手動更新
    isPending.value = false; // 手動更新
  } catch (err) {
    error.value = err; // 手動更新
  } finally {
    isLoading.value = false; // 手動更新
  }
}

refresh(); // 手動の初回データフェッチ
```

いい感じのデータフェッチ

- 宣言的に記述
- ボイラープレートなし
- いい感じにキャッシュ
- 型

Pinia Colada による解決

その前に

データフェッチの

~~考え方や用語を~~

Query と Mutation

 Query データを読み取る

 Mutation データを書き込む

 Cache

取得したデータを保存して再利用

同じデータへのリクエストを削減 アプリケーションの応答性が向上



Active Query

コンポーネントで使用中のクエリ

画面に表示されているデータ 自動的に更新・管理される

Inactive Query

使用されていないクエリ

キャッシュには残っているが非表示 一定時間後にガベージコレクション



Stale / Fresh

キャッシュデータの鮮度管理

データの新鮮さを判断する基準 鮮度に応じて自動的に再取得



Invalidation

キャッシュを明示的に無効化

データ更新後に関連キャッシュを無効化 最新のデータを再取得して表示

Pinia Colada による解決

改めて

宣言的

```
// 「何をフェッチするか」だけを記述
const { data, error, status } = useQuery({
  key: ["todos"],
  query: () => $fetch("/api/todos"),
});

// ✓ loading, error, data の状態管理は自動
// ✓ 初回フェッチも自動
// ✓ cache, dedupe も自動
```

query 関数は引数を取らない

Pinia Colada が「必要なときに自動的にトリガー」

開発者は「何を」 フェッチするかだけを宣言 「いつ」 フェッチするかは Pinia Colada が判断

型

```
const { data } = useQuery({  
  key: ["users"],  
  query: () => $fetch("/api/users"),  
});  
  
// ShallowRef<SerializeObject<User>[] | undefined>  
console.log(data);
```

useQuery()

```
<script setup lang="ts">
const { state, asyncStatus, refresh } = useQuery({
  key: () => ["users"],
  query: () => $fetch("/api/users"),
});
</script>

<template>
  <p v-if="asyncStatus === 'loading'">読み込み中...</p>
  <p v-else-if="state.status === 'error'" role="alert">
    エラー: {{ state.error.message }}
  </p>
  <section v-else-if="state.status === 'success'">
    <p>{{ state.data.name }}</p>
    <button @click="refresh">更新</button>
  </section>
</template>
```

query は引数を取らない 宣言的に記述できる

useMutation()

```
<script setup lang="ts">
const { mutate: createTodo, status } = useMutation({
  mutation: (newTodo: { title: string }) => {
    $fetch("/api/todos", { method: "POST", body: newTodo });
  },
});
</script>

<template>
  <button
    @click="createTodo({ title: '新しいタスク' })"
    :disabled="status === 'pending'"
  >
    {{ status === "pending" ? "作成中..." : "タスク作成" }}
  </button>
</template>
```

mutate は引数を取る

Query と Mutation の連携

Mutation でデータを変更 ✨

Query のキャッシュを無効化。 🟠

Pinia を Cache に使用

内部でキャッシュ管理に Pinia ストアを使用

キャッシュストアの構造

Query Invalidation

```
const queryCache = useQueryCache();

const { state } = useQuery({ key: ["todos"], query: () => $( "/api/todos" ) });

const { mutate } = useMutation({
  mutation: (text: string) =>
    $fetch( "/api/todos", {
      method: "POST",
      body: { text },
    }),
  onSettled: () => {
    queryCache.invalidateQueries({ key: ["todos"] });
  },
});
```

query cache の無効化(onSettled) 新しいデータを取得

Hooks

Mutation のライフサイクル

Mutation Hooks

- `onMutate` - リクエスト送信直前
- `onSuccess` - リクエスト成功時
- `onError` - リクエスト失敗時
- `onSettled` - 成功・失敗に関わらず実行

Hooks で実現できること

- クエリの無効化
- 楽観的更新 (Optimistic Updates)
- エラー時のロールバック
- コンテキストの受け渡し

Query Invalidation

```
// key: ["todos"] を指定すると  
// その key とその子のクエリが無効化される  
  
// exact: true を付けると  
// 指定した key のクエリだけ無効化される  
  
// 引数なしで呼ぶと  
// すべてのアクティブなクエリが無効化される
```

データ更新後、関連するクエリを無効化

useQuery() の 2 つの重要なこと

```
const { state, asyncStatus } = useQuery({  
  key: ["todos"],  
  query: () => $fetch("/api/todos"),  
});
```

- `key` : キャッシュの識別子(配列による階層構造)
- `query` : Promise を返す関数

Query Key の設計が重要

- 配列形式で構造化されたキー
- 階層的な設計でキャッシュを柔軟に管理
- リアクティブな key は関数で定義

```
// 静的な key
key: ["products"];
```



```
// 動的な key(関数形式)
key: () => ["products", route.params.id];
key: () => ["products", route.params.id, { searchResult: true }];
```

Query, Mutate, Invalidate

```
const queryCache = useQueryCache();

// Todo リストの Query
const { data: todos } = useQuery({
  key: ["todos"],
  query: () => $fetch("/api/todos"),
});

// Todo 作成の Mutation
const { mutate: createTodo } = useMutation({
  mutation: (text: string) =>
    $fetch("/api/todos", {
      method: "POST",
      body: { text },
    }),
  onSettled: () => {
    // 作成後に todos リストを無効化して最新データを取得
    queryCache.invalidateQueries({ key: ["todos"] });
  },
});
```

Query Keys の重要性

Query Keys はクエリを一意に識別するための配列

キャッシュ管理の基盤

同じ key を持つクエリは同じキャッシュを共有

key が異なれば別々にキャッシュされる

階層的な構造

```
["todos"], ["todos", id], ...
```

親 key で子も含めて一括無効化が可能

適切な無効化

Mutation 後に関連するクエリを効率的に更新

部分的な無効化で必要なデータだけ再取得

Key 設計の例

```
// 基本: 単純なリスト取得
const { data } = useQuery({
  key: ["todos"],
  query: () => $fetch("/api/todos"),
});

// 階層: ID を含むクエリ
const { data } = useQuery({
  key: () => ["todos", todoId],
  query: () => $fetch(`/api/todos/${todoId}`),
});

// ["todos"] で無効化すると、
// すべての todos 関連のクエリが無効化される

// 階層: さらに詳細な条件
const { data } = useQuery({
  key: () => ["todos", todoId, { withComments: true }],
  query: () => $fetch(`/api/todos/${todoId}?comments=true`),
});

// ["todos"] で無効化 → すべての todos
// ["todos", todoId] で無効化 → 特定の todo だけ
// ["todos", todoId, {}] で無効化 → 特定の todo のすべての条件
```

型安全なキー設計

```
// Before: 文字列キーの問題
const { data } = useQuery({
  key: () => ["users", userId], // typo の可能性
  query: () => fetchUser(userId),
});

// 別の場所で同じキーを使おうとすると...
cache.invalidateQueries({ key: ["user", userId] }); // ✗ typo

// After: defineQueryOptions で型安全に
const userQuery = defineQueryOptions((id: string) => ({
  key: ["users", id],
  query: () => fetchUser(id),
}));

// 使用時は型付きのキーが自動生成される
const { data } = useQuery(userQuery(userId));

// キャッシュ操作も型安全
cache.invalidateQueries(userQuery(userId));
```

key 設計

って

また文字列？

Key Factory パターン

Query Key を一元管理する 型安全なパターン

Key Factory の実装例

```
export const TODO_KEYS = {
  root: ["todos"] as const,
  withFilters: (filters?: TodoFilters) => [...TODO_KEYS.root, filters] as const,
  byId: (id: string) => [...TODO_KEYS.root, id] as const,
  comments: (id: string) => [...TODO_KEYS.byId(id), "comments"] as const,
};
```

Key Factory の使用例

```
// Before: 文字列ベース
const { data } = useQuery({
  key: ["todos"], // typo の可能性
  query: () => fetchTodos(),
});

// After: Key Factory
import { TODO_KEYS } from "~/queries/key-factory";

const { data } = useQuery({
  key: TODO_KEYS.root, // 型安全 !
  query: () => fetchTodos(),
});

// 無効化も簡単
const queryCache = useQueryCache();

// すべての todos を無効化
queryCache.invalidateQueries({ key: TODO_KEYS.root });

// 特定の todo だけ無効化
queryCache.invalidateQueries({ key: TODO_KEYS.byId(todoId) });
```

query の設定も共有

defineQueryOptions

```
import { defineQueryOptions } from "@pinia/colada";

export const todoListQuery = defineQueryOptions(() => ({
  key: ["todos"],
  query: () => $fetch("/api/todos"),
}));

export const todoByIdQuery = defineQueryOptions((id: string) => ({
  key: ["todos", id],
  query: () => $fetch(`/api/todos/${id}`),
}));
```

defineQueryOptions の使用例

```
// Before: 毎回 key と query を書く
const { state } = useQuery({
  key: ["todos", todoId],
  query: () => $fetch(`/api/todos/${todoId}`),
});

// Cache に手動で型を指定
const queryCache = useQueryCache();
const todo = queryCache.getQueryData<Todo>(["todos", todoId]);

// After: 再利用可能な定義
import { todoByIdQuery } from "~/queries/todos";

const { state } = useQuery(todoByIdQuery, todoId);

// Cache の型が推論される
const queryCache = useQueryCache();
const todo = queryCache.getQueryData(todoByIdQuery(todoId).key);
```

Key Factory × defineQueryOptions

```
import { defineQueryOptions } from "@pinia/colada";

export const TODO_KEYS = {
  root: ["todos"] as const,
  getById: (id: string) => [...TODO_KEYS.root, id] as const,
};

export const todoByIdQuery = defineQueryOptions((id: string) => ({
  key: TODO_KEYS.getById(id),
  query: () => $fetch(`/api/todos/${id}`),
}));
```

実装パターン比較

プロジェクト規模に応じた 3 つの実装パターン

① 直書き（インライン）

```
<!-- pages/teams/[id].vue --&gt;
&lt;script setup lang="ts"&gt;
const route = useRoute();

const team = useQuery({
  key: () =&gt; ["teams", route.params.id],
  query: () =&gt; $fetch(`/api/teams/${route.params.id}`),
});

const members = useQuery({
  key: () =&gt; ["teams", route.params.id, "users"],
  query: () =&gt; $fetch(`/api/teams/${route.params.id}/users`),
});
&lt;/script&gt;</pre>
```

特徴: 最少コードだが、キー・取得ロジックが画面に分散しやすい

② Key Factory 单独

```
// queries/key-factory.ts
export const TEAM_KEYS = {
  root: ["teams"] as const,
  byId: (id: string) => [...TEAM_KEYS.root, id] as const,
  users: (id: string) => [...TEAM_KEYS.byId(id), "users"] as const,
};
```

```
!— pages/teams/[id].vue —>
<script setup lang="ts">
import { TEAM_KEYS } from "~/queries/key-factory";

const route = useRoute();

const team = useQuery({
  key: () => TEAM_KEYS.byId(route.params.id),
  query: () => $fetch(`/api/teams/${route.params.id}`),
});

const members = useQuery({
  key: () => TEAM_KEYS.users(route.params.id),
  query: () => $fetch(`/api/teams/${route.params.id}/users`),
});
</script>
```

特徴: キー形状を一元化でき、キャッシュ整合性が向上。取得ロジックは画面に残る

③ Key Factory × defineQueryOptions

```
// queries/key-factory.ts
export const TEAM_KEYS = {
  root: ["teams"] as const,
  byId: (id: string) => [...TEAM_KEYS.root, id] as const,
  users: (id: string) => [...TEAM_KEYS.byId(id), "users"] as const,
};

// queries/team.ts
import { defineQueryOptions } from "@pinia/colada";
import { TEAM_KEYS } from "./key-factory";

export const teamByIdQuery = defineQueryOptions((teamId: string) => ({
  key: TEAM_KEYS.byId(teamId),
  query: () => $fetch(`api/teams/${teamId}`),
}));

export const teamUsersQuery = defineQueryOptions((teamId: string) => ({
  key: TEAM_KEYS.users(teamId),
  query: () => $fetch(`api/teams/${teamId}/users`),
}));
```

```
><!-- pages/teams/[id].vue --&gt;
&lt;script setup lang="ts"&gt;
import { teamByIdQuery, teamUsersQuery } from "~/queries/team";

const route = useRoute();

const team = useQuery(teamByIdQuery, route.params.id);
const members = useQuery(teamUsersQuery, route.params.id);
&lt;/script&gt;</pre>
```

特徴: キー整合性 (Factory) とパラメータ契約・取得ロジック (defineQueryOptions) を集約

3つのパターンの比較

構成	キー整合性	パラメータ共有	変更耐性	規模
直書き	低	低	低	小
Key Factory 単独	中	中	中	中
Key Factory × defineQueryOptions	高	高	高	大

Scalable & Progressive

気軽に書けるし ガチガチにも書ける

Pinia Colada を使用すると

デメリットなく クエリとコンポーネントを 近くにかける

じゃあどうやって決めるか

公式ドキュメントに書いてある

"As your project grows" and you start using more and more queries as well as concepts as Optimistic Updates, you will want to organize your queries.

プロジェクトが成長したら整理しよう

<https://pinia-colada.esm.dev/guide/queries.html#Organizing-Queries>

defineQueryOptions で prefetch

```
// クエリ定義を再利用可能に
const userListQuery = defineQueryOptions({
  key: ["users"],
  query: () => $fetch("/api/users"),
});

// use
const { data, status } = useQuery(userListQuery);

// prefetch (`cache.ensure()`)
const queryCache = useQueryCache();
await queryCache.refresh(queryCache.ensure(userListQuery));
```

Pinia Colada で DX 向上

Pinia Colada のここが好き

いい感じにクエリとコンポーネントを近くに書ける

!?

A screenshot of a code editor showing a TypeScript error in a Vue component. The file is named `[userId=int].vue`. The code is as follows:

```
src > pages > users > [userId=int].vue > {} script setup
    You, 4 minutes ago | 1 author (You)
2  <script lang="ts" setup>
1  const route = useRoute()
3  route.params.userId = parseInt(route.params.userId as string,
10)    You ⚭ userId
1  </script>  Property 'debugger' does not exist on type '{ userId: num
2
```

The error occurs at line 3, column 10, where the IDE highlights the word `userId` with a yellow warning icon. A tooltip appears over the highlighted text, showing the message: `(property) userId: number`. The code editor interface includes a navigation bar with icons for back, forward, and search.

Nuxt Typed Router + Pinia Colada

型安全なデータフローの完成

IDE とのシームレスな統合

ルート名の自動補完 + データ状態の推論

リファクタリング時の安全性

AI 時代

データフェッチとナビゲーションの課題

コンポーネント内でのデータフェッチ

ナビゲーションガードでのプリフェッチ

カスケードフェッチ：依存クエリ

カスケードフェッチ：コンポーネントネスト

unplugin-vue-router の Data Loaders

次世代の データフェッチング手法

unplugin-vue-router の Data Loaders



Eduardo San Martin Morote - Async State Management with Vue Route...



後で見る

共有

Async State Management with Vue Router



Eduardo San Martin Morote



<https://youtu.be/lhjS-6Flxgk>

ありがとうございました