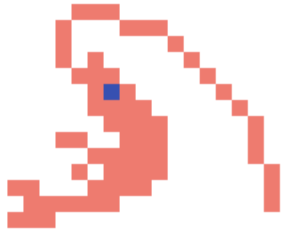
複雑なビジネスルールに挑む：

正確性 と 効率性 を両立する

fp-tsのチーム活用術

株式会社カケハシ

kosui (岩佐 幸翠)



自己紹介

kosui (岩佐 幸翠)

株式会社カケハシ

薬局のDXを推進し
日本の医療体験を変革を目指すSaaS群を提供

組織管理・認証基盤リノベチーム

顧客に寄り添うプロダクト開発チームが
本質的な価値提供に集中できる世界を目指す

2023年9月よりテックリードとして
社内プラットフォームが創出できる
顧客への価値とは何か模索中

@kosui_me

@iwasa-kosui

https://kosui.me

この発表のねらい

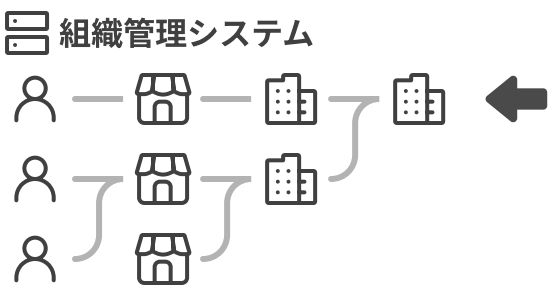
fp-ts で複雑性と闘う

複雑なビジネスロジックでも **正確性** と **効率性** を両立するために

- どのようにfp-tsを活用したか？
- チームで運用するためにどんな工夫をしたか？

題材

一括入稿機能の開発で遭遇した悩みと解決策を紹介



一括入稿機能

ユーザー

名前	所属
飯塚	横浜店
角田	渋谷店
豊本	渋谷店

店舗

店名	割当先
横浜店	神奈川
渋谷店	東京
中野店	東京

部門

部門名	親部門
神奈川	関東
東京	関東
関東	東日本

目次

1. 背景

   SaaSのエンタープライズ対応と  Excel一括入稿機能の重要性
2. 課題

   表形式データの検証にて

   正確かつ効率的にエラーを

   フィードバックするには？
3. fp-tsによるエラー合成
4. fp-tsのチーム活用

背景

SaaSのエンタープライズ対応とファイル一括入稿機能の重要性

バーティカル SaaS (Software as a Service) の発展と変化

医療や建築や物流などの領域でもDXが進んでいる

バーティカルSaaS業界が発展するにつれ
要求される機能や品質も変化している

個人・中小企業向け

- 差別化につながる先進的な機能と品質
- 安心して長期利用できるランニングコスト
- 眼の前の業務をすぐに楽にできる導入コスト

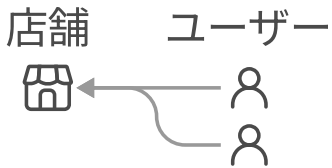
大企業・行政向け

- 上場企業の監査にも安心して対応できる
  セキュリティ性やコンプライアンス
- 組織管理
  大規模で階層的な組織を
  効率的かつ柔軟に管理できる認可

SaaSの組織管理

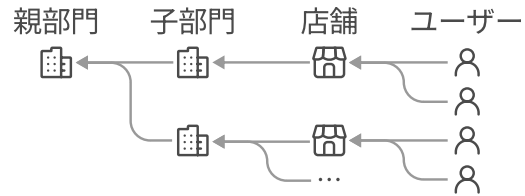
個人・中小企業向け

データは 少量 で構造も シンプル



大企業・行政向け

データが 大量 で 階層的



GUI から登録したい

ユーザー

名前
田中

生年月日
97/06/01

所属
新宿店

店舗

店名
新宿店

割当先
東京

ファイル から一括入稿したい

ユーザー

名前	生年月日	所属
山田	97/06/01	新宿店
田中	54/10/12	渋谷店
山本	70/10/01	中野店

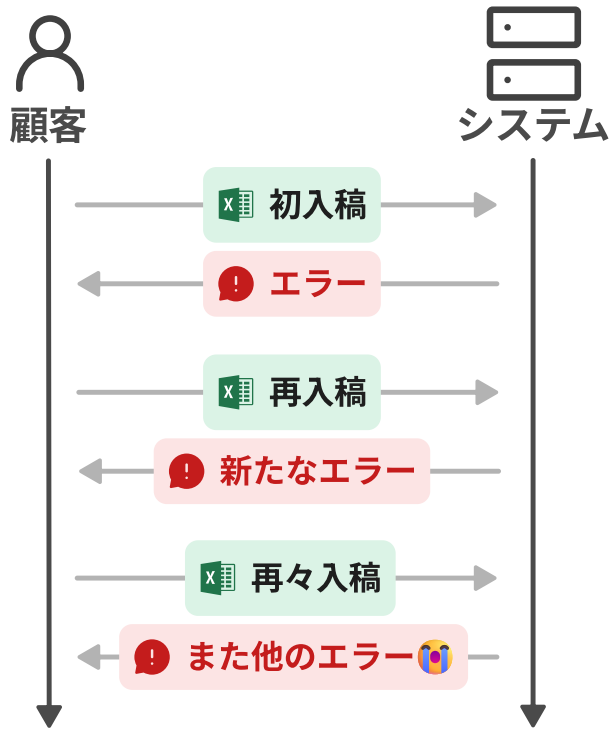
店舗

店名	割当先
新宿店	東京
渋谷店	東京
中野店	東京

部門

部門名	親部門
東京	関東
神奈川	関東
関東	東日本

顧客 から見たファイル一括入稿のつらさ



不明瞭なフィードバック

エラーの原因が判別しにくい
➡️ 修正作業は多大な時間と労力を伴う

- 修正すべき箇所(セル)が不明瞭
- 修正すべき理由が不明瞭

  データ型？使用不能文字？値の重複？

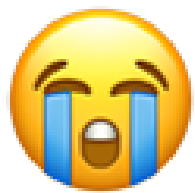
  他のシートと依存関係がある場合は特に難しい

繰り返される再入稿

全てのエラーが一度に返却されない
➡️ 何度も修正と再入稿を繰り返す

課題①

表形式データの検証にて正確 かつ 効率的 にエラーをフィードバックするには？



表形式データの検証の3ステップ

セルをパース

データ型やフォーマットを検証
例) 不正な日付、絵文字の混入

行を検証

行の重複や参照関係を検証
例) IDの重複チェック

表を検証

他の表との依存関係などを検証
例) 店舗IDの存在チェック

表形式データ検証の課題

表形式データのエラー処理の実例

セルをパース
1行目がエラー
不正な日付セル

ユーザーシートのセル[][]

2	山田	24/05/**32**	22
2	田中	97/06/01	66
3	山本	70/10/01	999

行を検証
2行目がエラー
IDが1行目と重複

ユーザーシートの行[]

ID	名前	生年月日	店舗ID
2	山田	24/05/32	22
2	田中	97/06/01	66
3	山本	70/10/01	999

シートを検証
3行目がエラー
店舗IDが店舗
シートに存在しない

ユーザーシート

ID	名前	生年月日	店舗ID
2	山田	24/05/32	22
2	田中	97/06/01	66
3	山本	70/10/01	**999**

店舗

ID	...
22	
44	
66	

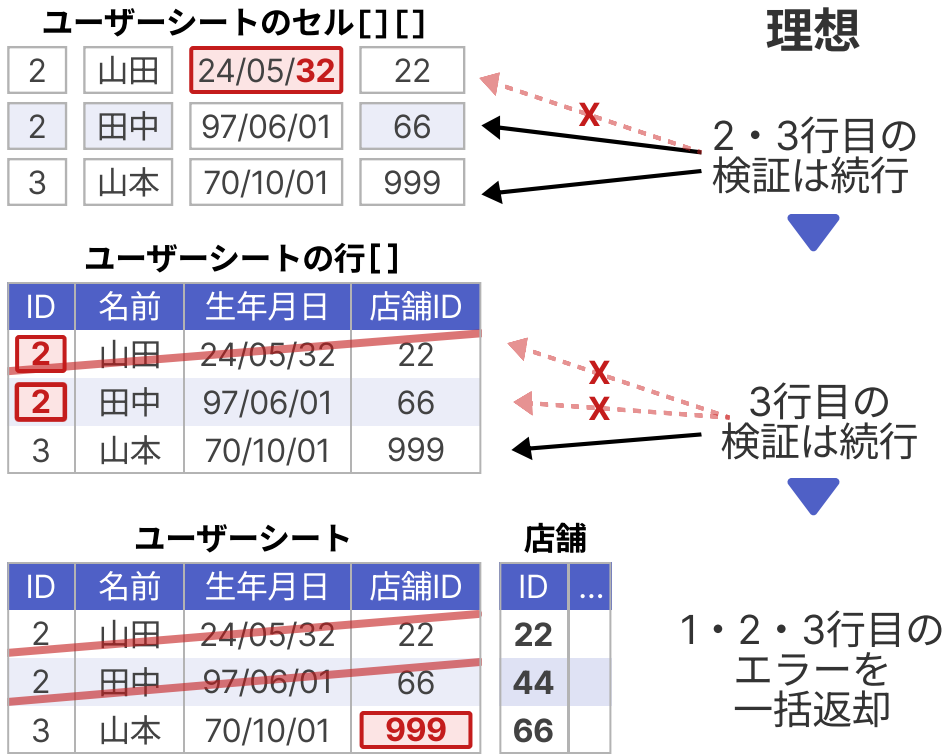
表形式データ検証の課題

表形式データのエラー処理の理想

セルをパース
1行目がエラー
不正な日付セル

行を検証
2行目がエラー
IDが1行目と重複

シートを検証
3行目がエラー
店舗IDが店舗
シートに存在しない



表形式データ検証の課題

表形式データのエラー処理のあるある

セルをパース
1行目がエラー
不正な日付セル

行を検証
2行目がエラー
IDが1行目と重複

シートを検証
3行目がエラー
店舗IDが店舗
シートに存在しない

ユーザーシートのセル[][]

2	山田	24/05/**32**	22
2	田中	97/06/01	66
3	山本	70/10/01	999

ユーザーシートの行[]

ID	名前	生年月日	店舗ID
2	山田	24/05/32	22
2	田中	97/06/01	66
3	山本	70/10/01	999

ユーザーシート

ID	名前	生年月日	店舗ID
2	山田	24/05/32	22
2	田中	97/06/01	66
3	山本	70/10/01	999

店舗

ID	...
22	
44	
66	

理想

2・3行目の
検証は続行

3行目の
検証は続行

1・2・3行目の
エラーを
一括返却

ナイーブな実装

例外を発生
させ終了

**ユーザーは再入稿
しないと他のエラー
に気付けない**

解決策

fp-tsによるエラー合成

TypeScriptのエラー処理

```
class ParseError extends Error {
  readonly row: number;
  constructor(row: number, msg: string) {
    super(msg);
    this.name = "ParseError";
    this.row = row;
  }
}
```

ユーザー定義エラーを投げる

`Error` オブジェクトを拡張し
様々な情報を追加する

`instanceof` で型を絞り込める

```
type Either<E, A> = Left<E> | Right<A>;

type Left<T> =
  Readonly<{ _tag: 'Left'; left: T; }>;

type Right<A> =
  Readonly<{ _tag: 'Right'; right: A; }>;
```

Either型 (Result型) で返す

Discriminated Union
判別可能な直和型で `Left` ・ `Right` の
いずれかを取る値を表現

`Left` で失敗、`Right` で成功を表現し
例外を投げずに結果を返す

TypeScriptのエラー処理

例外を投げる場合の悩み😭

複数のエラーを同時に伝搬しづらい

```ts
const parseRow = (cells: string[]) => ({
  id: parseUserId(cells[0]),
  name: parseUsername(cells[1]),
  birthday: parseBirthday(cell[2]),
});
```

例外が発生

不正な値

不正な値

不正な値が検証されずユーザーへフィードバックされない

`parseUserId` が例外を投げてしまうと
他のセルの検証エラーをクライアントへ
返せないまま処理が中断

```ts
const errors: Error[] = [];
let id: number;
try {
  id = parseUserId(cells[0]);
} catch(e) {
  errors.push(e);
}
```

try...catch文で拾って
配列に詰めていけば不可能ではない

しかし、流石にこれはつらい😭

Either型(Result型)

```ts
type Either<E, A> = Left<E> | Right<A>;

type Left<T> =
  Readonly<{ _tag: 'Left'; left: T; }>;

type Right<A> =
  Readonly<{ _tag: 'Right'; right: A; }>;
```

判別可能なユニオン型を用いて
`Left` と `Right` のどちらかを取る値を表す

エラー処理に使う場合

`Left` の場合をエラー
`Right` の場合を成功と表現できる

TypeScriptのエラー処理

Either型(Result型)で返す

```ts
const isRight = <E, A>(e: Either<E, A>) => e.tag === 'Right';
const isLeft = <E, A>(e: Either<E, A>) => e.tag === 'Left';

const right = <A>(val: A): Right<A> => ({ tag: 'Right', left: val });
const left = <E>(val: E): Left<E> => ({ tag: 'Left', left: val });

const parseUsername = (v: string): Either<ParseError, string> =>
  v.length > 0 && v.length < 16 ? right(v) : left(new ParserError());

const parseRow = (cells: string[]) => ({
  id: parseUserId(cells[0]),
  name: parseUsername(cells[1]),
});

console.log(parseRow([NaN, '😭']));
```

エラーが発生

エラーが発生

👋 両方のセルのエラーを伝搬できる

```ts
{
  id: { left: new ParseError() },
  name: { left: new ParseError() },
}
```

それぞれのセルを行へ合成したい

エラーとなるセルが含まれる場合

```
{
  id: left([new ParseError(1, 'ID')]),
  name: left([new ParseError(1, '名前')]),
  storeId: right(22),
};
```

→

```
left([
  new ParseError(1, 'ID'),
  new ParseError(1, '名前'),
]);
```

エラーとなるセルが無い場合

```
{
  id: right(1),
  name: right('田中'),
  storeId: right(22),
};
```

→

```
right({
  id: 1,
  name: '田中',
  storeId: 22,
});
```

それぞれのセルを行へ合成したい

```
const errs: ParseError[] = [];
if (isLeft(id)) {
 errs.push(id.left);
}
if (isLeft(name)) {
  errs.push(name.left);
}

if (errs) {
  return left(errs);
}

assert(isRight(id));
assert(isRight(name));
return right({
  id: id.right,
  name: name.right;
});
```

自前実装

いずれかのセルが `Left` の場合

エラーを取り出して配列に詰め
`Left<ParseError[]>` として返す

全てのセルが `Right` の場合

`Right<{ id: number, ... }>` として返す

ライブラリに頼りたい

😭 自分でやりたくはない

それぞれのセルを行へ合成したい

fp-tsによるオブジェクトのエラー合成

```ts
import * as AP from 'fp-ts/Apply';
import * as A from 'fp-ts/Array';
import * as E from 'fp-ts/Either';

// これを合成したい
const cells = {
  id: left([new ParseError(1, 'ID')]),
  name: left([new ParseError(1, '名前')]),
  storeId: right(22),
};

// 1. `Left` の `ParseError[]` を結合する関数 `ap` を定義
const ap = E.getApplicativeValidation(
  A.getSemigroup<string>(),
);

// 2. 先ほど定義した `ap` を用いて合成
const row = AP.sequenceS(ap)(cells);
```

👋 オブジェクトの値に含まれるEitherを合成できた

```ts
left([
  new ParseError(1, 'ID'),
  new ParseError(1, '名前'),
]);
```

それぞれの行をシートへ合成したい

エラーとなる行が含まれる場合

```
const rows = [
  left([ new ParseError(1, 'ID') ]),
  left([ new ParseError(2, '名前') ]),
  right({ id: 3, name: '田中', storeId: 66 }),
];
```

```
const sheet = left([
  new ParseError(1, 'ID'),
  new ParseError(2, '名前'),
]);
```

エラーとなる行が無い場合

```
const rows = [
  right({ id: 1, name: '長野', storeId: 22 }),
  right({ id: 2, name: '大崎', storeId: 44 }),
  right({ id: 3, name: '田中', storeId: 66 }),
];
```

```
const sheet = right([
  { id: 1, name: '長野', storeId: 22 },
  { id: 2, name: '大崎', storeId: 44 },
  { id: 3, name: '田中', storeId: 66 },
]);
```

fp-tsによるエラー合成

配列のエラー合成

```ts
const rows = [
  left([new ParseError(1)]),
  right({
    id: 2,
    name: '田中',
  }),
  left([new ParseError(3)]),
];

// 行へ合成
const sheet = A.sequence(ap)(rows);
```

👋 Eitherの配列を合成できた

```ts
left([
  new ParseError(1),
  new ParseError(3),
]);
```

fp-tsによるエラー合成

セル→行→シートまで一気通貫で合成

```
import { pipe } from 'fp-ts/function';
const raw = [
  [right(1), right('田中'), right(22)],
  [right(2), right('山田'), right(33)],
];

pipe(
  raw,
  // 1. 行へ合成
  A.map(AP.sequenceS(ap)),
  // 2. シートへ合成
  A.sequence(ap),
);
```

1. 各セルで構成されるオブジェクト ➡️ 行へ
2. 行の配列 ➡️ シートへ

合成処理を簡潔に表現できる😘

顧客への提供価値に直結

冒頭でも述べた通り表形式データの検証は
極力 一度に 全てのエラーを返すことが
顧客体験のために重要

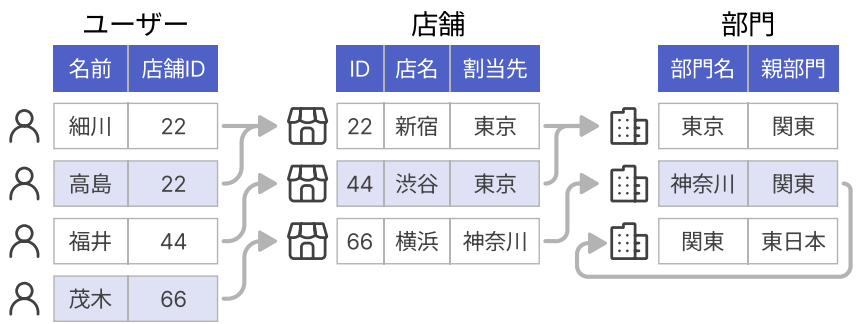
課題②

シート間の依存関係を型によって表現したい

表形式データ検証の課題

依存関係の解決

組織の階層構造を表現するために、ある行が他シートの行を参照する
例) ユーザーの所属を表現するため店舗シートを参照



😱検証済みの行を参照したいのに
未検証の行を参照してしまった

➡️ 型検査で未然に防ぎたい

公称型の活用

公称型を利用して検査の関門を一つに絞る

```ts
import * as E from 'fp-ts/Either';
import { Newtype, iso } from 'newtype-ts';

type Username = Newtype<{ readonly Username: unique symbol }, string>;
const bad: Username = '田中'; // ちゃんと型検査で落としてくれる
```

```
Type 'string' is not assignable to type 'Username'.
```

```ts
const Username = {
  // 「この関数を通らないとUsername型にできない」という状態へ
  parse: (v: string): E.Either<ParseError, Username> =>
    v.length > 0 && v.length < 16
      ? iso<Username>.wrap(v)
      : new ParseError('文字列長が誤っています'),
} as const;
```

fp-tsのチーム活用

fp-tsの難しさ

fp-tsのコンセプト

関数型プログラミングのためのデータ型や型クラスなどの抽象化を提供する

> fp-ts provides developers with popular patterns and reliable abstractions
> from typed functional languages in TypeScript.
> – https://gcanti.github.io/fp-ts/

fp-ts採用後に直面した課題

オンボーディングのコストが高い😱

- 関数型プログラミングへの一定の知識が必要
- 日本語の情報が少ない
- 業務での実用例の解説が少ない

  抽象度が高すぎて使い方が分からない

チームでfp-tsを利用するために

オンボーディングコストの削減

私自身 ~~学生時代にHaskellの講義の単位を落としかけたこともあり~~
関数型プログラミングの知見が少なくチーム参画時にかなり苦労した

中長期的に持続して運用できるプロダクトにするためにオンボーディングコストを低減したい

スコープを限定する

オンボーディングが必要なコンテキスト自体を削減し
その代わりにコンテンツの密度を高める

短期集中

高い密度で反復して学習できるように
メンバー参画時に集中してオンボーディングする

チームでfp-tsを利用するために①

スコープを限定する

fp-tsは前述の `Either` のみならず様々な抽象化を提供する
まず小さく始めるために、利用するものを限定するとよい

関数

- `pipe` `flow`
  合成関数

データ型

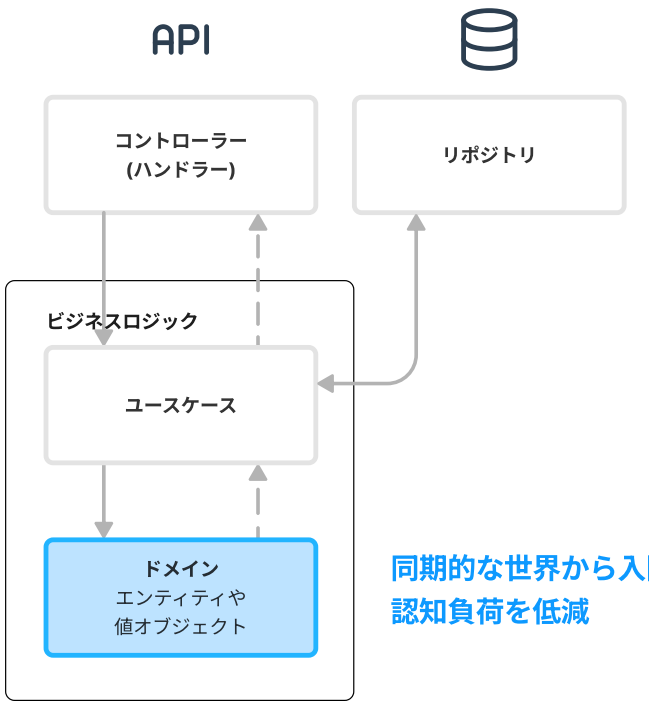
- `Readonly(Array|Map)`
  readonlyの配列やマップ
- `NonEmptyArray`
  空ではない配列
- `Task` `TaskEither`
  非同期処理の抽象化

型クラス

- `Eq`
  等価性を表現
- `Ord`
  比較を表現
- `Bounded`
  上限と下限を表現

チームでfp-tsを利用するために②

短期集中型 ペアプロ・モブプロ



1日1回15-30分ほどペアプロ・モブプロをする

- タスクをあらかじめ決めておく
- まずは同期的な世界から入門し
  認知負荷を低減させる

チームでfp-tsを利用するために③

社内向けレシピ集とプレイグラウンド

実務での具体的な利用例を示す

- `O.fromPredicate` と `RA.filterMap` を用いて重複するコードを持つ部門だけを抽出する
- `Eq.struct` を用いてユーザーが同姓同名か判定する

まとめ

ある程度成長したSaaSが直面する新たな要求

大企業・政府向けにはセキュリティやコンプラ対応が重要
それに加えて大規模な組織を管理できる機能が求められる

複雑な検証とエラー処理には `fp-ts` は便利

📊Excel一括入稿機能にある表形式データの検証を例示

- エラー合成
- 検証前後の文脈の明示
- 同一性の担保

`fp-ts` をチームで活用するために

利用する機能をあらかじめ限定した上で
具体例を含めた社内レシピを用意し、ペアプロ・モブプロで密度の高いオンボーディングを