

Vue.js Options API の課題と Composition APIのメリット

はじめに

背景

- 前職でVue 2のOptions APIを使用
- Vue 3のComposition API導入の理由に疑問

目的

- Options APIの実際の課題を明らかにする
- Composition APIによる解決策を具体例で解説
- Vueの設計思想の進化を理解する

目次

1. Options APIとは
2. 課題1: コードの散在性
3. 課題2: 機能の再利用性
4. 課題3: TypeScriptとの統合
5. Composition APIによる解決
6. まとめ

Options APIとは

- Vue.jsの伝統的なコンポーネント記述方法
- `data`, `methods`, `computed` などのオプションでコンポーネントを構成
- Vue 2から使われている馴染み深いAPI

```
export default {  
  data() { return { /* データ */ } },  
  methods: { /* メソッド */ },  
  computed: { /* 算出プロパティ */ },  
  etc...  
}
```

課題1: コードの散在性

- 関連する機能のコードが異なるオプションブロックに分散
- コードの把握・追跡が困難になる



課題2: 機能の再利用性

同じ機能を複数のコンポーネントで実装したい場合OptionsAPIだと下記になる

```
// 対策1: ユーティリティ関数の分離(validation.js)
export function validateEmail(email) {
  return !email.includes('@') ? 'メールアドレスが無効です' : '';
}

// 対策2: ミックスインの利用(validation-mixin.js)
export const validationMixin = {
  methods: {
    validateEmail(email) {
      return !email.includes('@') ? 'メールアドレスが無効です' : '';
    }
  }
}
```

既存対策の限界

ユーティリティ関数:

- 状態（data）の共有ができない
- リアクティブな連携が複雑

ミックスイン:

- 名前空間の衝突リスク
- コードの出所が不明確になりやすい
- 複数ミックスインの相互作用が複雑

ミックスインの問題

```
export default {  
  // どのミックスインが何を提供しているか把握しづらい  
  mixins: [validationMixin, formMixin, userMixin],  
  methods: {  
    // コンポーネント独自の実装があるとミックスインとの関係が複雑に  
    validateEmail() { /* オーバーライドのロジック */ }  
  }  
}
```


余談：ミックスインの優先ルール

- data: オブジェクト同士がマージされる
- methods/computed: コンポーネント側が優先され、ミックスインを上書き
 - コンポーネント側に同名のものが存在する場合コンポーネント側を利用する
 - `mixins: [mixinA, mixinB]` では mixinB が優先される
- ライフサイクルフック: すべて実行される
 - `mixins: [mixinA, mixinB]` の場合の実行順序
mixinA のフック → mixinB のフック → コンポーネントのフック

課題3: TypeScriptとの相性の悪さ

TypeScriptの利点（自動補完、型推論）を最大限活用できない

```
export default {  
  data() {  
    return {  
      count: 0,  
      user: { name: 'タロウ' }  
    }  
  },  
  
  methods: {  
    increment() {  
      this.count++; // OK  
      this.user.age++; // エラーにならないが実行時エラー（ageは存在しない）  
      this.nonExistentMethod(); // エラーにならないが実行時エラー  
    }  
  }  
}
```

課題3: TypeScriptとの相性の悪さ

Vue.extend を利用すれば型定義は可能ではある(らしい)が面倒

```
interface ComponentData {  
  count: number;  
  user: { name: string };  
}  
  
export default Vue.extend({  
  data(): ComponentData {  
    return {  
      count: 0,  
      user: { name: 'タロウ' }  
    }  
  },  
  methods: {  
    increment() {  
      this.count++;  
      // this.user.age++; // エラーになる  
      // this.nonExistentMethod(); // エラーになる  
    }  
  }  
});
```

5. Composition APIによる解決

- 関連するロジックを一箇所にまとめられるようになった
- コンポーネント間での機能再利用が容易になった
- TypeScriptとの統合が自然で効率的になった

関連するロジックを一箇所にまとめられる

Options API



Composition API



- <https://github.com/vuejs/rfcs/blob/master/active-rfcs/0013-composition-api.md>

再利用可能なコンポジション関数

```
// useValidation.js - 再利用可能なコンポジション関数
import { ref } from 'vue';

export function useValidation(initialValue = '') {
  const value = ref(initialValue);
  const error = ref('');

  const validateEmail = () => {
    error.value = !value.value.includes('@')
      ? 'メールアドレスが無効です' : '';
  };

  return { value, error, validateEmail };
}
```

名前の衝突回避

```
import { useEmail } from './useEmail';
import { usePassword } from './usePassword';
export default {
  setup() {
    // 分割代入で名前変更
    const { value: email, validate: validateEmail } = useEmail();
    const { value: password, validate: validatePassword } = usePassword();

    return { email, password, validateEmail, validatePassword };
  }
}
```

TypeScriptとの自然な統合

```
// 標準的なTypeScript機能がそのまま使え、型推論も機能するようになる
import { ref, Ref } from 'vue';

interface ValidationResult {
  value: Ref<string>;
  error: Ref<string>;
  validate: () => void;
}

export function useValidation(initialValue = ''): ValidationResult {
  const value = ref(initialValue);
  const error = ref('');

  const validate = () => {
    error.value = !value.value.includes('@')
      ? 'メールアドレスが無効です' : '';
  };

  return { value, error, validate };
}
```


6. まとめ

Options APIの課題:

- 関連機能の分散による保守性低下
- 機能の再利用方法に限界がある
- TypeScriptとの統合は「可能」だが「現実的でない」

Composition APIのメリット:

- 関連ロジックをまとめて保守性向上
- 状態とロジックを一体として再利用可能
- TypeScriptとの自然で効率的な統合

ご清聴ありがとうございました！