国際的技術フレームワーク『OpenFisca』の活用可能性の検討レポート

2025 年 4 月 16 日

一般財団法人 GovTech 東京

本計画書案は令和 6 年度「行政課題の解消に向けた新サービス創出に係る調査委託」事業において、委託先である一般社団法人防窮研究所による調査・分析内容をもとに作成されました。

1 章 OpenFisca の特徴

1.1. OpenFisca フレームワークの概要

Rule as Code の実践

本稿の主題である「OpenFisca」という技術は、法や制度の定義を日本語のような言語ではなくプログラミング言語により記述することを可能にするものです。

法律とプログラムは、どちらも綿密な設計・記述、厳密な記述・処理が求められる点で共通する分野であり、似通った性質を持つと言えます。通常、日本語などの自然言語で記述される法律を、プログラミング言語によって記述することにより効率性・正確性・堅牢性などの向上を図るアプローチを「Rule as Code」と呼びます。

Rule as Code によって得られるメリットとしては、主に下記のようなものが挙げられます。

- 規則や制度などのルールをプログラムとして記述し、コンピュータがそのルールを解釈できるようにすることで、正確かつ高速に計算が行える。
- 他のシステムからの読み取りが行いやすい形式になることで、検索性が高まる。
- プログラムの設計の考慮漏れを防ぐ技術が法律にも応用できるようになり、堅牢な制度設計が可能となる。

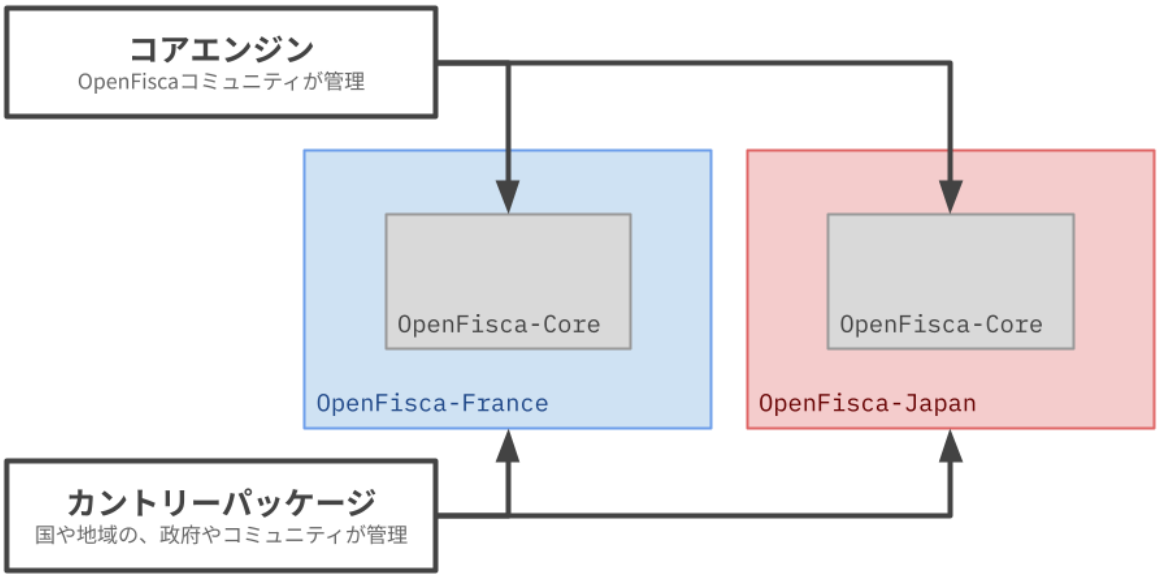
OpenFisca は Rule as Code の実践として、特に税制や給付金などに関する法や規則の記述を得意とする技術・フレームワークです。
例えば「児童手当」制度を OpenFisca で記述すると、このプログラムに子供の年齢・学年・世帯構成を入力して、手当の月額を出力として得ることができます。

OpenFisca は数値計算を伴う制度に関して汎用的に扱えるように設計されています。
これにより、制度のシステムが分散してしまうことを防ぎ、記述・管理・処理を一貫して行えるようになっています。

OpenFisca のコアエンジンと各国版 OpenFisca

OpenFisca プログラムは OpenFisca-Core と呼ばれるコアエンジン（基盤技術）と、OpenFisca-France や、OpenFisca-Japan のような、国や地域ごとに個別に開発される「カントリーパッケージ[1]」の 2 つの部分に分けられます（図 1）。

[1]ソフトウェア開発の文脈における「パッケージ」とは、コードや画像などのソフトウェアの資産を、実行できるようにまとめたもの。



《図 1 OpenFisca コアエンジンとカントリーパッケージの関係性》

具体的な法や制度に関するロジックはカントリーパッケージに実装することとなっており、また後述する API 機能やテスト機能のような、国や地域の間で共通する部分についてはコアエンジンに実装することとなっています。
2 つのロジックを分離して開発することで、それぞれの管理主体が関心のある部分に集中しやすくなっています。

Open Source Software（OSS）

上述の OpenFisca-Core や、カントリーパッケージの仕様や詳細なコードは全て公開されており、これらのリソースは GNU Affero General Public License（AGPL）と呼ばれるライセンスの元利用することができます。

このような誰でも中身を見ることができ、派生や再配布も自由に行なって良いとするソフトウェアは Open Source Software（OSS）と呼ばれ、OpenFisca もそのうちの一つです。

数ある OSS 向けライセンスの中において、AGPL の主な特徴には下記のようなものが挙げられます。

- 誰でもソフトウェアの使用・改変・再配布が許可される。また、ネットワーク越しに機能を提供することも許可される。
- 改変の上、ソフトウェアを再配布、またはその機能をネットワークに向けて公開する場合、改変後のソースコードも公開しなければならない。
- 改変後のソフトウェアは、同じライセンスを継承しなければならない。

このライセンスの採用の裏には、OpenFisca の持つ強い公共性を維持・拡大し続けられるようにする意図があると説明されています。（OpenFisca 公式ドキュメントサイト）

OpenFisca に基づいて構築された価値あるソフトウェアが、独占されることなく広く公開されるよう促すことで、公平性・透明性を高く保ちながら OpenFisca を取り巻くエコシステムへの還元につながるようなライセンスを採用しています。

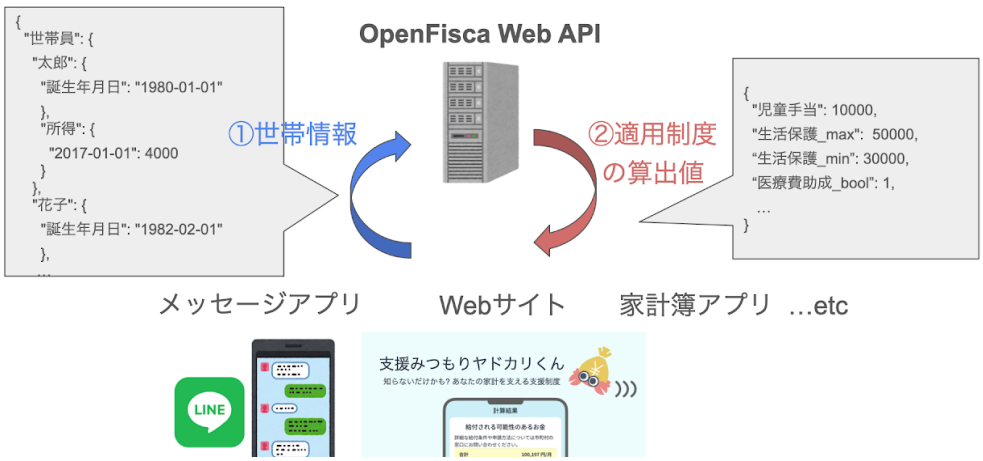
さらに、このプログラムの運営管理はソフトウェア開発者向けプラットフォームである GitHub で公に行われており、誰でも Issue（バグや改善案の報告）や Pull Request（具体的なコード変更の提案）の作成などを通して、開発に参加することができるようになっています。

OpenFisca 派生ソフトの開発によるエコシステムへの貢献に留まらず、OpenFisca 自身に対する変更提案も広く受け付けることで、あらゆる人にとって開かれた運営の仕組みが整えられている。

数値計算に特化した API

OpenFisca はプログラミング言語 Python の数値計算ライブラリ[2]である「NumPy」を核として構築されています。
Numpy は大規模データを効率的に処理するための「ベクトル計算」や「行列計算」、またそのためのデータ構造を提供しており、機械学習の分野などでも広く活用されています。
NumPy を採用することにより、国や地域レベルでのシミュレーションや、複数のシナリオでの並列の見積りなど、高負荷なニーズにも OpenFisca は対応できるようになっています。

また、OpenFisca は外部のプログラムからの再利用性にも優れています。
OpenFisca で法や規則を記述すると Web API[3] が自動的に使えるようになり、他のシステムはこの API を呼び出して機能を利用することが簡単にできるようになっています（図 2）。



≪図 2 OpenFisca Web API を他システムが利用する様子（proj-inclusive 公式サイトから引用）≫

---

[2]「 ライブラリ」とは、特定の目的のための再利用可能なソフトウェアの部品である。ソフトウェア開発においてライブラリの活用は開発の効率化、信頼性の向上に大きく寄与し、またそのソフトウェアの使い勝手も左右することがあります。

[3] API（Application Programing Interface）とはあるシステムを他のシステムから利用するための仕様のこと。システムのための窓口のようなものであり、システム連携には欠かせないものです。Web サイトの閲覧などに用いられる技術である HTTP によりシステム連携する API の場合、Web API とも呼ばれています。

政府や自治体など公的な組織が OpenFisca の Web API を市民に向けて提供することで、制度検索システムや制度シミュレーションといった多様なアプリケーションの開発も効率的に行えるようになり、また計算結果の信頼性も高めることができます。

なおライセンスに関して、OpenFisca の派生ソフトウェアは AGPL に則りソースコードの開示義務やライセンスの継承義務が生じるが、単に Web API を利用しているだけのアプリケーションはこの限りではなく、必ずしもソースコードを開示する必要はありません。

保守運用しやすい仕組み

法律、ソフトウェア、そのどちらとも、内容に抜けや漏れがないか、古くなっていないかなどを継続的に確認して保守していく必要があります。
OpenFisca にはこれを支援する仕組みも備わっており、そのうちの一つに「テスト」機能が挙げられます。
「テスト」機能とは、OpenFisca で記述された制度に関して、具体的な入力と出力をセットで記述しておくことにより、内容の誤りがないことを自動的に検算できるようにするものです。

OpenFisca で記述された制度のプログラムは、世帯やその構成員の属性（年齢や収入など）を入力すると、その世帯や人の手当額や税額が出力されます。
手計算の結果など、確からしい入力と出力の組を YAML 形式[4]であらかじめ列挙しておくことで、その入力に対する出力が期待通りであること、ロジックに誤りがないことを機械的・自動的に検算することができます。

標準的には、下記のように入力としての世帯情報と期待する出力としての金額を記載することで、テスト項目が表現できます。

≪テスト項目の yaml ファイル（OpenFisca-Japan の「子育て支援制度テスト」から引用）≫

[4] ファイル形式の一つ。単純なテキストファイルで「メモ帳」などで編集ができる点で CSV 形式と似た性質を持っています。CSV は単純な表形式のデータに特化しているのに対し、YAML は入れ子を含むような複雑な構造のデータも扱える形式であることを特徴としています。

```yaml
- name: テスト ID=2
  period: '2024-10-01'
  input:
    世帯:
      親一覧:
      - 親 1
      子一覧:
      - 第 1 子
      児童手当の控除後世帯高所得:
        '2024-10-01': 0
    世帯員:
      親 1:
        誕生年月日: '1994-09-01'
      第 1 子:
        誕生年月日: '2022-09-01'
  output:
    世帯:
      児童手当:
        '2024-10-01': 15000
```

法改正により新しい算出方法が加えられたような場合には、その具体的な入力と出力のセットを追記することで、新しいロジックにも対応したプログラムが書けているかどうかを確実にしながら更新作業を行うことができます。

この手法は通常のソフトウェア開発で確立され、広く取り入れられている方法であるが、Rule as Code により法律の分野にも応用ができるようになりました。OpenFisca においては、さらにシンプルに管理・保守できるよう、プログラムを書くことなくテストが記述できるようになっています。

また、OpenFisca では控除額や税率といった法定の値の改定にも容易に追従できる「Parameters（パラメータ）」と呼ばれる機能があります。
これは、上に挙げたような控除額などの定数を、テストと同様に YAML 形式で管理できる機能です。
Parameters により、エンジニアではない者も安心かつ安全に、法改正に対応した変更を行うことができるようになっています。

パラメータ名やロジック中の変数名は英語に翻訳して定義する必要はなく、日本語などネイティブな言語を用いることが推奨されています。（OpenFisca 公式ドキュメント）
この点も、コミュニケーションを円滑に保ち、運用のためのコストを押し下げる特徴と言えます。

1.2. 制度記述方法の特徴

1.2.1. 国外事例の紹介

市民のための制度検索システム（フランス）

フランス政府により、プロフィールを入力することで自身に関する制度や権利を検索できるアプリケーションが提供されています。

- Aides jeunes - 若者向け。国や地方の 1000 以上の制度の中から適するものを推薦。
- Mes droits sociaux - 一般向け。60 程度の制度の中から適するものを推薦。

国会議員向けの制度シミュレータ（フランス）

フランス国会により提供されている高機能な税制や支援金のシミュレータとして LexImpact があります。

例えば「この税制改革が、何世帯に影響が及ぶ見込みなのか、どの程度手取り額に影響するか」といった具体的なシチュエーションでの試算を、専門の技術や知識がなくとも実施することを可能にしている。

税制の見積もりアプリ（アメリカ・イギリス）

PolicyEngine は、アメリカやイギリスの税制に基づき、税金や給付金の計算を行えるアプリケーションです。
世帯構成や収入状況などを入力することで、税額や給付金について何が対象となっているかを知ることができるようになっています。

さらに、LexImpact と似た税制改革が及ぼす影響のシミュレーション機能も提供しています。

再利用可能な Rule as Code のための共通基盤（フランス）

「OpenFisca-France」は、フランスの社会給付や税金など、社会財政システムのロジックを OpenFisca によって表現（モデル化）したものです。フランス地域結束庁（ANCT）を中心とした OSS コミュニティによりメンテナンスされており、その機能はフランスの公共データプラットフォームの一部という位置付けで、API としても公開され、利用できるようになっています。

先述の Aides Jeunes や LexImpact といったアプリケーションのうち社会財政システムに関するロジックはこの OpenFisca-France から再利用される形でも実現しています。
個々のアプリケーション開発者が正確に法や規則を解釈する必要なく、正確なシミュレーション結果を利用できるような役割分担が行えていると言えます。

1.2.2. 国内事例の紹介

一般社団法人 防窮研究所は、有志市民団体 proj-inclusive と連携して制度検索 Web アプリ「支援みつもりヤドカリくん」を開発・運営しています。

利用者は居住地・所得額・世帯構成などを入力することで、20 以上の制度をもとに受け取れる可能性のある手当を知ることができ、さらにその金額も見積もることができます。
OpenFisca が手当の計算を得意とする特徴を活かし、具体的な金額を見せることで公的な支援を必要としている人々に届けることを目指しているプロジェクトです。

当プロジェクトのチームはフランスの Rule as Code 共通基盤 OpenFisca-France と同じ位置付けにあたる OpenFisca-Japan の開発・運営も「支援みつもりヤドカリくん」と合わせて行っており、OpenFisca ならびに Rule as Code を先進的に実践した例です。

2025 年 3 月の執筆時点では、その他日本国内制度向けの OpenFisca カントリーパッケージ、OpenFisca 利用アプリケーションの事例は見当たりませんでした。

2 章 日本の制度データにおける課題

2.1 日本の制度データの性質

各種制度の概要や支給条件等について説明した情報が、官公庁等によって web サイト上に公開されています。これらの情報は、第一に住民が制度を理解しやすくすることを目的としていると考えられます。一方、Rule as Code の文脈からこれらの情報源をデータとして活用する（以下「制度データ」とする）ことを考えた場合、以下のような性質が挙げられます。

レイアウトに基づいた説明

web サイト中の 1 ページに情報を載せているという形式のため、運営主体（自治体等）のページレイアウト、構成に沿った形で掲載しています。

人が読むことのみを想定している

制度理解のために住民が読むことを想定した形式になっています。金額表や金額の計算式等一部構造化された情報も含まれているが、説明文と混ざっており解釈には目視確認が必要です。

自治体等各官公庁が個別に作成している

web サイトのレイアウトは運営主体によって異なるため、制度データの内容も運営主体それぞれで別個のものになっています。同一制度の説明に対しても、内容の取捨選択や説明文の言い回しは各 web サイトによって異なります。

2.2 性質から考えられる日本の制度データの課題

2.2.1 給付金制度における、日本の制度データと OpenFisca プログラムのギャップ

制度データを Rule as Code として利用する一例として、制度データをもとに OpenFisca で制度記述を行った際に生じた課題の例を挙げます。なお、OpenFisca の性質上世帯や個人に対する支援金、貸付等のシミュレーションを主眼においているため、本節ではこれらの制度を中心に取り上げます。

2.2.1.1 構造におけるギャップ

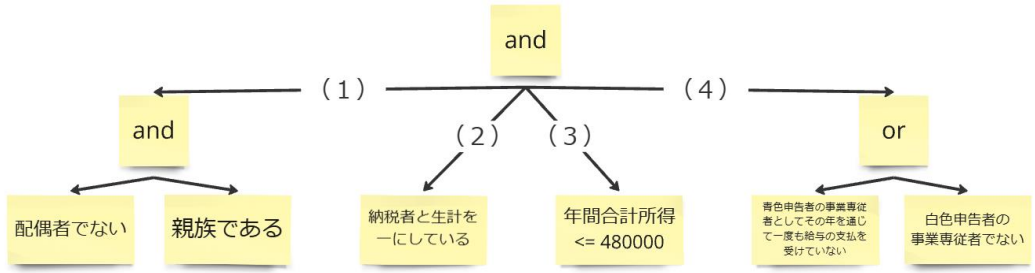
制度データと OpenFisca プログラムを比較した際、以下の違いが存在します。そのため、制度データから直接 OpenFisca プログラムを生成することは現状困難であると考えられます。

支給条件や金額が構造化されていない

OpenFisca をはじめ、Rule as Code に向けた制度をプログラムで記述する際には論理関係の階層構造が重要となります。条件「かつ」「または」等が指し示す範囲や階層構造の解釈を誤ると間違った結果が出力されてしまうため、プログラム生成時にはこれらの情報を正確に扱う必要があります。しかし、制度データの自然言語では構造を機械的に解釈しきれません。
例えば、下図の扶養親族の制度データ（図 3）の論理関係は（図 4）の構造で表されます。一方、接続語「かつ」「または」等を機械的に探すだけでは見落としてしまう箇所も存在します（例：条件（1）の「配偶者以外の親族である」の論理構造は「配偶者でない、かつ親族である」）。

扶養親族とは、次の4つの要件のすべてに当てはまる人です。
（1）配偶者以外の親族である
（2）納税者と生計を一にしている
（3）年間の合計所得金額が48万円以下である
（4）青色申告者の事業専従者としてその年を通じて一度も給与の支払を受けていない、または白色申告者の事業専従者でない

≪図 3 制度データで記述された扶養親族の定義（国税庁 HP より引用、一部簡略化）≫



≪図 4 扶養親族の定義の論理関係を階層構造で表したもの≫

計算方法が自然言語で表現されている

OpenFisca のプログラムには、支援制度中で定義される金額の計算式が必要です。制度データでは金額計算を自然言語による説明のみで定義している場合があるが、数式での表現に比べ OpenFisca プログラム生成に活用されてにくい場合もあります。前述の論理関係同様、自然言語では計算式の構造が把握しづらいためです。

居宅世帯員の第1類基準額を合計し、世帯人員に応じた逓減率を乗じ、世帯人員に応じた第2類基準額を加える。

≪生活保護、生活扶助基準の計算方法（厚生労働省「最低生活費の算出方法」より引用≫

≪上記を記述した数式≫

```
生活扶助基準 = 第 1 類基準額の合計 * 逓減率 + 第 2 類基準額
```

```
def formula(対象世帯, 対象期間, parameters):
    第 1 類基準額 = 対象世帯("生活扶助基準_第 1 類基準額", 対象期間)
    逓減率 = 対象世帯("生活扶助基準_逓減率", 対象期間)
    第 2 類基準額 = 対象世帯("生活扶助基準_第 2 類基準額", 対象期間)
    return 第 1 類基準額 * 逓減率 + 第 2 類基準額
```

≪上記を記述した OpenFisca プログラム≫

支給金額計算式中に登場する別制度をたどれない

支援制度の計算式には別制度の概念が登場することがあります(例:所得控除条件の計算に所得税を参照)。この際、別制度に関する情報のリンクが存在しないと定義を取得することができません。すでに OpenFisca に実装済みである場合を除き、リンク先の制度を別途調査し実装する必要があります。特に管理団体が異なる場合は web サイト自体が独立しており、例えば、地方自治体の web サイトにある条例の制度データから国の法律を参照している場合、用語検索し政府の web サイトを改めて参照する必要があります。

2.2.1.2 内容におけるギャップ

続いて、制度データを直接 OpenFisca に利用せず、制度データを人手で読み解いて OpenFisca のプログラムとして記述する場合を考えます。この場合も、必要な内容が制度データだけでは得られない場合があります。

条件が網羅されていない

web サイトによっては、本文中の記述だけでは条件分岐をすべて網羅していない場合があります(例:児童育成手当における所得制限額が、表の大きさの都合上扶養親族人数 5 人の場合までしか記載されていません。実際には 6 人以上の場合も定義されており、省略せずに記載されている制度データも存在すします(東京都福祉局の pdf))。簡潔さのため一部条件を省略することが考えられるが、その場合 OpenFisca のプログラム上特定の条件節に対して実装が欠落した状態になってしまいます。欠落した条件を見逃すと大きな見積もり誤差が発生したり、予期せぬエラーにより計算自体が停止したりしてしまう可能性があります。

制度改定の際の差分が明確でない

OpenFisca のプログラムを制度改定に合わせて改修する場合、制度の変更箇所を全て再実装する必要があります。制度データ上で更新箇所の差分が明示されていないと、OpenFisca 側の改修漏れが発生する可能性があります。

2.2.2 制度データの計算困難性の由来

給付金制度以外の制度においても、以下の要因から現状制度データを rule as code へ利用することが難しいです。

形式が統一されていない

web ページの HTML 上支給条件、金額と説明文を判別することができず、機械可読な状態ではなくブラウザ上で目視確認する必要があります。また、制度データのレイアウトが運営主体によって異なるため、得たい情報を web ページごとに逐一探す必要があります。

支給金額計算式中に登場する用語の定義が存在しない

「支給金額計算式中に登場する別制度をたどれない」の項とも関連するが、制度中に別制度の概念が登場した際にリンクが無いと参照することができません。人手で読み解く場合であっても、制度情報のリンクは同名の別概念との混同を防ぐ上で有効であると考えられます。

3 章 制度データ利用のケーススタディ：OpenFisca プログラム記述での利用

3.1 過去のトライアルにおけるギャップフィリングの手法

OpenFisca-Japan の開発において、OpenFisca の設計上制度をプログラムで記述する際の制約が存在する。例として、OpenFisca では計算を世帯単位で行うため、企業等を対象とする支援のシミュレーションには不向きです。
また、「支援みつもりヤドカリくん」のユーザー体験を維持するために発生した固有の課題も存在する。「ヤドカリくん」は受けられる可能性のある制度とその金額のシミュレーションにより利用者が窓口へ行き支援に繋がりやすくなることを主眼に置いており、フォーム入力の負担が増えないよう意識しています。フォームの分量を減らすと正確な値が一意に求まらない場合があるため、代わりに可能性のある金額範囲を算出しています。

以下、生じた制約と取った対処法について説明します。

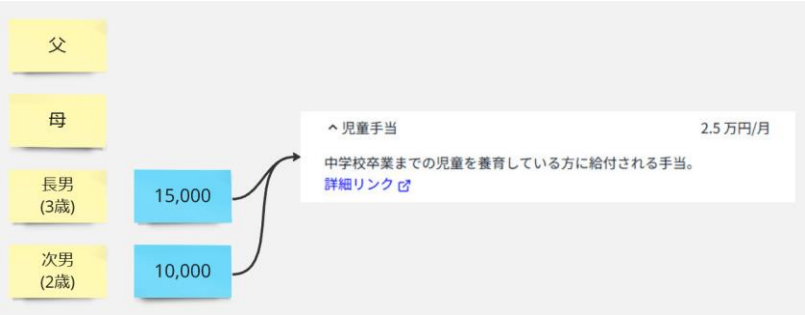
3.1.1 OpenFisca の制約

数値以外の形式で出力できない

- OpenFisca の出力は数値形式であるため、給付額、貸付額等の金額計算以外を扱うことが難しい
    - 例：現物支給、補助サービス等自然言語での説明が必要なもの
- 解決策：「ヤドカリくん」では給付、貸付制度のシミュレーションに特化
    - 金額での出力が難しい一部の対応制度に関しては適用可否のみ計算
        - OpenFisca 側では適用不可の場合に 0，適用可の場合に 1 を出力し、受け取り側で可否に置き換え
        - ただし、条件を計算式で表せない場合は適用可否も算出不可

世帯または個人単位でしか計算できない

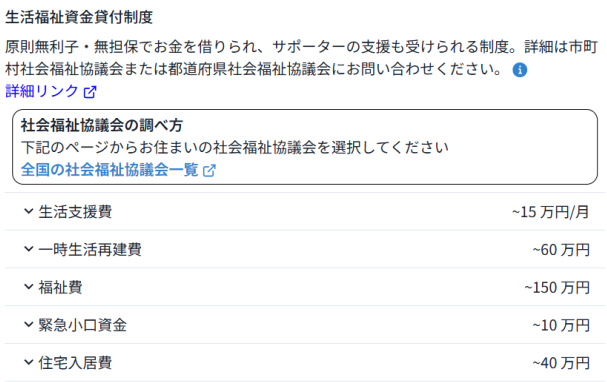
- 解決策：「ヤドカリくん」では世帯、個人に対する支援、貸付のみ対応している（その他の支援制度は対象外）
- 表示形式を揃えるため、個人に対する支援金は全世帯員分の金額を合計し世帯に対する支援として算出（図 5）



≪図 5 各世帯員に対する支給額(左)と「ヤドカリくん」表示結果(右)≫

支給、貸付額が一意に定まらないと算出できない

- 制度によっては最大額のみや幅のある金額で支給額を定義している
- 例:生活福祉資金貸付制度 福祉費(厚生労働省 HP)
    - 貸付上限額目安しか定められていない
- 解決策:取りうる最小額、最大額を算出
    - 見積もり結果の画面では「～最大額」や「最小額～最大額」の形式で表示(図 6)



≪図 6「ヤドカリくん」見積もり結果で幅を持たせた金額を表示した例≫

制度の命名を階層構造で管理できない

- OpenFisca のソースコード上で制度を記述する際、制度名をプログラム全体で一意にする必要がある
    - 制度名を、管理する自治体や制度の種類ごとに階層化することはできない
        - 拡張パッケージによって OpenFisca のソースコードを階層化することは可能だが、名称は全パッケージで共有される
    - 名称は同じだが計算式が異なる概念を記述しようとした際に名前が重複してしまう
        - 例:特別障害者手当の所得制限における所得控除は所得税の所得控除額と異なる(福祉局 HP)
        - OpenFisca 上で、両者を以下のような階層構造で区別することができない

《制度の階層構造》

```
所得税
|- 所得控除
特別障害者手当
|- 所得控除
```

- 解決策：開発者で命名規則を定め名前が重複しないようなプレフィックス（接頭辞）を付与
    - 「控除後所得」と区別するため「特別障害者手当の控除後所得」と命名

《制度の記述》

```
class 特別障害者手当の控除後所得(Variable):
    # …

class 控除後所得(Variable):
    # …
```

《計算式中での上記制度の計算結果を利用》

```
# 名称が異なるため、両者を混同することなく利用可能
対象世帯.members("特別障害者手当の控除後所得", 対象期間)
対象世帯.members("控除後所得", 対象期間)
```

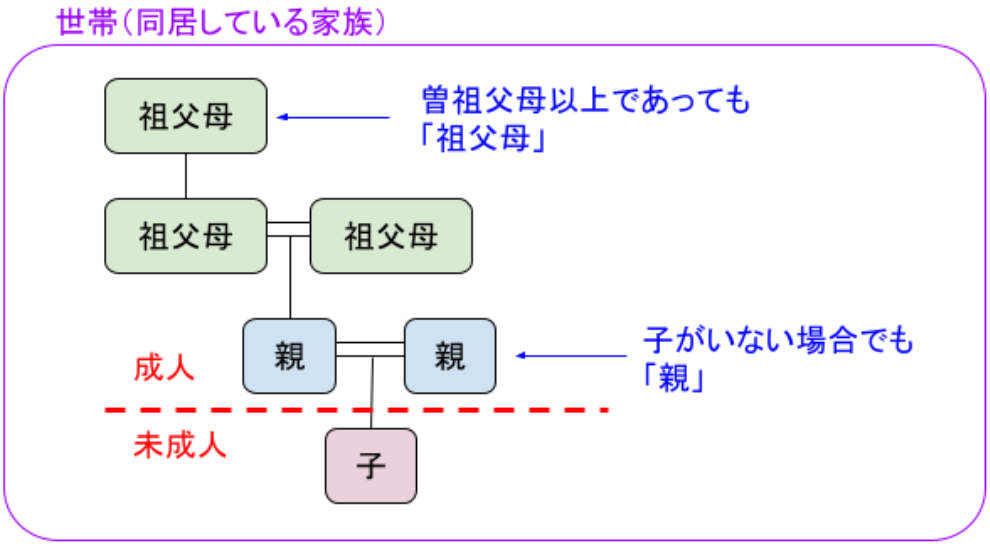
計算単位となる世帯の定義

OpenFisca のコアパッケージには、計算対象となる単位（エンティティ）は「人」とその集合である「グループ」の 2 種類が存在します。（参考：OpenFisca 公式ドキュメントの「Person, entities, role」）
各カントリーパッケージにおいて、「人」エンティティは 1 種類しか定義できないが、「グループ」エンティティは複数定義することができます。
例えば、OpenFisca-France では「グループ」エンティティとして「家族」「世帯」など複数定義されており、その構成員が重複する場合であっても制度ごとに計算単位を使い分けることが可能であります。OpenFisca-Japan では「グループ」エンティティは同居している家族として「世帯」のみ定義しています。

各「グループ」エンティティにおいて、構成員は特有の役割（ロール）を持ちます。例えば OpenFisca-Japan の「世帯」の構成員は「親」「子」「祖父母」の 3 種類のロールのいずれか 1 つを持ちます。実際の制度に適用するため、各ロールは以下のように当てはめることができます（図 7）。

- 親：成人世代のうち最も下の世代。子がいない場合でもこのロールとなる

    - （例）50 歳の A さんとその親の 80 歳の B さんの 2 人世帯の場合、A さんは「親」、B さんは「祖父母」となる
- 子：未成年。成人であっても未成年の兄弟姉妹がいる場合はこのロールとなる
- 祖父母：成人世代のうち最も下でない世代。即ち曽祖父母もこのロールとなる



《図 7 OpenFisca-Japan における世帯構成員の役割例》

以上のようにロールを定義している理由は以下の通りです。

- 子どもに対する支援制度は通常未成年の子どもを対象としており、上記の例に示した A さんのような場合は基本的に対象にならない
  （子どもに対する制度の実際の実装では、念の為「子」のロールと年齢を参照している）
- 「親」と「祖父母」を「大人」としてまとめてしまうと、一人親の判別が困難になる。上記のロール定義であれば、「親」の人数によって一人親の判別が可能
- 親子関係としての「祖父母」が制度で指定されていることは基本的になく、通常は年齢によって高齢者として扱われる
- ただし、一人親の判別等の際に「子」に対する直接の「親」でないことを明示的に判別できるよう、「祖父母」のロールを定義している

また、これらのロールは「世帯」（同居している家族）内で定義されます。そのため「親」の属する「世帯」において、血縁関係があっても同居していない祖父母や子はそれぞれ「祖父母」「子」のロールとなりません。

実際の世帯においては、事実婚や血縁関係のない養育者などが存在する場合があるが、それらを細かくロールとして定義し、漏れ・ダブりなく扱うことは難しいです。
そのため、OpenFisca-Japan に情報入力する前のインターフェースにおいて、人を各ロールに割り当てることが望ましいです。事実婚か否かなどの情報は、ロールではなく世帯の属性として定義することも可能です。例えば、災害弔慰金制度は災害で亡くなった世帯員の情報を扱うが、

OpenFisca-Japan の「世帯員」は存命の世帯員のみを指すため、世帯の属性「災害で死亡した世帯員の人数」で表現し扱っています。

《「災害で死亡した世帯員の人数」を世帯の属性に追加した例》

```
# 死亡した人は OpenFisca-Japan の運用上「世帯員」には追加できない
世帯員:
  自分:
    所得: 0
  配偶者:
    所得: 0
# 代わりに、世帯に「災害で死亡した世帯員の人数」という属性を追加し表現
世帯:
  親一覧:
    - 自分
    - 配偶者
  災害で死亡した世帯員の人数: 1
```

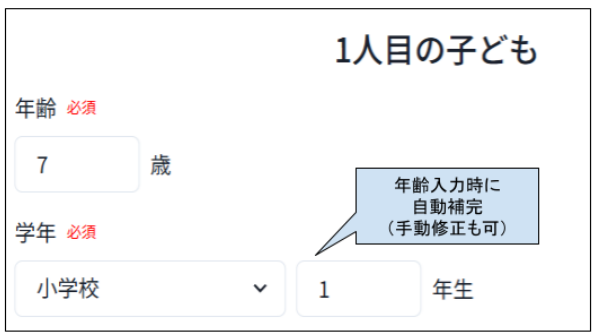
3.1.2 「ヤドカリくん」のユーザー体験による制約

継続して支給される制度において、支給期間が異なる（年額、月額、日額）

- 解決策：継続されるものは月 or 日単位に揃えて計算
- 額は機械的に按分（例：月額として、年額の 1/12 を使用）

学年を参照する制度と年齢を参照する制度が混在

- 例：児童手当の条件には、学年と年齢が両方登場（「3 歳以上高校生年代まで」）
    - こども家庭庁 HP
    - 厳密な算出には生年月日が必要だが、入力項目が多く利用者にとって煩雑だった
- 解決策：フォームには原則年齢のみ入力、子どもの情報のみ学年の入力を求める
    - 学年の入力は年齢から推測し初期値を自動補完（例：7 歳の場合「小学校 1 年生」）、ずれがある場合のみ利用者に修正してもらう（例：「小学校 2 年生」）（図 8）



≪図 8 過去版「ヤドカリくん」の生年月日入力欄（左）と現在の年齢入力欄（右）の比較≫

世帯主が特定できず、所得税が計算できない

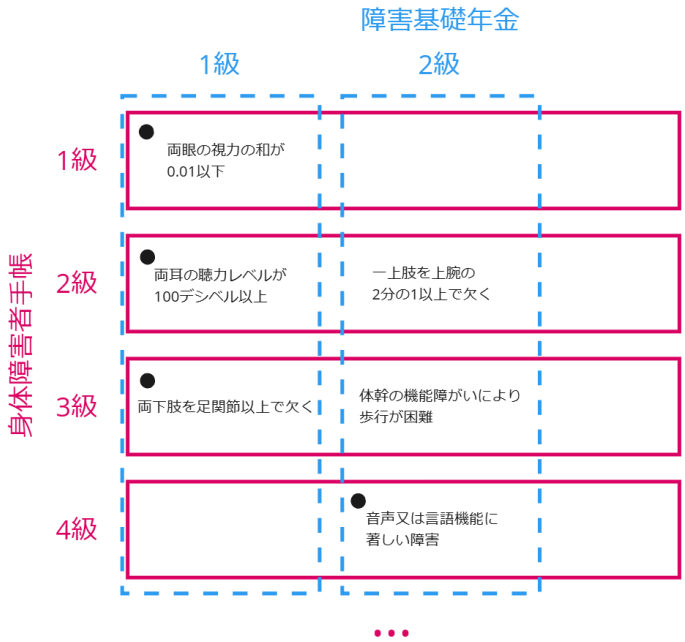
- 「世帯主が誰か」をフォームへ入力する場合、利用者に制度知識が求められる
- 解決策：もっとも収入が高い世帯員を世帯主と仮定して計算

正確な算出に制度固有の情報を必要とし、現実的な入力負荷の範囲では算出が困難

- 例：高等学校等就学支援金、単位制の場合、支給限度額算出に今年度の取得単位数が必要
    - 文部科学省 支給期間・支給限度額一覧
- 各制度に対して制度固有の情報をフォームへ追加すると、対象制度数の増加に従い必要入力項目が増えてしまう
- 解決策：厳密な支給金額の代わりに、取得可能単位の上限、下限から支給金額の上限、下限を算出
    - 高等学校等就学支援金の支給額は定額ではなく「支給限度額」として定められているため、元々幅を持たせた算出

障害関連の区分の違い

- 障害者手帳の等級、障害年金の等級、特別障害者手当の対象がすべて異なる条件である
    - 算出にはそれぞれの等級の入力、または障害の状態についての詳細な情報が必要
    - 一方、支援につながる必要のある利用者はその情報を知らない
- 解決策：「ヤドカリくん」では手帳等級にもとづき概算
    - 障害者手帳の等級を定める障害の事例から、該当しうる障害年金の等級、特別障害者手当支給金額の最大値、最小値を幅を持たせて算出（図 9）



≪図 9 身体障害者手帳等級と障害基礎年金等級それぞれに該当する障害例および
手帳等級から推定される年金等級最大値（●印）≫

3.2 運用で/目視で確認せざるを得ない注意点

OpenFisca にはテストコードを容易に作成できる機能が存在するが、テストだけでは避けられない注意点も存在します。

制度改定が起きた場合追随してソースコードの改修が必要

- テストコードを含め改定前の状態のため、未対応のままでもテストが成功してしまい更新の必要性を検知することができない
- 最新の制度に対応するためには、運用者が定期的に制度改定情報を収集する必要がある

算出した支給金額の支給期間（日額、月額、年額）の取違い防止が必要

- 「ヤドカリくん」のように web アプリケーション上で情報を使用する場合の注意点
- 得られた金額を画面に表示する際、表示欄の期間と計算結果の期間がずれていない（例：月額と年額）ことを目視で確認する必要がある

4 章 OpenFisca 導入検討に向けた提案

4.1 制度データはどうあるべきか（OpenFisca の観点から）

OpenFisca をはじめとした Rule as Code の観点から、制度データはどうあるべきか（自治体側でどのように表現・提供すると利活用しやすいか）以下に記載します。

4.1.1 具体例の提示

Rule as Code の観点のみならず一般市民が制度を理解する上でも、制度が適用されるケースの具体例を示すことは非常に重要です。
また、具体例は Rule as Code のシステムをテストする際のテストケースとなるため、その意味でもいくつか示されていることが望ましいです。特に境界となる条件（未満・以降等）においては判断が難しいため、具体例があると理解の大きな助けとなり、誤った実装を防ぐことに寄与します。

例えば、国の児童手当は公式 HP（こども家庭庁 児童手当制度のご案内）において下記のように表と図を用いて説明されています（図 10）。

1. 支給対象

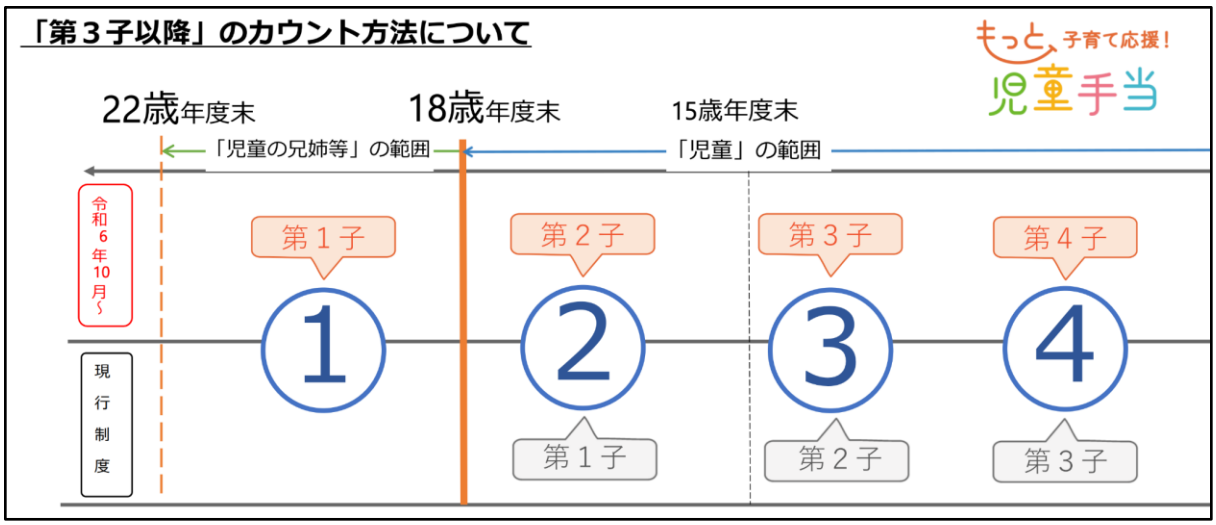
児童（0歳から18歳に達する日以後の最初の3月31日までの間にある子をいいます。以下同じ。）を養育している方

2. 支給額

児童の年齢	児童手当の額（一人あたり月額）
3歳未満	15,000円（第3子以降は30,000円）
3歳以上 高校生年代まで	10,000円（第3子以降は30,000円）

※「第3子以降」とは、児童及び児童の兄姉等のうち、年齢が上の子から数えて3人目以降の子のことをいいます。
「第3子以降」のカウント方法はこちら（PDF／322KB）をご確認ください。

《図 10-1 児童手当支給額の表（こども家庭庁 HP 抜粋）》



《図 10-2「第 3 子以降」のカウント方法についての図(こども家庭庁 PDF 抜粋)》

上記の表・図を一見しても「『第 3 子以降』のカウント方法」及び具体的な金額計算方法について正確に理解するのは難しいです。(「児童の兄妹等」と「児童」の区別が計算にどのように影響するか等。)
このように表や図を用いても、もともと複雑な制度を分かりやすく説明するのは限界があるため、下記のように場合分けして具体例をいくつか示すのが分かりやすいです。

[子が 1 人のケース]
- ケース①
  - 1 人目の子:17 歳
    - 3 歳以上 高校生年代まで→10,000 円
- ケース②
  - 1 人目の子:19 歳
    - 高校生年代越え→0 円

[子が 2 人のケース]
- ケース①
  - 1 人目の子:17 歳
    - 3 歳以上 高校生年代まで→10,000 円
  - 2 人目の子:2 歳
    - 3 歳未満→15,000 円
- ケース②
  - 1 人目の子:19 歳
    - 高校生年代越え→0 円
  - 2 人目の子:2 歳
    - 3 歳未満→15,000 円

[子どもが 3 人のケース]
- ケース①

- ○ 1 人目の子：17 歳→第 1 子
  - ■ 3 歳以上 高校生年代まで→10,000 円
- ○ 2 人目の子：15 歳→第 2 子
  - ■ 3 歳以上 高校生年代まで→10,000 円
- ○ 3 人目の子：2 歳→第 3 子
  - ■ 3 歳未満（第 3 子以降）→30,000 円

● ケース②

- ○ 1 人目の子：21 歳→第 1 子（22 歳の年度末までのため）
  - ■ 高校生年代越え→0 円
- ○ 2 人目の子：15 歳→第 2 子
  - ■ 3 歳以上 高校生年代まで→10,000 円
- ○ 3 人目の子：2 歳→第 3 子
  - ■ 3 歳未満（第 3 子以降）→30,000 円

● ケース③

- ○ 1 人目の子：23 歳→第 1 子とならない（22 歳の年度末以上のため）
  - ■ 高校生年代越え→0 円
- ○ 2 人目の子：15 歳→第 1 子
  - ■ 3 歳以上 高校生年代まで→10,000 円
- ○ 3 人目の子：2 歳→第 2 子
  - ■ 3 歳未満（第 3 子**未満**）→15,000 円

上記のように、具体例は典型的なケースや判断が難しい境界条件ケースにおいていくつか示しています。

4.1.2 支給条件や金額情報の構造化

制度の説明が文章のみだと、機械だけでなく人が解釈するのも困難です。
そのため、以下のように制度情報を構造化することが望ましいと考えています。

条件・金額の明確化

条件や金額は、下記の観点において、可能な限り曖昧さがないように分かりやすく表現されているべきだと考えております。

- ○ 誰に対しての給付・貸付か
  個人単位（例：1 人につき〇〇円）か、世帯単位（例：対象世帯員が複数いても世帯で一括して〇〇円）であるか
- ○ どの期間ごとに給付・貸付されるか
  年・四半期・月・日単位等
- ○ 金額の一意性・幅
  条件によって金額が一意に定まるか、細かい条件や明示されていない条件により幅のある金額となり得るか（貸付の場合はそもそも上限額しか指定されていないか）等

条件の区切りや関係性の明示・数式表現

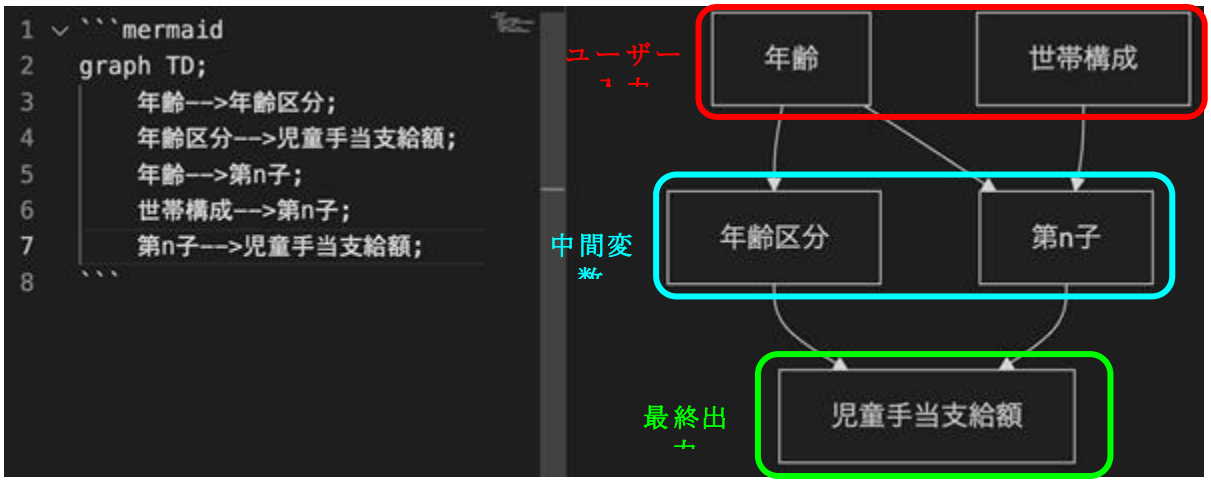
通常の制度説明文は、人が読みやすいよう基本的に一連の文章となっています。しかし、それにより、計算順序や関係性を読み解きにくい場合があります。
例えば、国税庁の扶養控除の説明 HP では、扶養親族の範囲について以下のように読みやすさを重視して記述しています。
(1)配偶者以外の親族(6 親等内の血族および 3 親等内の姻族をいいます。)または都道府県知事から養育を委託された児童(いわゆる里子)や市町村長から養護を委託された老人であること。

これを以下のように箇条書きやインデントを用いて表記すると、機械的に論理的な関係性が理解することができます。
(1)以下のいずれか （A or B or C）

- （A）配偶者以外の親族(6 親等内の血族および 3 親等内の姻族)
- （B）都道府県知事から養育を委託された児童(いわゆる里子)
- （C）市町村長から養護を委託された老人

計算の依存関係の可視化

制度の金額や適用可否の算出は、複数の要素が絡み合っているため、依存関係を可視化すると分かりやすくなります。
また、それにより
①ユーザーによる明示的な入力が必要な情報(居住地や年齢など)と、
②それらをもとにした中間的な変数(所得控除など)
を分離でき、部品化できる。それらの部品は他制度で使い回すことが可能です。
①のユーザー情報は、そのまま制度検索システムの入力フォームやマイナポータルの自己情報取得 API の入力情報となります。
フローチャートは画像を載せるだけではなく、Mermaid のようなテキストから画像生成できるツールを用いてその元テキストも公開すると、汎用的に編集することができます(図 11)。



《図 11 制度の依存関係を Mermaid により可視化した例》

条件の漏れ・ダブりの排除

数式・フローチャートによる表現により、条件の漏れ・ダブりを排除することが容易になります。（文章のみの表現では、境界条件や反復情報を正確に分かりやすく表現することが難しいです。）
条件の漏れ・ダブりがあると、プログラムにするときにあるべき状態が分からず、また不具合の要因となる可能性かもしれません。

4.1.3 参照リンクの整備

制度の説明内で参照される他制度や用語の定義について、公式情報が記載されている HP へのリンクがあることが望ましいと考えています。
同一名の制度情報であっても、複数の HP で異なった説明がされている場合があるため、どの情報が公式として参照すべきか明示されていることが重要です。
国や自治体の公式情報に抜け漏れ等がある場合は、まずそれらが整備される必要があります。

4.1.4 制度改訂情報の明示化・通知

制度が改訂された際に、改訂箇所が強調表示等で分かりやすく示されていることが望ましいと考えています。その際、制度情報が数式やフローチャートで表現されていれば、差分を明確に表現しやすいです。
また制度改訂はしばしば行われるが、多様な情報源に変更が掲載されても、それらを常時モニタリングするのは困難です。そのため、可能な限り統一されたハブのような情報源に制度改訂情報が掲載されることが望ましいと考えています。（詳細な変更情報は他ページに記載し、統一情報源にはそのリンクを掲載すれば十分です。）その情報源から OpenFisca に変更が通知されると、対応漏れを防ぐことができます。

4.1.5 自治体ごとのフォーマットの統一

制度情報の構造化や参照リンクの整備を行うことで、自治体ごとの制度情報フォーマットを統一することが可能です。それにより、自治体の制度情報整備コストの削減や機械可読性向上、自治体間の差異明確化が期待できます。
統一フォーマットは、各自治体の HP に掲載しやすいように、PDF や Word のように処理が加えられたデータではなく、Markdown や Mermaid、HTML といった柔軟に編集・表示しやすいテキストベースのフォーマットで作成されることが望ましいと考えています。

4.2 OpenFisca 導入の利点

以下の観点から、日本で OpenFisca を始めとした Rule as Code のシステムを導入することは必要だと考えます。

4.2.1 市民の制度へのアクセス性向上

現状では、複雑な制度を一般市民が理解することは難しく、使える制度があるのに使えていないケースが多々あると考えられます。それには制度検索システムが有効だが、そのためには構造化された制度データ及び OpenFisca を始めとする Rule as Code のシステムが必要となります。

4.2.2 制度の保守運用・追加の容易化

制度を構造化データとしOpenFiscaを導入することで、行政関係者が制度を参照し市民への給付可否・金額の検討を行うことが容易になります。
また、制度変更や追加の対応も容易となります。

4.2.3 政策の透明性・検証可能性・市民参加

制度データを解釈しやすい形で公開し、OSSであるOpenFiscaを導入することで、政策の透明性や検証可能性を担保することができます。また、政策への市民理解増進や政策決定への市民参加にもつながることが期待されます。

4.2.4 民間団体・営利組織のGovTech参入推進

制度データをプラットフォームとして整備・公開することで、民間団体や営利組織によるGovTech参入を推進することができます。
それにより、市民の利便性向上や行政コストの削減が期待できます。
。

4.3 OpenFisca導入の留意点

上記の通りOpenFiscaを導入する利点は多々あるが、留意点も存在します。
制度にはOpenFiscaで扱いやすいものとそうでないものがあるため、OpenFiscaはまず扱いやすいものへ適用することに集中し、そうでないものに対しては代替案も含めて幅広い視野で検討を行うことが重要です。

4.3.1 OpenFiscaで扱いやすい制度と扱いにくい制度

OpenFiscaはその仕組み上、対象制度を拡大する際に下記のようなハードルがあります。（詳しくは「「3.1.1 OpenFiscaの制約」を参照。）

- 数値計算への特化
  数値のみ算出するため、現金以外のサービス給付制度や具体的な金額が定まりにくい制度（医療費助成など）を扱いにくいケースもあります。

- 制度の分類・階層化がしにくい
  OpenFiscaは国や自治体ごとにパッケージを分けることが可能ではあるが、その中でさらに制度と変数を分類・階層化させる仕組みはありません。そのため、制度や中間変数は名前のみで区別する必要があり、対象制度の数・種類を広範に拡大させることは困難です。

- 特殊なプログラミング構文
  OpenFiscaは大規模高速演算に適したフレームワークを用いています。そのフレームワークはif分岐やforループといった条件分岐を直接的に表現しないため、OpenFiscaでは通常のプログラミングでは用いない特殊な構文を使用します。
  数値計算フレームワークの使用経験があるプログラマーでない場合、それらの構文の学習

コストがかかる可能性があります。
また、特殊な構文ゆえに、通常のプログラミングと比べて、GUI ツールや大規模言語モデル等を活用したコード自動生成を適用されにくい傾向があります。

《通常のプログラムの例》

```
…
if（学年 <= 高校生学年.三年生）and（年齢 <= 18）:
    高校生以下である ＝ True

    if 二十二歳以下の出生順 >= 2:
        高校生以下かつ第三子以降である ＝ True
…
```

《OpenFisca の例》

```
…
高校生以下である ＝（学年 <= 高校生学年.三年生）＊（年齢 <= 18）
高校生以下かつ第三子以降である ＝（二十二歳以下の出生順 >= 2）＊ 高校生以下である
…
```

一方、国や自治体の制度は種類・数が多く、現金以外のサービス支援制度も数多く存在します。また、非プログラマーが OpenFisca を使って制度を追加・修正するのは難しいため、OpenFisca を書くことができるプログラマーの確保が必要となります。

そのため OpenFisca は下記の条件を満たす制度に集中的に適用することが望ましいです。

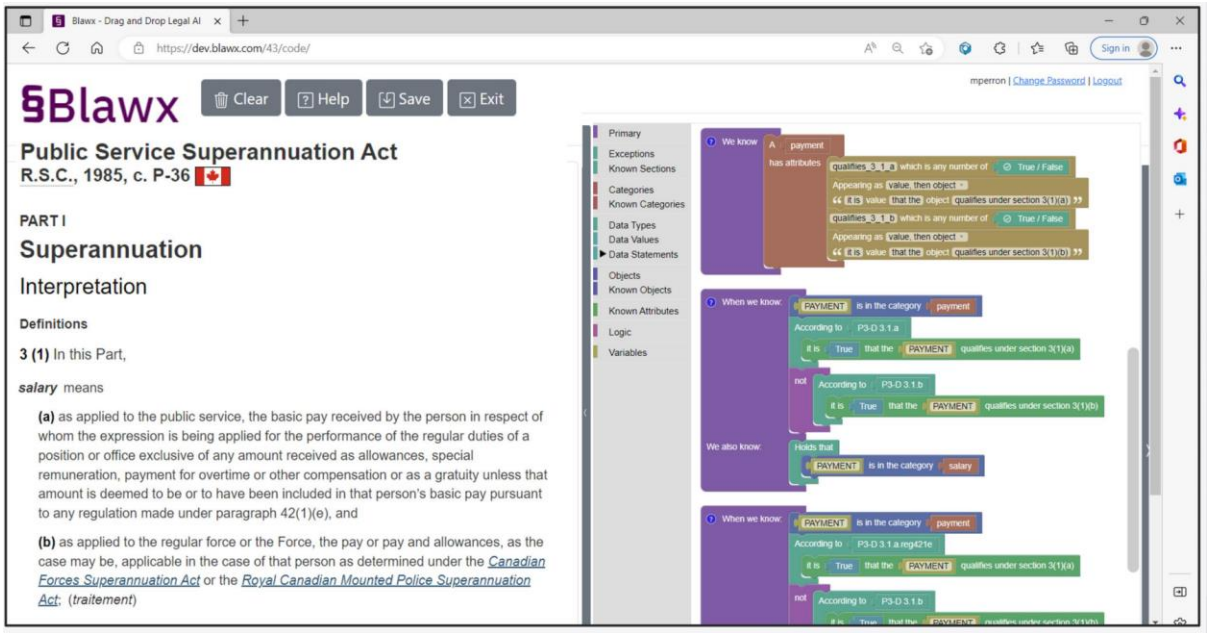
- 現金給付・貸付を主とする制度
- 対象者が多い制度
- 公的重要性が高い基本的な税制・社会保障制度

4.3.2 OpenFisca 以外の選択肢

逆に上記の条件を満たさない制度に対しては、OpenFisca の適用が難しいあるいは遅れることが考えられます。しかし、特に制度検索システムの用途においては、対象者が少なくてもその対象者にとってはニーズが高い場合があります。（例：血液製剤に混入した HIV により健康被害を受けた方の救済制度）
そのため、以下のような OpenFisca 以外の選択肢も検討することが望ましいと考えています。

- 表形式などの他のフォーマットでの情報提供
  ユニバーサルメニュー等の統一されたフォーマットで制度情報を網羅的に提供することで、OpenFisca 対象外の制度についても一定の情報提供が可能となります。

- 他の Rule as Code システムの採用検討
  OpenFisca 以外にも Rule as Code のシステムは存在します。その中でも Blawx（GitHub - Blawx）という OSS のシステムは、コーディングせずに GUI の操作で制度追加ができ、数値計算以外のロジックも扱えるため、検討材料になります。また、Blawx はカナダ政府が運用を試行中です。（参照：Rule as Code in Canada）



《図 12 Blawx の画面例》

4.4 日本の OpenFisca 導入提言

4.4.1 行政機関、GovTech 東京の採るべきアプローチ

プラットフォームの整備

行政機関、GovTech 東京としては、制度データ及び Rule as Code システムの Web API をプラットフォームとして整備し、民間団体や営利組織が利用しやすいよう整備することが重要ではないかと考えます。
その際、単に公開するだけでなく、機械可読なデータフォーマットとすること、ドキュメントやデータ・API 利用ガイドも用意することが望ましいと考えております。
また、制度の種類・量は膨大なため、公的な観点や 4.3 で述べた OpenFisca の留意点を鑑み、対象とする制度の優先順位を決めることが重要です。
同様に、OpenFisca 以外のツール（表形式や他の Rule as Code システム）を使用した制度データの情報提供も検討・推進することが重要ではないでしょうか。

他自治体への展開支援

制度は自治体によって異なるものが多いため、それぞれの自治体が自らの手で構造化データと Rule as Code システムの拡張・保守運用できることが望ましいと考えております。
そのためには、リソースの限られる地方自治体職員でも簡単に拡張・保守運用できるフォーマット、およびそのガイドや啓蒙は求められます。

また、プログラマーの確保が難しい自治体もあるため、可能な限り非プログラマーが扱えるツールを用意することも重要です。（プログラムの完全自動生成は難しいため、要件定義の補助やテストの半自動生成を行うツールなどが考えられます。）

4.4.2 想定される応用分野

制度検索システムによる市民の制度アクセス向上・行政コスト削減

日本の社会保障制度は複雑なため、一般市民が理解することが難しく、結果として支援制度が使えるのに使えていない利用者が多く存在すると考えています。（生活保護を利用する資格のある人のうち現に利用している人の割合は2割程度にすぎないという報告があります。参照：日本弁護士連合会 生活保護 Q&A）
また、実際の窓口業務を行う行政担当者にとっても、制度検索システムを用いることで、市民に対する制度の適用可否・給付額の算出等の労力・時間を削減できると考えられます。
加えて、マイナポータルの自己情報取得 API を活用すれば、ユーザーの入力負担が少なくなりさらなる効率化・利便性向上が見込まれます。

制度シミュレーションによる社会保障制度の検証・改善

OpenFisca は高速な大規模数値演算が可能なため、人口規模でシミュレーションを行うことができます。そのため、既存・仮想的な制度に対して、その効果や予算への影響の検証が可能です。実際にフランス政府は、OpenFisca を使用した LexImpact という政策シミュレーションを行う Web アプリケーションを運用しています。
また OpenFisca は OSS のため、外部研究者や一般市民が制度について検証し、不具合を調査・改善することも可能です。それにより、効果的な制度の立案や、政策決定への市民理解・参加の増進も期待されます。

付録

A. OpenFisca における制度記述のあるべき姿

曖昧性の排除

4.1 節で示したような制度データをもとに、可能な限り曖昧性を排除して制度記述を行うことが重要です。

結果の一意性

3.1.1 節で示したように、入力された情報から結果が一意の値として幅を持たずに出力される必要があります。

整理された制度体系・モジュール性

制度全体の体系が整理され、複数制度において共通する要素を同一モジュールとして定義できると、制度全体の見通しが良くなります。
例外やロジックのステップ、依存関係を少なくすることも重要です。

制度変更の即時対応とその履歴記録

制度の変更に対して OpenFisca が可能な限り素早く対応できるよう、1.1 節で示したようなテストやパラメータの仕組みを用いることが重要です。
また、その変更履歴が後から分かるよう、Git 等のソースコード管理の仕組みや OpenFisca の対象期間設定の仕組み（参考：OpenFisca 公式ドキュメント「Periods, Instants」）を活用することも有用です。

B. Rule as Code のあるべき姿

1.1 節で示した OpenFisca の特徴に関連して、Rule as Code 全体の観点からあるべき姿を記載します。

API 化

Rule as Code システムは、様々な用途（制度検索・制度シミュレーション等）・媒体（Web アプリ、スマホアプリ等）で用いられることが考えられます。それらが容易に制度データにアクセスでき、必要な情報を取得できるよう、Rule as Code システムは Web API の機能を備えていることが望ましいと考えています。
また、それにより Rule as Code システムはプラットフォームとしての役割に重点を置くことができます。
その際、標準的な API の仕様に則ることで、API を有効活用する既存のツール（OpenAPI 等）を使用することができます。

OSS

制度データは OSS として公開されていることが望ましいと考えています。
まず、政策の透明性の観点から、誰でも検証できることが望ましいと考えています。その際、単に公開するだけでなく、データの取得・解釈がしやすいよう、ドキュメントの整備やデータ構造のガイド、機械可読なデータフォーマットで公開されることが望ましいと考えています。
また、外部検証性や市民の参加可能性の観点では、公開されているプラットフォーム上で、外部研究者や市民が制度データの課題(イシュー)を指摘できること、改善提案(プルリクエスト)を行えることが望ましいと考えています。その際、不適切な言論空間とならないよう、Code of Conduct の設定や不適切な言論のモデレーションを行うことが有効だと考えられます。(参考: GitHub リポジトリ上のモデレーション例)

日本語での表現

日本の制度は当然ながら日本語で記述されているため、Rule as Code システムも人が確認する箇所は日本語で表現されていることが以下の観点から望ましいと考えています。

- 解釈容易性
  Rule as Code システムを幅広い人々にアクセス・理解してもらいます。

- 表現の一意性
  制度名は日本語で定義されているが、他言語に訳すと適切な単語が見つからない、または表記ゆれが生じる可能性があります。
  表現の一意性は、構造化・機械可読化の観点から重要です。

(OpenFisca のベースとなっているプログラミング言語 Python では、日本語を変数名として使用することが可能です。)

一方で、国内の日本語を母語としない話者の利便性、海外からのアクセス性も担保することも重要です。これらは、Rule as Code システムを活用するインターフェース側で対応することが可能だが、そのための重要単語の多言語対応表などのガイドを提供することが望ましいと考えています。

ロジックの可視化・解釈容易性

複雑な制度は構造化(コード化)しただけでは、ロジックを解釈することが難しく、保守運用・検証が困難になります。そのため、構造化されたデータ・ロジックを分かりやすい形で可視化することが重要です。
例えば、依存関係やコーディングブロックの可視化が有効だと考えられます。

ローコード・ノーコードによる拡張性

制度の定数となるパラメータ(税率等)は、随時変更が行われます。その際、構造化データも変更しやすいことが重要です。コードに変更を加えず、定義された値をノーコードで変更できると、変更の

際に対応可能な作業者を増やすことができます。（OpenFisca の該当する仕組みは 1.1 節を参照。）

また、制度は種類・数が多く、また全国共通のもの以外に地方自治体によって異なるものをふくめると膨大な量となります。それらの制度とコンピュータサイエンスの両方に精通した人材を確保するのは困難なため、行政関係者がローコード・ノーコードで対応制度を拡張できることが重要です。その際、行政関係者が制度に精通していなくても、制度の条文から構造化データを作成できるガイド機能が Rule as Code に備わっていることが理想です。
これは、人材確保以外の観点でも、Rule as Code で対応する範囲や背景情報の取捨選択の際に行政関係者の知見が求められる点、および行政関係者とプログラマーのコミュニケーションコストの削減の観点でも重要です。

C. OpenFisca 開発の流れ（ケーススタディ）

制度情報を OpenFisca プログラムで記述するまでには、プログラム実装以外の工程も存在します。本章では、一例として「支援みつもりヤドカリくん」で利用している OpenFisca プログラムの開発の流れを取り上げます。

C.1 開発体制の前提条件

ケーススタディとして取り上げる OpenFisca プログラムの開発は有志市民団体 proj-inclusive を主体とする活動で成り立っており、企業等が業務で行う開発と性質が異なる点が存在します。以下の開発体制で行った事例であることに留意が必要です。

- 有志が本業の業務外時間（休日等）の時間に活動
    - 活動量に義務はない
    - 誰でも開発に参加できる
    - 単発（1 pull request）で参加する開発者も長期間参加している開発者（コアメンバー）も存在する
- 基本的にリモートで活動、GitHub の issue、pull request、Slack 等による非同期コミュニケーション
    - コアメンバーの OpenFisca-Japan 開発経験が蓄積され、自動テスト実行、デプロイ等が整備されると非同期でも円滑に開発が進むようになった
    - 新規参加者に向けても非同期で情報共有できるよう開発ドキュメントを整備
    - ただし、開発初期や複雑な制度に取り組んだ際はオンライン、オフラインでの同期コミュニケーションが無いと進みづらい状況であった
- 制度の内容の妥当性はエンジニア同士のレビューで確認できる範囲でチェック
    - 法制度の専門家からのレビューは未実施
    - 体制や開発規模の関係上内容の正確性は保証できず、可能な範囲で品質確認を行っている（ベストエフォート）
- 主に OpenFisca 実装にかかわるコアメンバー（2 名）の技術スタック

    - Python, Numpy（OpenFisca 開発に使用）: proj-inclusive 参加前から経験あり
    - 法制度の扱い：未経験
- OpenFisca-Japan は現状「ヤドカリくん」で扱う制度のみ対応している
    - 対象が「ヤドカリくん」のユースケース、想定利用者に強く依存
        - OpenFisca-Japan は「ヤドカリくん」と合わせて開発される
    - 設計も、「ヤドカリくん」の用途に合わせ金額の正確性よりも入力変数の少なさを重視
        - 入力変数を減らすことでフォームが簡潔になり利用者の入力負荷が減るため（3 章）

C.2「支援みつもりヤドカリくん」における OpenFisca プログラム記述工程

OpenFisca-Japan は基本的に「ヤドカリくん」の Web アプリケーションのニーズに合わせて開発されました。各工程は主に一制度一人で作業を進めていくが、分量が多い場合は数人で分担する場合もあります。以下、新規で制度に対応する場合の工程を取り上げます。

問題の整理と定義

支援に繋がるまでにどのようなギャップが存在するか、「ヤドカリくん」でギャップを埋めるために何ができるかを検討しました。場合によっては他団体と連携し、利用する状況を整理しました。この工程で、対応する制度の種類（例：「子育て支援」「特定疾病に関する支援」）やペルソナ（例：「支援を受ける当事者」「当事者をサポートする団体」）を明確化しました。

関連制度の洗い出し

ペルソナに関連する制度、利用頻度の高い支援制度を洗い出しました。連携団体に制度の利用状況、利用のハードル等を尋ねる場合もありました。

関連制度の内容調査

上記で挙げた制度一覧について、支援の対象となる条件や支援の内容について調査しました。調査の際には公的機関の Web サイトの他、有志による解説記事等も参照しました。調査結果をもとに、OpenFisca での記述可否や難易度を判断し、開発の優先順位を設定しました。優先順位については、記述難易度とペルソナの制度利用頻度から総合的に判断し、難易度の判断基準の例は以下の通りです。

- 支援内容が金額であるか否か
    - 実費支給やサービス等は対応不可
- 計算式や対象条件に含まれる要素の多寡
    - 多いほど難易度が高い
- 自治体や年収区分ごとに異なるパラメータが存在するか
    - 存在する場合難易度が高い

難易度を判断するにあたって、制度自体をできるだけあいまいなところなく理解する必要があります。しかし 2 章に記載の通り、制度に精通していない場合は困難かつ時間のかかる作業となります。また、後の工程で想定していなかったケースへの対処や他制度への影響考慮等が必要になり、手戻りや再検討に時間を要することも多いです。一見して理解が簡単そうな制度でも、コーナーケースの対処などが制度データに記載されておらず、最終的に想定以上の工数がかかる場合がありました。

GitHub 上に issue を作成

調査結果をもとに、OpenFisca-Japan GitHub リポジトリ上に開発タスク用の issue を起票しました。必要に応じて対応制度に関する web サイトのリンクや、現時点で考えている設計案、懸念事項等も記載しました。issue 化を行うことで、以降の開発工程を非同期で別メンバーへ引継ぎ、全体的な作業進捗を可視化することができました。

テストケース作成

制度情報をもとにテストケースを作成しました（記述例については 1 章参照）。テストケースは以下の点を考慮して選択されます。

- 金額計算式の分岐や境界条件等に応じた代表的な入出力例を使用する
- テストケースの件数が増えすぎないよう、要素ごとのテストを重視して作成する
    - 制度の計算式では多数の要素が複雑な依存関係を持っており（4.1.2）、あらゆる分岐を網羅しようとすると要素の組み合わせパターンが膨大になるため
    - 例：児童手当では世帯構成の入力と手当額の出力ペアのテストは最低限にし、途中段階の世帯構成と年齢区分のペアのテストを多数用意する

設計・実装を無駄なく効率的に進めるため、事前に可能な限りテストケースを作成することが望ましいと考えています。特定の条件下（コーナーケース）での金額がどうなるか判断ができない場合、制度の内容調査に戻りさらに情報を集めました。

設計

制度を OpenFisca-Japan 上でどのように記述するかを設計しました。計算式中に登場する別制度がすでに記述されている場合、それを呼び出して利用するようにしました。または、同名で計算式のみ異なるものが存在する場合、区別できるように命名しました（3.1.1）。実際には、実装やテストケース作成と同時並行で進める場合も多いです。

実装（OpenFisca プログラム記述）

設計に基づき、OpenFisca プログラムを記述しました。設計やテストに抜け漏れがあれば適宜修正し、テストケースが（既存の制度のものを含め）全て成功することを確認しました。実装の際の注意点については（3.1）参照ください。

レビュー

実装に不備、改善点が無いか確認するため、作成したプログラムを別メンバーが確認（レビュー）しました。開発者は作成したプログラムを GitHub の pull request に起票し、レビュー可能な状態にしました。必要であれば変更箇所について補足説明を記載しました。

- GitHub Action による自動テストの実行
- 別メンバーの人手でのレビュー

の 2 観点で検証され、両方通過した場合マージされ既存のプログラムと統合されました（コアメンバーが軽微な修正を行った場合、自動テストのみでマージする場合もある）。マージされたプログラムは開発版の「ヤドカリくん」へ反映しました。

アプリケーション（OpenFisca プログラムを利用するアプリケーション）の実装

OpenFisca-Japan で追加した制度を「ヤドカリくん」の web アプリケーション側で利用可能にするため、アプリケーション（フロントエンド）側を改修しました。開発の流れは OpenFisca-Japan 同様、設計、実装、レビューの順に行いました。

リリース

対応予定の制度の開発が完了した段階で、作成したプログラムを本番の「ヤドカリくん」にも反映（リリース）しました。リリースによって、一般の利用者も新規追加した制度の機能を利用可能になります。

ユーザーテスト、フィードバック

当該制度に対応した「ヤドカリくん」について、想定利用者や関連団体が使用感を確認しました。内容不備や改善点、改良案等が出た場合は要件を整理し改めて開発を計画しました。

C.3 開発にかかる時間の概算

新規制度追加の各工程に費やした時間と担当者数の一例を取り上げます。なお、工数は制度の複雑さやメンバーの習熟度、コミュニケーションコスト等に依存し大きく変化することに留意が必要です。

- 対応制度: 9 制度
- 問題の整理と定義、関連制度の洗い出し: 15 時間×1 人（打ち合わせ等）
- 関連制度の内容調査、GitHub 上に issue を作成: 0.5 か月（7 時間）×2 人
- テストケース作成、設計、実装、レビュー:
    - OpenFisca-Japan: 2 か月（30 時間）×2 人
    - フロントエンド（チャットボット AI 含む）: 2.5 か月（37 時間）×2 人
- リリース: 3 時間×1 人
- ユーザーテスト: 15 時間×1 人（フィードバック対応開発除く）