

インプレスR&D [Next Publishing]



Cloud シリーズ

E-Book / Print Book

Next Publishing プレビュー



NextPublishing Sample

目次

XBRL の基礎知識	5
本章の目的	5
XBRL とは	5
XBRL の特徴	5
メリット	7
問題点	9
印刷会社の存在	11
XBRL の歴史	11
まとめ	13
 第1章 有価証券報告書とは	15
1.1 本章の目的	15
1.2 EDINET の概要	15
1.3 有価証券報告書の概要	15
1.4 項目ごとに記述される内容	16
1.6 まとめ	19
 第2章 XBRL ファイルを読み解く	21
2.1 本章の目的	21
2.2 XBRL ファイルについて	21
2.3 タクソノミの種類	22
2.4 タクソノミの更新要素	24
2.5 インスタンスの持つ付加情報	26
2.6 XML の特徴と関係性	28
2.7 まとめ	30
 第3章 XBRL で見られる XML (名前空間、XLink、XML Schema、XPointer)	32
3.1 本章の目的	32
3.2 名前空間	32
3.3 XML Schema	33
3.4 XPointer	43
3.5 まとめ	43

第4章 XMLとタクソノミ	45
4.1 本章の目的	45
4.2 タクソノミの構造	45
4.3 提出者タクソノミの作成方法	53
4.4 まとめ	54
第5章 分析環境の構築	55
5.1 本章の目的	55
5.2 EDINET APIを使えるようにする	55
5.3 プログラムを扱う準備をする	69
5.4 まとめ	70
第6章 大量の有価証券報告書を自動でダウンロードする	71
6.1 本章の目的	71
6.2 取得するコードを書く	71
6.3 ソースコード	74
6.5 まとめ	80
第7章 連結財務諸表から営業利益を自動で取得する	81
7.1 本章の目的	81
7.2 1社から取得	81
7.3 10社から取得してみる	90
7.5 まとめ	94
第8章 事業等のリスクのテキストデータを自動で取得する	95
8.1 本章の目的	95
8.2 事業等のリスクのテキストデータを自動で取得する	95
8.3 まとめ	101

第9章 提出者別タクソノミのデータを取得する～ソニーのコンテンツ価値を取得する～ (新規)	102
9.1 本章の目的	102
9.2 提出者別タクソノミの注意点	102
9.3 取得する勘定科目について	103
9.4 タクソノミを確認	103
9.5 コンテンツ資産のデータを取得	104
9.6 まとめ	108
 第10章 自前データベースを持つ(新規)	109
10.1 本章の目的	109
10.2 データベースとは	109
10.3 使用する技術	109
10.4 実際にDBを作つてみよう	110
10.5 まとめ	118
 第11章 独自のパーサーを作成する(新規)	119
11.1 本章の目的	119
11.2 Arellleと独自パーサーの違い	119
11.3 使用される技術	119
11.4 独自パーサーを作る	119
11.5 Arellleのようなライブラリーを作るには	127
11.6 まとめ	129

本章の目的

XBRLとは、財務報告をするための情報を誰もがまとめやすく、拡散しやすく、利用しやすいように国際的に標準化された電子開示に適したコンピューター言語です。XBRLはワールドスタンダードな技術であり、各国で投資家たちはXBRLを用いた意思決定を実施しております。しかし、世間にはXBRLの解説があまり存在せず、学習コストが高い傾向にあります。

そこで本章では、そもそもXBRLとは何かについて解説します。内容としては、XBRLの特徴、メリット、注意点に加え、XBRLの作られる背景の印刷会社についてやXBRLの歴史について取り扱います。

XBRLとは

XBRLはXMLというマークアップ言語をベースに作られ、財務情報の記述に特化しています。XBRLは「eXtensible Business Reporting Language」の略で、「拡張可能な事業報告言語」を意味します。

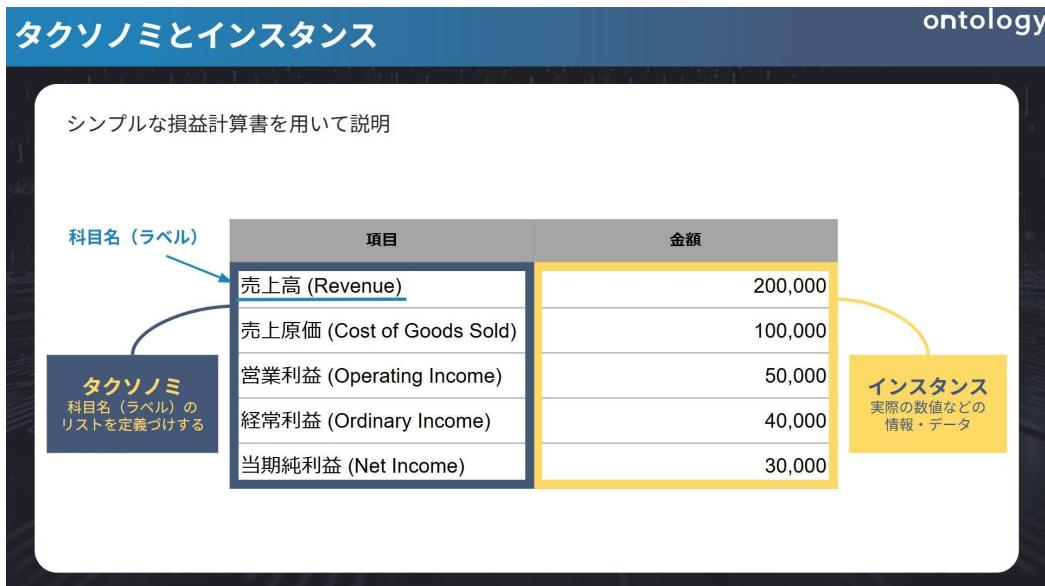
作成者側のメリットは、フォーマットに企業情報を項目として入力するだけで簡単に報告書を作成することができます。利用者側のメリットは、有価証券報告書であれば、EDINETという金融庁が運営するサービスから誰でもワンクリックでダウンロードすることができます。ダウンロードしたものは様々な形式に変換できるため、各々の用途に合わせて情報を使用できます。

XBRLの特徴

XBRLはタクソノミとインスタンスというふたつの要素によって構成されています。タクソノミとインスタンスを一言で表すとそれぞれ、タクソノミはデータを埋め込むためにひな形、インスタンスはその中身のデータです。

こちらでは、それぞれがどのようなものなのかイメージを持つ程度で大丈夫です。後に詳しくその仕組みなどについて説明します。

図1: タクソノミとインスタンス



タクソノミ

タクソノミとは、英語で「Taxonomy」、「分類」という意味を持ちます。XBRLにおけるタクソノミは、情報・データなどを階層構造で整理したものを指します。

この階層構造で整理した要素のひとつひとつに対して項目（名称）を定義し、ひも付けする役割を持つものがタクソノミです。実際に知りたい情報を取得する際には、プログラム上で情報に呼応するタクソノミを指定することで取得が可能です。また、言語が異なるなど表面上の名称などが異なっていても、本質的な意味が同じものであれば、裏で指定されているタクソノミが同じ場合があります。そういう場合には、そのタクソノミを使用することで、複数の企業からその情報を一度に引っ張ってくることも可能です。

インスタンス

インスタンスとは、英語で「Instance」、「実例」という意味を持ちます。XBRLにおけるインスタンスは、実際の情報・データそのものを指します。

たとえば、「今年度の売上高」の値段や、「事業等のリスク」のテキストなどの実データの部分を指します。ただし、このときの実データは数値やテキストなどに縛りはありません。つまり、書類に記載された情報・データがインスタンスです。

また、データがいつのものなのか、通貨単位は日本円なのか米ドルなのかなど、期間や通貨単位等はタクソノミではなく、インスタンスで設定されます。

最近ではIT技術の発展により、テキストデータが簡単に取れるということに重要な価値があると注目されています。

メリット

XBRLのメリットは、圧倒的なコスト削減とデータの柔軟性です。これはタグを指定することに由来しますが、さらに具体的に言うと、以下のようなメリットを実現させました。

- ・ひな型に項目（インスタンス）を入力するだけで書類を作成できる、作成コストの大幅カット
- ・EDINETなどで簡単に様々な形式でダウンロードができるデータ流通のよさ
- ・タグで情報を紐づけているため、ファイル形式の変換が容易で二次利用がしやすい

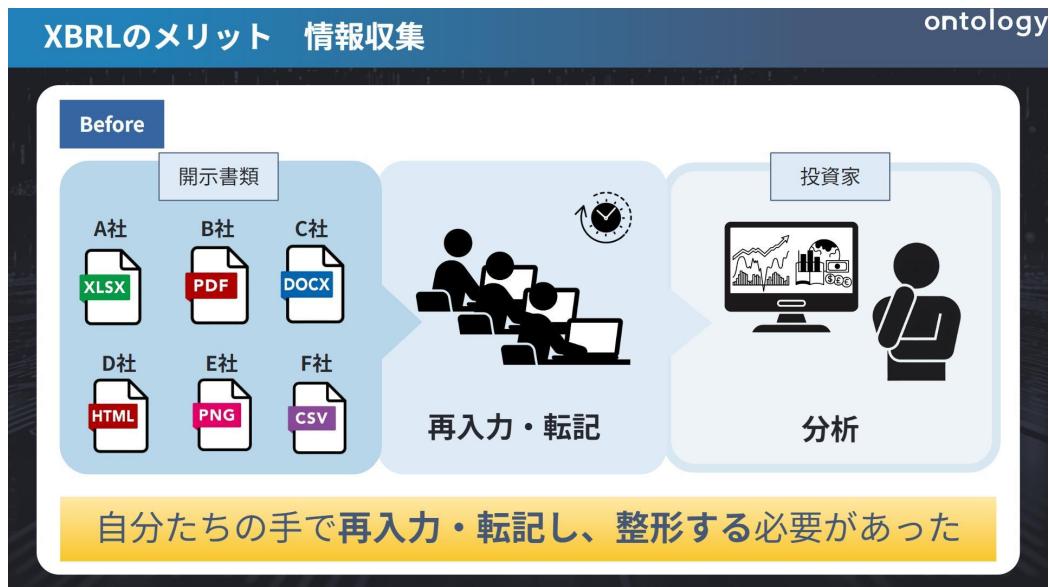
以上のメリットを利用者、作成者、提出機関の三者の立場に分けて詳しく説明します。

利用者へのメリット

利用者というのは投資家、アナリスト、情報ベンダーなどXBRLを利用し情報を取得する方を指し、本書を手に取られた多くの読者も、こうした利用者に該当すると考えられます。

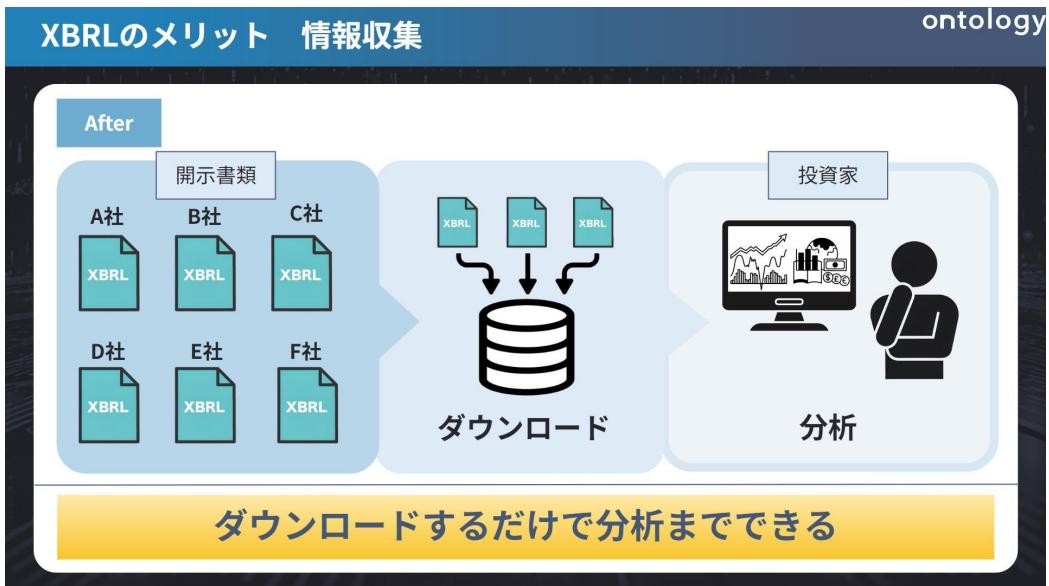
どの企業の調子がいいのか、これから好調になるのかを判断するための材料として有価証券報告書を見て、その他の多くの企業と比べることを目的とされている方たちです。

図2: XBRL 登場前の処理



利用者は開示された資料を見て、手動で再入力・転記することで情報を手元に置いていました。容易に想像がつくように、この方法では時間もかかるうえミスが生じやすく、確認作業の負担も大きくなりがちです。このような状況では、十分な分析に取り組むことは困難になります。

図3: XBRL登場後の処理



XBRLではデータをひとつの規格に統一して管理するため、自動的な整合性チェックによるミスやエラー防止対策が実装されています。これにより、情報の品質と透明性が向上しました。さらに収集したデータをブラウザ上で閲覧できるようにした結果、データの加工や分析を迅速に行えるだけでなく、情報へのアクセスも容易となりました。

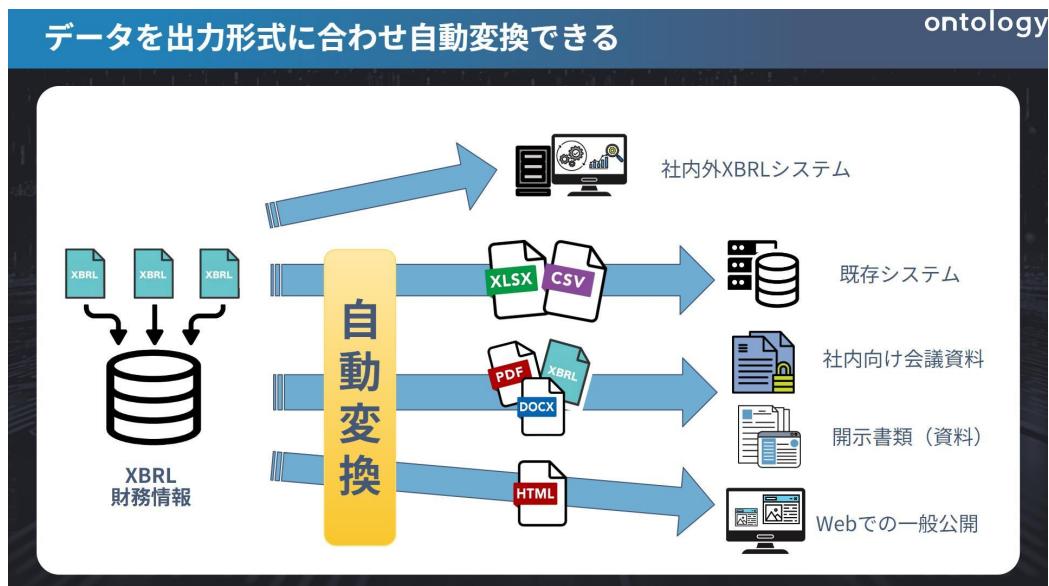
作成者へのメリット

主に上場企業など、XBRLを用いて書類を作成する人を指します。

財務情報は年度や組織、業種ごとで文書構造や項目、計算式などが大きく異なります。これらをフォーマットもない0の状態から毎回作成すると、時間もお金も莫大にかかります。

EDINETでは、ExcelやCSV、PDFなど多様な形式で情報取得することができます。これもXBRLで情報を集約しているため、ひとつひとつを個別で作成する必要なく形式を自動で変換することができます。

図4: 出力形式の自動変換



提出機関へのメリット

主に証券取引所や監督機関などを指します。作成された書類に誤った情報があつてはなりません。そのため、正しいかどうか、整合性チェックなどの財務情報の確認作業をする必要があります。

XBRLには「事前に指定された計算式に対してありえない情報が入っていないか」という整合性チェックの機能が実装されています。

これにより確認作業を自動化し、業務効率化を図ることができます。そういうた地道な細かいチェックは人間よりも機械の方が優れるため、制度・信憑性の高い財務情報の取得にも繋がります。こういった正しいデータができると、深度を増した企業分析が可能となります。

問題点

ここまでXBRLのメリットを紹介しました。ですが、デメリットも存在します。

主なデメリットは以下の3つです。

- ・タグの付け方にすべてが依存される
- ・会計基準の違いがある場合は一括取得できない
- ・タクソノミを毎年チェックしなくてはならない

タグの付け方にすべてが依存される

XBRLのタクソノミには、タグというものがあります。タグは要素を表すための<と>で囲まれたマーク付けです。このタグを使用して情報を識別し、取得に使用します。

一般的な項目に関してはEDINETがタグを設定していますが、企業独自が設定しているタグも存

在し、これに関しては共通化されていません。そのため、複数の企業の情報を取得する際には、タグによっては欲しい情報を取得することができないこともあります。要因としては、XBRLファイルの作成自体を印刷会社のシステムに依存していることが一因になります。印刷会社のシステムを利用することで、利用者はタグの付け方を意識せずにXBRLファイルの作成自体はできますが、代わりにタグが正確につけられないことがありますとえば、昨今、注目されているESGの活動についても2023年までは政府が指定しているタクソノミがなかったため、会社独自の拡張タクソノミを指定していました。その結果、当時は複数の企業が一括で取得することが困難でした。

会計基準の違いがある場合は一括取得できない

会計分野の大前提として、会計基準の違いがある場合は、一見同じ勘定科目や似たような意味を持つ項目であれど、まったくの別物です。会計基準は企業活動に関わるお金の一切を管理するルールであるため、ルールが異なるもの同士で単純な比較を行うことはできません。企業や財務諸表（連結、個別）によって会計基準が異なることがあるため、常にその企業、財務諸表が採用している会計基準を確認するべきです。

プログラム的な側面から見ると、日本会計基準と国際会計基準では、そもそもタグ自体が異なるものを使用しています。そのため、タグの付け方にすべてが依存されているXBRLでは、異なる会計基準のデータを抜き出すことが不可能です。

タクソノミを毎年チェックしなくてはならない

EDINETでは、毎年タクソノミが公表されており、変更が生じる可能性があります。そのため、利用者は毎年の公表時期にタクソノミを確認する必要があります。たとえば、ESG関連のタグのように多くの企業が開示情報として記述する内容であれば、標準タクソノミへ追加される場合もあります。しかし、将来的に開示項目が追加・削除されると、それに伴ってタクソノミも変更されるリスクがある点に留意が必要です。

コラム：会計基準や財務諸表の規模が異なるとデータの同時取得ができない理由

会計基準が異なると同じコード（単一のタクソノミだけ）では、データ取得ができません。理由は会計基準というそもそものルールが異なるためです。日本の企業が財務諸表を作成する場合、従来からの日本会計基準に則って作成される場合と各国の異なるルールを一般化しようとする国際会計基準に則って作成する場合が混在しています。

同じ財務諸表とはいえ、このように基本的に会計基準が異なると比較することはできません。取得に関しても同様に、勘定科目ひとつひとつに紐づいているタクソノミは会計基準によっても異なるため、同じタクソノミではインスタンスを取得することはできません。

同じ財務諸表とはいえ、このように基本的に会計基準が異なると比較することはできません。取得に関しても同様に、勘定科目ひとつひとつに紐づいているタクソノミは会計基準によっても異なるため、同じタクソノミではインスタンスを取得することはできません。

印刷会社の存在

有価証券報告書をはじめとする開示書類はEDINETの稼働により、全面XBRL化を果たしました。開示書類は主に上場企業が投資家たちに向けて、企業の状況を説明する書類です。しかし、すべての企業の経理担当者がXBRLを理解し、開示書類を作成できるわけではありません。それにも関わらず、日本の上場企業はXBRLを活用し、毎年開示書類を作成できています。その背景には、印刷会社の隠れた努力があります。特に、宝印刷とプロネクサスの2社による強力なサポートなくして、現在のXBRLの利用状況はありません。この2社はそれぞれ、WizLaboとPRONEXUS WORKSという開示書類の作成支援システムを開発しています。これらはXBRLの大枠となるフォーマットであり、各企業ではこのシステムを導入することで、作成者自身はXBRLを気にせずに開示書類を作成することができます。このシステムを利用することで、WordのようなUIを用いて数値や文章を入力するだけで、XBRLに対応した書類を作成できるようになっています。これらのシステムでは、XBRLの特徴でもあるタグが暗黙的に記録されるため、わざわざ作成者側がタグ付けを意識する必要もなく、自動的にタグが付与されたXBRL形式でデータがEDINETに展開されることになります。こうした統一性により、開示情報としての価値が格段に向上しています。このように、XBRLはXBRLの技術はもちろん、印刷会社の協力のもと成り立っています。日本はXBRL大国であると言われますが、印刷会社のサポートがあるおかげでXBRLの情報を蓄えられた結果と言えるでしょう。

ただし、いいか悪いか、こうした印刷会社の努力により、開示書類を作成されている方でもXBRLの存在を知らないケースも多いのもまた実情です。

XBRLの歴史

XBRL誕生についてと、日本におけるXBRL導入の流れ、XBRL現状の姿などについて説明します。

XBRLの誕生

アメリカのナイト・ベイル&グレゴリー (Night, Vale & Gregory) 事務所所属の公認会計士チャールズ・ホフマン (Charles Hoffman) 氏によってXBRLの原形は作られ、「XBRLの父」とも呼ばれます。

ホフマン氏が財務情報の電子的報告にXMLを応用することができるのではないかと考えたことで、XBRLのプロジェクトが始まりました。1998年4月にホフマン氏はXBRLのひとつ目のプロトタイプの開発に着手しました。そこから間もなく同年6月、ホフマン氏は米国公認会計士協会 (AICPA) の先端技術検討部会の部会長であったウェイン・ハーディング (Wayne Harding) 氏に相談し、財務報告にXMLを使用することの意義を知らせました。同年10月にAICPAの協力が得られると、同年の12月31日にプロトタイプを完成させ、翌年1999年1月にプロトタイプが披露されました。これによりホフマン氏とハーディング氏は、XMLが会計士にとって有用であることをAICPAに理解してもらいました。それにより、AICPAはXMLベースの財務諸表を検討するための事業計画を立案する運びとなります。ただし、そのときの事業計画のコード・ネームはXBRLではなく、XFRML (eXtensible Financial Reporting Markup Language) というものでした。

1999年6月にホフマン氏ハーディング氏に加え、公認会計士のエリック・コーベン (Eric Cohen) 氏と AICPA 情報技術ディレクター兼公認会計士のルイス・マルサーン (Louis Matherne) 氏によって XFRML の事業計画が作成されました。同年7月にはホフマン氏は、XFRML の実験的プロトタイプを AICPA に提出し、予算をつけることに成功します。同年8月には XFRML 運営委員会の発足、AICPA の他に 12 組織がメンバーとして参加することとなり、事業計画に沿って XML 財務諸表の仕様を作成を発表しました。

同年10月にニューヨークで最初の XFRML 運営委員会が開催されました。これにより、XBRL はわずか1年半という短い期間で軌道に乗ることに成功しました。

翌年、2000年4月に XFRML 運営委員会は「XBRL 運営委員会」と名前を改めました。このときより現在まで XBRL と呼ばれています。また、XBRL はその基盤が2003年までに作られ、現在も変わらず使用され続けています。

日本での XBRL

従来、財務報告書は電子的報告が一般ではなく、紙媒体で作成・提出されていました。また、入手方法についても書店等で1冊2,000円前後で購入するなどしかない状況でした。なぜなら、財務情報というのは、組織や業種だけでなく年度によっても文書構造や項目、計算式が異なるため、文字通り白紙からの作成となり、作成コストはとても高い状況であったためです。さらに、紙媒体で作成されているため、情報の二次利用も困難です。二次利用をするのに、個人的にデータをパソコンで転記しないとデータとして扱えないためです。それゆえ、作成コストだけでなく使用コストも高い状況でした。

図5: EDINET のトップページ画面

The screenshot shows the EDINET homepage. At the top, there is a header with the EDINET logo and a search bar. Below the header, there are several menu items: 'トップページ', '書類詳細検索', '書類全文検索', '公告一覧', 'EDINETクラウド登録', and 'コードリスト' with a 'ログイン' button. On the left side, there is a sidebar with various links and sections, including 'お知らせ一覧', 'よくある質問', '規制取引法に基づく有価証券報告書等に関する電子化システム', '利⽤取扱', '操作ガイド', 'システムメンテナンス情報', and '金融商品取引法等に基づき有価証券報告書等の提出期間の変動が規定されている会社一覧'. The main content area has sections for '書類検索結果' (with a search form for '提出者/発行者/ファンド/証券コード' and a checkbox for 'ファンドを含む'), '書類種別' (checkboxes for '有価証券報告書 / 半期報告書 / 四半期報告書', '大量保有報告書', '臨時報告書', and 'その他の書類種別' with a note about '各訂正報告書を含みます。'), '提出期間' (dropdown menu set to '過去1年'), and '重要なお知らせ' (with links to '施行開示書類(参照方式)の参照書類について' (with a note '必ずご確認ください。'), '金融商品取引法等の一部を改正する法律(令和5年法律第79号)における経過措置によ...→続きを読む', '取次済の発行登録書及びその添付書類を監査される方へ' (with a note '必ずご確認ください。'), and '年次年始のお知らせについて').

そのような事態を開いたのは、XBRL と EDINET のふたつです。EDINET とは金融庁が提供する「金融商品取引法に基づく有価証券報告書等の開示書類に関する電子開示システム」を指し、Electric Discloser of Investors' NETwork の略称です。EDINET が導入されたことにより、従来、

紙媒体で提出されていた開示書類について、提出から閲覧まで手続きなどを全て電子化できるようになりました。つまり、今まで上場企業は直接財務局へ赴き、提出していた書類をインターネット上で提出できるようになりました。また、閲覧者はそれまでコストをかけて有価証券報告書を購入することで手にしていた情報を無料で閲覧できるようになりました。提出者の負担の軽減だけでなく、投資家等の企業情報へのアクセスの公平・迅速化が実現しました。財務省としても書類の受理業務を簡略化するなどメリットはありました。しかし、EDINETの普及だけではデータの再利用や多言語対応が難しいなどの点から、EDINETの機能を充実させるために2005年にXBRL化に向けた動きが表明されました。

2008年に実際にXBRLが導入されたときには、有価証券報告書、四半期報告書等の開示書類の財務諸表部分のみに限定したXBRL形式の義務化でした。続く2013年9月には、さらなる利便性の向上のため「次世代EDINET」を稼働し、検索機能等の強化や有価証券報告書、四半期報告書等の開示書類全体のXBRL形式化、XBRLの対象となる開示書類の拡大が行われることとなりました。それに伴い、XBRLファイルの形式としてiXBRL（オンラインXBRL）形式を導入しました。iXBRLが普及し、現在では作成・提出から閲覧までインターネット上で可能となりました。

iXBRLとは

現在使用されているiXBRL（オンラインXBRL）とは、単一の文書において、人間と機械の両方が判読可能な構造を持つデータの提供を可能にするオープンスタンダードです。iXBRLの特徴としては、ウェブページを表示するために使用されているHTMLを取り入れることで人が判読可能であり、XBRLのタグ情報も組み込むことでコンピューターにも判読可能な形式となっています。

提出者は印刷会社のシステムを用いて、「.htm」形式（iXBRLファイル）で開示書類を作成し、EDINETに提出します。提出された書類はEDINET側で自動的にXBRLインスタンスファイルという「.xbrl」形式のファイルを生成しています。そのため、EDINETより有価証券報告書等をダウンロードすると、「ダウンロードデータ」と「提出データ」と別れている階層が存在します。提出データのフォルダーには「.xbrl」ファイルは存在しないため、これからデータを利用する際には.xbrlファイルがあるダウンロードデータを参照する必要があります。

まとめ

本章では、XBRLについて、その特徴、メリット、注意点に、XBRLの作られ方や歴史を交えて解説しました。

XBRL（eXtensible Business Reporting Language）は、財務報告をするための情報を誰もがまとめやすく、拡張しやすく、利用しやすいように国際的に標準化された電子開示に適したコンピューター言語です。XML技術を基盤として、タクソノミとインスタンスで構成され、ひな型（タクソノミ）にデータ（インスタンス）を埋め込むことで書類作成や分析が容易となります。

この仕組みにより、投資家やアナリストは手動で転記する手間を省き、多数の企業情報を一括比較でき、整合性チェックで品質と信頼性が向上するなど、多くのメリットがあります。一方、異なる会計基準や企業独自タグの扱い、タクソノミの年次更新など、いくつかに留意することでより快

適に活用することができます。

また、XBRLの歴史やXBRLを作成する背景にある印刷会社の存在などにも触れました。これで、XBRL自体がどのようなものなのか大まかには理解できたかと思います。

次章では、データの取得元である有価証券報告書について解説します。

第1章 有価証券報告書とは

1.1 本章の目的

本書では、有価証券報告書のXBRLファイルを使用し分析を行います。なぜなら有価証券報告書は開示書類の中で最もデータ量が豊富であるためです。分析を行うには、有価証券報告書自体の理解も必要となります。XBRLの詳しい説明の前に、本章では分析対象である有価証券報告書の基礎知識を固めます。

1.2 EDINETの概要

有価証券報告書について学ぶためには、まず自分で書類を閲覧できなければなりません。

書類を閲覧するにはEDINETを使用します。EDINETとは金融庁が提供する「金融商品取引法に基づく有価証券開示書類に関する電子開示システム」です。EDINETでは、すべてのデータがXBRLデータをもとに提供されており、誰もが無料で企業の開示情報をwebで閲覧することができます。

サイトページを開き、任意で条件を絞り検索をかけることによって、自分の閲覧したい企業データを入手できます。

▼EDINET トップページ

<https://disclosure2.edinet-fsa.go.jp>

EDINETは金融庁が提供している一次データが入手可能であり、その信頼性から投資家や研究者、企業関係者にとって重要な情報源です。

1.3 有価証券報告書の概要

有価証券報告書とは、事業年度ごとに企業自ら財務情報や経営状況を外部に開示するための書類です。提出義務のある企業は金融商品取引法で定められており、基本的には上場企業が当たります。提出時期は、主に事業年度が切り替わってから前年度分の総まとめとして提出されます。

このような企業による私的情報開示をディスクロージャーと呼び、企業と我々の情報量の差を小さくする目的があります。ディスクロージャーは有価証券報告書以外には決算短信などがあり、提出時期や頻度、記述する項目などに違いがあります。

また、有価証券報告書は、記述される内容の大枠は内閣府令によって定められています。一般的な有価証券報告書に適用されている内閣府令は「企業内容等の開示に関する内閣府令 第三号様式」です。

1.4 項目ごとに記述される内容

企業情報の大項目は以下のように区分されています。また、有価証券報告書ではこれらの項目は隅付き括弧（〔 〕）で囲われることが多いです。

表1.1: 企業の大項目

大項目	内容
【企業の概要】	主要な経営指標の推移や従業員数、事業内容等の基本情報
【事業の情報】	経営方針や事業に対する課題、サステナビリティに対する取り組み等
【設備の情報】	建物や機械など設備への投資金額等
【提出会社の状況】	株式の総数や役員の状況等、主に株式とコーポレートガバナンスに関連した情報
【経理の状況】	財務諸表や注記によって勘定科目ごとの金額
【提出会社の株式事務の概要】	株式に関する事務手続きや権利関係についての情報
【提出会社の参考情報】	企業が当期の間に提出した書類や親会社等の参考情報

知りたい情報ごとに、見るべき項目をトヨタ自動車株式会社を例に提示します。

1.4.1 企業の全容を把握したい

図1.1: トヨタ自動車の「企業概要」

E02144 : トヨタ自動車株式会社 （法人番号） 1180301018771 S100TR7I : 有価証券報告書 - 第120期 (2023/04/01 - 2024/03/31)

過去5年分の主要な財務データや指標が記載

項目	国際財務報告基準				
	第118期	第117期	第118期	第119期	第120期
決算期	2020年3月	2021年3月	2022年3月	2023年3月	2024年3月
営業収益 (百円)	29,868,547	27,214,594	31,379,507	27,154,238	45,095,825
従業員数 (人)	2,322,254	2,320,532	2,688,733	2,685,005	2,685,005
平均年齢 (歳)	40.9	40.9	41.1	41.1	41.1
平均年収 (百円)	2,088,140	2,245,281	2,850,110	2,451,318	4,944,898

従業員数、平均年齢、平均年収などが記載

企業の全容について知りたい場合は、【企業の概要】にて確認できます。

企業の概要は、会社の基本的な情報がまとめられた項目です。ここでは、会社の経営状況や歴史、事業内容、グループ会社の関係、従業員の状況などが詳しく記載されています。これらの項目から情報を整理することで、企業の概要を把握します。

1.4.2 事業について詳しく把握したい

図 1.2: トヨタ自動車の「事業状況」

E02144 : トヨタ自動車株式会社 (法人番号) 1180301018771 S100TR7I : 有価証券報告書 - 第120期 (2023/04/01 - 2024/03/31)

どのような事業を行っているのか文章で記述される

事業で起こりうるリスクが記述される

経営者の分析による現在の成績・状態になった理由が記述される

企業の事業について知りたい場合は、【企業の概要】や【事業の状況】で確認できます。

企業の概要は企業がどのような事業を行っているのかがまとめられた項目であり、事業の状況は企業の事業に対する考え方やリスク・取り組みについてまとめられた項目です。企業の事業内容やそれに関わるリスクについて知ることができます。これらの項目から、多面的に事業を把握します。

1.4.3 株式や経営体制を把握したい

図1.3: トヨタ自動車の「提出会社の状況」

E02144 : トヨタ自動車株式会社 (法人番号) 1180301018771 S100TR7I : 有価証券報告書 - 第120期 (2023/04/01 - 2024/03/31)

提出本文書 | 監査報告書 | 代替書面・添付文書

提出本文書 | 検索

第4 【提出会社の状況】

1 【株式等の状況】

(1) 【株式の総数等】

発行可能な株式の総数やすべて発行がされている株式の総数が記載

種類	発行可能株式総数(株)
普通株式	50,000,000,000
計	50,000,000,000

(2) 【新株予約権等の状況】

自社の株を多く保有している他企業が記載

種類	事業年度末現在 発行枚(株)	提出日現在 発行枚(株)	上場会員登録番号 取扱い証券名	内容
普通株式	18,314,887,460	15,784,887,460	東京、名古屋、ニューヨーク、ロンドン各証券取引所(東京はブライム市場、名古屋はブルーミア市場)	単元株式数 100株 (注) 1
計	18,314,887,460	15,784,887,460	—	—

(注) 1 発行済株式は、すべて抹消権を有する株式です。
2 発行済株式は、2024年5月9日に自己株式を消却したことにより520,000株減少しています。

(2) 【新株予約権等の状況】

① 【ストックオプション制度の内容】

該当事項はありません。

② 【ライツプランの内容】

該当事項はありません。

③ 【その他の新株予約権等の状況】

該当事項はありません。

(3) 【発行価額修正条項付新株予約権付認定券の行使状況】

該当事項はありません。

(4) 【発行済株式総数、資本金等の推移】

役員の役職、名前、略歴等が記載

年月日	発行済株式 総数(株) (千株)	発行済株式 総数(株) (千株)	資本金増減額 (百万円)	資本金残高 (百万円)	資本準備金 増減額 (百万円)	資本準備金 残高 (百万円)

株式や経営体制について知りたい場合は、【提出会社の状況】で確認できます。

提出会社の状況は主に株式とコーポレートガバナンスに関連した情報がまとめられた項目です。発行株式数や株式所有者、企業の役員などが記載されています。これらの項目から株式や経営体制を把握します。

1.4.4 財務情報から企業を把握したい

図 1.4: トヨタ自動車の「経理の状況」

E02144 : トヨタ自動車株式会社 (法人番号) 1180301018771 S100TR7I : 有価証券報告書 - 第120期 (2023/04/01 - 2024/03/31)

提出本文書 監査報告書 代替画面・添付文書

提出本文書 検索

監査報告書

代替画面・添付文書

【連結財務諸表等】
(1) 【連結財務諸表】
① 【連結損益計算書】

子会社も含めた財務データが記載

括弧外の項目は
4 配当政策
4-1 ポリシー・ガバナンスの
実現
(1) コーポレートガバナンス
① 署名
② 役員の状況
③ 監査の状況
④ 治理の相談窓
⑤ 株式の保有状況
第5章 経理の状況
1 連結財務諸表等
(1) 連結財務諸表
① 連結財務状況
② 連結損益計算書及び直
結式損益計算書
③ 連結会計計算書
④ 連結貸借対照表
⑤ 連結キャッシュフロー計
算
2 連結財務諸表注記
(2) その他
② 連結財務諸表
① 連結財務状況
② 連結損益計算書
③ 連結会計計算書
④ 連結貸借対照表
⑤ 連結キャッシュフロー計
算
3 連結財務諸表注記
(2) その他
③ その他

子会社を含めない企業個別での財務データが記載

セグメント情報などのより詳細なデータが記載

子会社を含めない企業個別での財務データが記載

資産	注記	前連結会計年度 (2023年3月31日)	当連結会計年度 (2024年3月31日)
流動資産			
現金及び現金同等物	6	7,618,988	9,412,080
預金	7	3,688,130	3,789,428
販売債権及びその他の債権	8	8,279,806	11,087,283
金額請求による債権	9	1,715,876	4,702,188
その他の金融資産	10	4,256,614	4,805,388
棚卸資産			
未収法人所得税			
その他の金融資産			
流動資産合計		888,885	1,031,698
		26,459,781	34,714,279
非流動資産			
持分法で会計処理されている投資	11	5,227,945	5,710,106
金融事業による債権	8	18,439,045	20,687,030
その他の金融資産	9	10,558,431	11,980,559
有形固定資産			
土地	12	1,426,370	1,441,811
建物	12	5,484,811	5,884,749
機械装置	12	14,786,819	16,469,032
販賣用具及び器具	12	6,774,427	7,528,311
運送設備	12	845,886	1,040,188
小計		23,839,038	32,358,692
		△18,705,119	△19,101,305
無形資産	12	12,832,974	14,287,798
有形固定資産合計			
使用権資産	13	491,388	532,635
無形資産	14	1,785,152	1,995,328
繰延税金資産	15	887,427	502,230
その他の非流動資産			
非流動資産合計		886,887	1,014,058
		47,943,398	55,400,017
資産合計		74,838,180	80,114,286

(単位: 百万円)

財務情報について知りたい場合は、【経理の状況】で確認できます。

経理の状況には、企業の財務会計に関する詳しい情報が記載されています。企業の資産や負債、利益の推移や資金の流れなどを、財務諸表を通じて確認できます。また、セグメント情報のような詳細な内訳は注記などに記載されています。これら項目の数値から、企業の経営状況や財務の健全性を把握します。

1.5 コラム：有価証券報告書自体の勉強法

有価証券報告書の解析を行うには、それぞれの項目にどのようなことが記載されているのかを理解しておくことが重要です。

有価証券報告書に対する理解を深める手段として、実際に1社分の報告書を通して理解いたします。その際には目次を注意深く確認し、各項目に何が記載されているのかを把握しながら読み進めると、理解が深まります。

今後欲しい情報が出てきたときに「その情報は有価証券報告書のこの項目に書かれている」と見当をつけることができます。

また、財務データに関しては以下の3つのポイントを踏まえ、通読してください。

- ・貸借対照表・損益計算書・キャッシュフロー計算書に何が書いてあるか理解する
- ・注記に何が書いてあるのか理解する
- ・セグメントと収益認識とは何なのか理解する

これらを意識し通読することで有価証券報告書への理解がより深まり、スムーズに解析を行うことに繋がります。

1.6 まとめ

本章では、分析対象である有価証券報告書について解説しました。書類の内容を理解すればする

ほどXBRLを用いた分析は行いやすくなります。これらの内容は会計分野の知識がどうしても必要になるため、本章がその入り口への第一歩になれば幸いです。

次の章ではXBRLファイルを読み解くためにタクソノミとインスタンス、そしてそれらを構成するXMLの特徴について解説を行います。

第2章 XBRLファイルを読み解く

2.1 本章の目的

本章の目的は、XBRLファイルを読み解くことです。特にXBRLファイルの構造、タクソノミとインスタンスの細かい種類や定義している内容と、XBRLの元となったXMLの特徴を解説します。

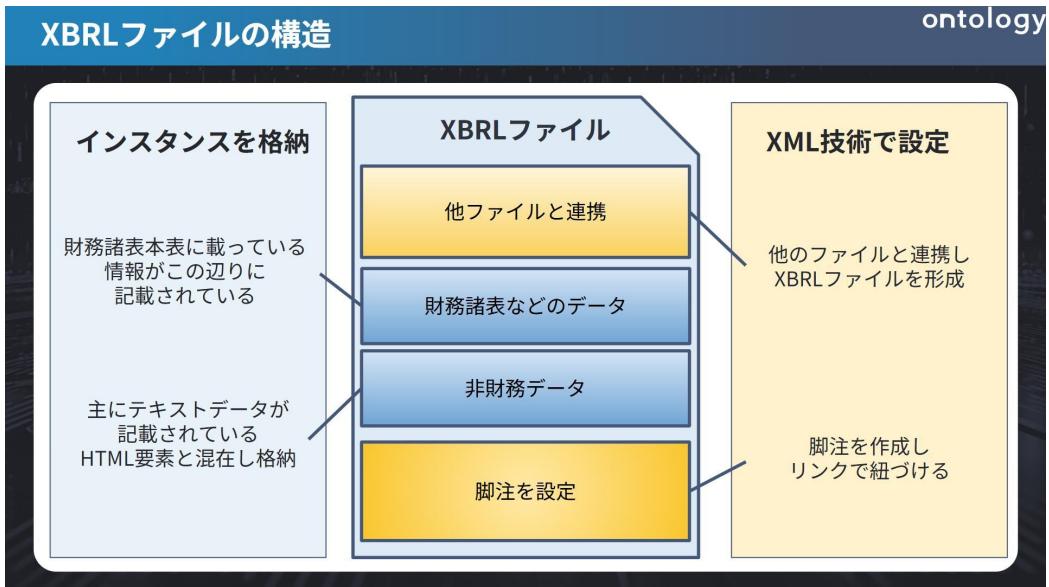
XBRLファイルの構造や、タクソノミとインスタンスをより詳しく理解することで、複雑なデータの中から必要なデータを探し出すことに繋がります。また、XMLを理解することで、表面上に記載されていない情報でさえ、XBRLフォルダー内にある情報であれば取得することができます。これらは、データを取得する場合に必ず必要となってくる知識です。

2.2 XBRLファイルについて

XBRLファイルは、とても多くの情報が記載されているため、数千行にも及ぶこともある大きなファイルです。ただし、現在の日本においては企業が直接XBRLファイルを作成することはできません。企業はiXBRL(オンラインXBRL)ファイルを作成し、それをEDINETに提出します。その手順が正しく踏まれると、EDINETがXBRLファイルに自動で変換し作成します。そのため、XBRLファイルがどのような形なのか知らない方が多い傾向にあります。

しかし、分析する上でXBRLファイルの中で何がどのあたりに記載されているかは理解すべきです。XBRLファイルは大きなファイルであるからこそ、目星をつけて読むことができなければ余計なものに惑わされ、分析が困難になります。

図2.1: XBRL ファイルの構造



XBRL ファイルの構造は、「インスタンスが格納されている部分」と「XML技術でXBRL ファイルを設定する部分」の大きくふたつに分けられます。

インスタンスが格納されている部分は、階層構造でタグの中に格納されています。その際、財務情報についてのインスタンスが前半に、非財務データは後半にまとまっているという構成です。

XML技術で設定する部分では、様々な設定がされています。そもそも XBRL フォルダーの中は、XBRL ファイルひとつだけではありません。実際に XBRL フォルダーを EDINET からダウンロードしていただければよくわかりますが、XBRL は複数のファイルによって構成され、その中の多くは XML ファイルです。XBRL ファイルの上部ではこれら多くの XML ファイル同士を連携させ、それぞれの項目に意味を持たせています。また、その書類によっては脚注を振る場合もあるかと思います。この脚注は、XBRL ファイルの下部にすべてまとめて記載されています。これらは XML の技術を用いて、それぞれ適切なものと結びつけることで、XBRL ファイルは構成されています。

2.3 タクソノミの種類

XBRl を構成する要素のひとつであるタクソノミは、XBRL ファイル内の多くで用いられます。タクソノミはデータを格納するひな形であるため、インスタンスだけでなく、XML で設定する部分にも用いられます。その結果、数千行に及ぶ XBRL ファイルの至るところでタクソノミが使用されています。また、XBRL ファイル内ではタクソノミが担う役割が複数あり、種類も細かく分類することができます。まず、大きく分けると「EDINET タクソノミ」と「提出者別タクソノミ」のふたつに分けられます。

2.3.1 EDINETタクソノミ（標準タクソノミ）

EDINETタクソノミはEDINETが設定しているタクソノミであり、多くの企業が使用する項目に振られているタクソノミです。売上高や営業利益などが例に挙げられ、印刷会社のシステム上で絶対に記載しなくてはならない項目にはEDINETタクソノミが振られています。勘定科目以外にも、テキストデータにかかる項目であっても記載する重要度が高いものに関しては、EDIENTタクソノミが振られています。このEDINETタクソノミは、EDINETからタクソノミ要素リストや勘定科目リストをダウンロードすることで確認できます。

また、EDINETタクソノミは「内閣府令タクソノミ」「財務諸表本表タクソノミ」「国際会計基準タクソノミ」「DEIタクソノミ」の4つの種類に分けられます。EDINETタクソノミであるため、全て金融庁が提供しているタクソノミです。これらは何か制度変更などが起こった場合、すべてを修正する必要がなく、その変更された制度に関わるタクソノミを修正するだけでいいという保守性の観点より区別されています。それぞれがどのようなものなのか、説明します。

2.3.1.1 内閣府令タクソノミ

提出書類全体のうち、注記事項を含む国際会計基準（IFRS）と日本基準の財務諸表本表のふたつ以外に関わるタクソノミです。各種内閣府令を対象としたタクソノミであり、内閣府令に応じてさらに開示府令タクソノミなどの分割単位があります。

2.3.1.2 財務諸表本表タクソノミ

日本基準財務諸表本表を対象としたタクソノミです。なお、注記事項の要素で財務諸表本表で利用されることがないものは、内閣府令タクソノミのひとつである開示府令タクソノミに含まれ、財務諸表本表タクソノミには含まれません。

2.3.1.3 国際会計基準タクソノミ

注記事項も含む国際会計基準（IFRS）財務諸表に関わるタクソノミです。日本会計基準と異なり、国際会計基準のタクソノミは、主に勘定科目のタクソノミに関わり、注記事項など補足情報であれば国際会計基準タクソノミに関わるのが国際会計基準タクソノミです。

2.3.1.4 DEIタクソノミ

提出書類の基本情報（Document Information）及び開示書類等提出者の基本情報（Entity Information）が格納されているタクソノミです。DEIは格納されているデータ「Document and Entity Information」の頭文字を取ってDEIと呼ばれ、そのタクソノミとして、DEIタクソノミと呼ばれます。

2.3.2 提出者別タクソノミ

提出者別タクソノミは、金融庁が設定したタクソノミではなく、各企業がそれぞれで設定するタクソノミです。事業は各企業によって異なるため、それらを正確に報告するには大多数の企業で使用される項目の他、企業は独自の項目を拡張して有価証券報告書に記載する必要があるためです。

そのため、すべての有価証券報告書に提出者別タクソノミの作成義務があります。

それぞれ別の業態で成立している数多の企業がみな有価証券報告書というひとつのフォーマットで財務報告ができているのは、この提出者別タクソノミで各企業に合わせて拡張できるためです。

提出者別タクソノミの例として、宝くじで発生するお金や一流スポーツ選手の移籍金などがあります。これらは多額の金額が移動するにも関わらず、多くの企業では使用されない勘定科目ですので、EDINET タクソノミにはありません。しかし、多額の金額が動く以上、有価証券報告書に明記しなくてはならないため、各企業が独自で勘定科目についてタクソノミを作成する必要があります。このように、特定の企業のみで使用されるのが提出者別タクソノミです。

また、複数の企業が使用している項目であっても、提出者別タクソノミの場合もあります。その場合、各企業で提出者別タクソノミをそれぞれ作成しているため、タクソノミが一致していない可能性が高いです。その場合、同じ項目であっても同じタクソノミで一括取得ができないので、注意が必要です。

2.4 タクソノミの更新要素

1章では、毎年タクソノミを確認する必要があるという問題点に触れました。タクソノミは事業や経理に関連して振られているものです。時代が変わればタクソノミも変化するため、常に最新の情報を活用しなくてはなりません。EDINET タクソノミは毎年公表されおり、最新の EDINET タクソノミを確認する方法について解説します。

2.4.1 要素リスト・科目リストの最新情報を取得

以下のリンクより、EDINET の「各種情報検索サービス」のページにアクセスします。

<https://www.fsa.go.jp/search/index.html>

図2.2: EDINET 各種情報検索サービス

各種情報検索サービス

EDINET (電子開示)

- [EDINETについて](#)
 - ・ [【閲覧サイト】](#)
 - ・ [【提出サイト】](#)
- [EDINETに関するお問い合わせ](#)
- [EDINETの運営状況について](#)
- [EDINETタクソノミの知的所有権について](#)

新着情報

- ・ 2025年版EDINETタクソノミの公表について[2024年(令和6年)11月12日]
- ・ 2025年版EDINETタクソノミ(業)の公表について[2024年(令和6年)8月9日]
- ・ 「EDINETタクソノミ年次更新に係る意見交換会」のご案内[2024年(令和6年)2月8日]
- ・ 2024年版EDINETタクソノミの公表について[2023年(令和5年)12月11日]
- ・ 2024年版EDINETタクソノミ(業)の公表について[2023年(令和5年)9月14日]

公表済のEDINETタクソノミ

- ・ [2023年版EDINETタクソノミ](#) [2024年(令和6年)1月12日]
- ・ [2024年版EDINETタクソノミ](#) [2023年(令和5年)12月11日]
- ・ [2023年版EDINETタクソノミ](#) [2022年(令和4年)11月8日]
- ・ [2022年版EDINETタクソノミ](#) [2021年(令和3年)11月9日]
- ・ [2021年版EDINETタクソノミ](#) [2020年(令和2年)11月10日]

これ以前のものを含むEDINETタクソノミの一覧は、こちら。

▶ EDINETの高度化に関する協議会 実施実績会合

各種窓口のご案内

- ・ 金融サービス利用者相談室
- ・ 金融行政モニター
- ・ 情報公開等
- ・ パブリックコメント
- ・ 申請・届出・照会
- ・ 入札公告等
- ・ 採用情報

新着情報配信サービス

- ・ [金融庁ソーシャルメディアアカウント](#)
- ・ [関連リンク](#)
- ・ [SEC 証券取引等監視委員会](#)
- ・ [CPAABC 公認会計士・監査審査会](#)

公表済みのEDINETタクソノミの欄から最新のEDINETタクソノミのページに行くことで、公表資料の閲覧が可能です。

公表資料には、更新概要の他に最新のEDINETタクソノミの概要を説明する資料や、EDINETタクソノミに設定されている要素一覧表示したものなどがあります。本書では「タクソノミ要素リスト」「国際会計基準タクソノミ要素リスト」「勘定科目リスト」を参照し、タクソノミを確認します。さらにXBRLの仕組みについて知りたい場合は、XBRL作成ガイドの「提出者別タクソノミ作成ガイドライン」や「フレームワーク設計書」を読むことを推奨します。

図2.3: 要素リスト種別

日本会計基準のタクソノミ	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">(e) タクソノミ要素リスト (EXCEL:2,861KB)</td><td style="padding: 5px;">(非財務データのタクソノミ)</td></tr> <tr> <td colspan="2" style="padding: 5px;">EDINETタクソノミ（財務諸表表タクソノミ及び国際会計基準タクソノミを除く。）に設定されている要素を一覧表示したものです。</td></tr> </table>		(e) タクソノミ要素リスト (EXCEL:2,861KB)	(非財務データのタクソノミ)	EDINETタクソノミ（財務諸表表タクソノミ及び国際会計基準タクソノミを除く。）に設定されている要素を一覧表示したものです。	
(e) タクソノミ要素リスト (EXCEL:2,861KB)	(非財務データのタクソノミ)					
EDINETタクソノミ（財務諸表表タクソノミ及び国際会計基準タクソノミを除く。）に設定されている要素を一覧表示したものです。						
IFRSのタクソノミ	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">(f) 勘定科目リスト (EXCEL:1,397KB)</td><td style="padding: 5px;">(財務データのタクソノミ)</td></tr> <tr> <td colspan="2" style="padding: 5px;">EDINETタクソノミのうち、財務諸表表タクソノミに設定されている勘定科目を一覧表示したものです。</td></tr> </table>		(f) 勘定科目リスト (EXCEL:1,397KB)	(財務データのタクソノミ)	EDINETタクソノミのうち、財務諸表表タクソノミに設定されている勘定科目を一覧表示したものです。	
(f) 勘定科目リスト (EXCEL:1,397KB)	(財務データのタクソノミ)					
EDINETタクソノミのうち、財務諸表表タクソノミに設定されている勘定科目を一覧表示したものです。						
	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="padding: 5px;">(g) 国際会計基準タクソノミ要素リスト (EXCEL:305KB)</td><td style="padding: 5px;">(主に財務データ、一部注記など)</td></tr> <tr> <td colspan="2" style="padding: 5px;">EDINETタクソノミのうち、国際会計基準タクソノミに設定されている勘定科目等の要素を一覧表示したものです。</td></tr> </table>		(g) 国際会計基準タクソノミ要素リスト (EXCEL:305KB)	(主に財務データ、一部注記など)	EDINETタクソノミのうち、国際会計基準タクソノミに設定されている勘定科目等の要素を一覧表示したものです。	
(g) 国際会計基準タクソノミ要素リスト (EXCEL:305KB)	(主に財務データ、一部注記など)					
EDINETタクソノミのうち、国際会計基準タクソノミに設定されている勘定科目等の要素を一覧表示したものです。						

EDINETタクソノミは、要素リストと科目リストにて確認します。これらは簡単に言うとEDINETタクソノミの一覧で、毎年公表されています。要素リストには「タクソノミ要素リスト」と「国際会計基準タクソノミ要素リスト」のふたつが存在し、科目リストは正しくは「勘定科目リスト」と

言います。

タクソノミ要素リストは、財務諸表本表タクソノミと国際会計基準タクソノミ以外のEDINETタクソノミに設定されている要素が一覧表示されています。つまり、ほとんどの非財務データと基本情報に関するデータのタクソノミがここで確認することができます。なお、有価証券報告書についてのタクソノミは、タクソノミ要素リストのシート9にて確認ができます。逆に財務データのタクソノミを探したい場合は、勘定科目リストで財務諸表本表タクソノミに属するタクソノミを確認できます。

国際会計基準タクソノミ要素リストは、国際会計基準タクソノミに設定されている要素が一覧表示されています。ただし、基本的に国際会計基準（IFRS）の勘定科目にかかるタクソノミがほとんどであり、一部の中期を除く非財務データに関しては、EDIENTタクソノミ要素リストに含まれていることが多いです。

この3つでほとんどのタクソノミは確認できるでしょう。逆に言えば、要素リスト・科目リストに記載されていない勘定科目や項目に関しては、提出者別タクソノミで振られている要素と判断してください。

2.5 インスタンスの持つ付加情報

1章でインスタンスは実データであり、私たちが知りたい情報そのものと説明しました。XBRLでインスタンスは、タクソノミのようにEDINETインスタンスなどは存在せず、インスタンス1種類だけです。企業はこのインスタンスだけをまとめて報告する「報告書インスタンス」や「(XBRL)インスタンス文書」を提出します。これらも意味合いとしては同じなため、インスタンスと一括りにされることが多いです。

インスタンスについて、より厳密に言えば実データの部分のみをfactと呼びます。逆に言えば、インスタンスには実データ以外の情報も含まれているということです。財務データと非財務データ、それぞれのインスタンスにはどんな付加情報があるのか解説します。

2.5.1 財務データ

財務データは、比較的多くの情報がインスタンスファイルの中で定義されています。

たとえば、期間や通貨単位、有効桁数などが階層構造で定義されています。そのため、同じ有価証券報告書の中で本年度と昨年度の財務データが混在することや桁数の調整、通貨単位などさまざまな指標について柔軟に対応が可能になっています。

リスト2.1: 財務データのインスタンス

```
1: <!-- 財務データのインスタンスのイメージ -->
2: <jppfs_cor:NetSales
3:   contextRef="CurrentYearDuration"
4:   decimals="-6"
5:   unitRef="JPY" >
6:   1234000000
```

```
7: </jpffs_cor>
```

一行目のNetSalesが、売上高を示すタクソノミのタグです。タグを指定することで、タクソノミに対するインスタンスを参照します。2行目のcontextRefは期間などを定義します。今回であれば「CurrentYearDuration」で”当年度の期間”つまり、「今期」を意味します。3行目のdecimalsは「どの桁まで正確か（どの程度丸めているか）」を示す有効桁数を定義します。今回は -6 であるため、桁数は百万単位は正確である（有効である）ことを意味します。最後にunitRefです。これで通貨単位を定義し、JPYは円を意味します。つまり、このインスタンスは「本年度の売上高は日本円で1,234,000,000円（表示上は1,234）」であるという情報がインスタンスに含まれています。

2.5.2 非財務データ（テキストデータ）

非財務データ、特にテキストデータに関しては財務データと異なり、インスタンス側で定義するのは期間などを表すcontextRefのみです。しかし、情報自体のボリュームが多いため、実際にファイルを見ると、財務データよりも多くの情報が入っているように見えます。

リスト 2.2: 非財務データのインスタンス

```
1: <!-- テキストデータのインスタンスイメージ -->
2: <jpcrp_cor:BusinessRisksTextBlock
3:   contextRef="FilingDateInstant">
4:     &lt;p style="page-break-before:always; line-height:0.75pt; width:100%; font-size:0.75pt;"&gt; &lt;/p&gt;
5:     &lt;h3 class="smt_head2" style="font-family:&apos;MS ゴシック&apos;;" &gt;タイトル &lt;/h3&gt;&lt;p class="smt_text2" style="orphans:0;widows:0;text-align:justify;padding-left:9pt;padding-right:0pt; margin:0pt 0pt 0pt 0pt; letter-spacing:0pt;line-height:15pt;font-size:9pt;"&gt;「テキスト1」
6:     &lt;/p&gt;&lt;p class="smt_text2" style="orphans:0;widows:0;"&gt;「テキスト2」
7:     &lt;/p&gt;&lt;p class="smt_text6" style="orphans:0;widows:0;"&gt;
8:       &lt;/p&gt;
9:     &lt;/p&gt;
10:    &lt;/p&gt;
11:  &lt;/p&gt;
12: </jpcrp_cor:BusinessRisksTextBlock>
```

contextRefのFilingDateInstantは報告書提出日が設定されており、その時点でのテキストデータを参照します。

情報の中身が文章である分、そもそもボリュームが大きいということもあります、それに加え、ブラウザー上で文字の大きさやフォント、配置などをHTMLとCSSで設定しており、これらの情報を文章とともにインスタンスの中で定義しているためデータ量がより大きくなり、人が直接読む

には困難な形式をしています。

また、インスタンスの中には「<」や「>」といった文字がしばしば見られますが、これらはHTMLの「<」「>」(タグ)を意味します。

HTMLだけでなく、XML・XBRLでも「<>」を使用します。ただし、「<>」の持つ意味はHTMLとXML・XBRLで異なります。それにも関わらず同じようにプログラムを書くと、機械はどちらの意味で書かれているのか区別できません。そのため、HTML・CSSの要素である「<>」を「<」や「>」といったエスケープ文字(代わりの文字)を使用することでこの問題を解決しています。

ただし、データを取得しようと思うと、XBRLのタグ内を取得する方法が一般的であるため、HTML・CSS要素も一緒に取得することになってしまいます。そのため、実際にデータ取得する際には、この部分を除去する工程が加えられることが一般的です。

2.6 XMLの特徴と関係性

XBRLは、XMLという言語で構成されています。構成されているというよりも、XMLで財務情報を報告するためにチューニングしたものがXBRLと言った方がイメージしやすいかもしれません。XBRLの「タグを自分で指定できる」という特徴によるメリットについて1章で触れましたが、この特徴は実はXMLのものになります。このタグを自分で設定できる特徴により、あらゆる状況に柔軟に対応することが可能となりました。

電子財務報告では、各企業ごとの事業を正確に報告できるフォーマットを要します。ある程度の型を持つつ、拡張要素を持たせることで柔軟かつ正確に電子報告できるという点や、XBRLが生まれた当時の世界標準技術になりうる点などが影響し、XMLが採用されたのだと思います。XBRLファイルを読み解くということは、XMLを読み解くことと同義と言えるでしょう。XMLを理解することで、複雑なXBRLファイルをより楽に理解することができる他、表面上に記載されていない情報であれ、XBRLフォルダー内にある情報であれば取得できるといったことが望めます。たとえば、先ほど扱ったインスタンスの付加情報の取得や、提出者別タクソノミのタクソノミの確認などがあります。

XBRLが引き継いでいるXMLの特徴として、以下の4つが挙げられます。

1. タグを使う（自分で設定できる）
2. 階層構造を持つ
3. 拡張性がある
4. 属性を付与できる

2.6.1 タグを使う（自分で設定できる）

XMLは<>というマークアップ記号を使用します。<>のことをタグと言い、マークアップとはタグを埋め込む作業のことを指しています。

このタグの中に文字や数値などを囲むことで、構造を定義します。

リスト 2.3: タグのイメージ

```
1: <!--イメージ1-->
2: <要素名>インスタンス</要素名>
```

<>を開始タグと言い、</>を終了タグと言います。

HTMLなどと異なり、このタグを自分で自由に作成できることがXMLの特徴のひとつです。

2.6.2 階層構造を持つ

タグで入れ子にすることで階層構造にすることができるのも、XMLの特徴です。この入れ子構造を使うことで、入れ子構造で親子関係を作れることで、何がどこに分類されるのかなどが明確になります。

リスト 2.4: 階層構造の例

```
1: <!--イメージ2-->
2: <XML>
3:   <要素名1>
4:     <子要素1>子インスタンス</子要素1>
5:   </要素名1>
6: </XML>
```

以上のタグで記述すると、以下のような階層構造を示します。

- ・ XML
 - 要素名 1
 - ・ 子要素 1

XMLという親構造があり、要素名1が子要素、子要素1がさらに子要素、つまりは孫要素というような階層構造を定義しています。

財務諸表などを表す場合、Excelやcsvファイルで表すよりも階層構造で表した方がより正確に表すことができます。正確さが必要な財務報告において、XMLは相性がよかったです。

2.6.3 拡張性がある

XMLはユーザーが自由にタグを定義できるため、特定の用途やニーズに合わせることができます。タグを自由に定義することで、データ構造も自由に変えられます。この自由度の高さから財務報告用にカスタマイズされたのが、XBRLだと言えます。

世界基準に統一された記述方式により、あらゆるコンピューターシステムで利用が可能で、共有のしやすさや管理のしやすさがメリットです。

リスト 2.5: 拡張性

```
1: <!--イメージ3-->
2: <XML>
3:   <要素名1>インスタンス1</要素名1>
4:   <要素名2>インスタンス2</要素名2>
5:   <要素名3>
6:     <子要素1>子インスタンス</子要素1>
7:   </要素名3>
8: </XML>
```

ここでは新たに要素名1の兄弟要素として、要素名2、3を追加することで拡張しています。子要素も要素名1の子要素ではなく、要素名3の子要素に変更する形で拡張しました。このような拡張が簡単に行えることが、XMLの強みです。そのため、提出者別タクソノミのように企業ごとで異なる会計基準や、独自の項目にも対応できたのでしょう。

2.6.4 属性を付与できる

開始タグには、属性というものを定義できます。

この属性とは、インスタンスに付加的な情報を記述することを目的としています。これにより、インスタンスに追加の情報を付与できます。

たとえば、財務データに関するインスタンスであれば、その通貨が円なのかドルなのか、どの桁まで正確なのか、文書データは何かなどの情報を付与できます。

リスト 2.6: 属性の付与

```
1: <!--イメージ4-->
2: <XML 文書種別="有価証券報告書">
3:   <NetSalesSummaryOfBusinessResults 通貨="円">100</NetSalesSummaryOfBusinessResults>
4:   <NetAssetsSummaryOfBusinessResults 通貨="円">100</NetAssetsSummaryOfBusinessResults>
5: </XML>
```

上の例では、文書種別が有価証券報告書であり、それぞれNetSalesSummaryOfBusinessResultsは「売上高、経営指標等」、NetAssetsSummaryOfBusinessResultsが「純資産額、経営指標等」のことを指し、通貨が円という属性が付与されています。

このようにして、前述したインスタンス情報はXMLの構造化によって同じ形式でまとめています。

2.7 まとめ

本章では、XBRLファイルの構造、タクソノミの種類・インスタンスの付加情報などより深いXBRLの知識に加え、XMLの特徴について解説しました。

タクソノミの種類やインスタンスの付加情報などは、今後情報を取得する際に使用する知識でも

あるため、覚えておきたい項目です。また、XBRLが様々な企業の財務報告が可能なのは、XMLの幅広い拡張性によるものであり、その特徴を生かしているからこそ、少し複雑な部分があります。

この他にもXBRLはXMLの技術を使って成り立っていることが多くあるため、次章ではそんなXBRLで使用されているXMLの技術について解説します。ここを理解するのは少しハードルが高いかもしれません、理解できてしまえば理論上、有価証券報告書の情報であればすべて抜き出せるようになります。

第3章 XBRLで見られるXML（名前空間、XLink、XML Schema、XPointer）

3.1 本章の目的

本章では、XBRLで使用されているXML技術について解説します。XMLの技術とはどのようなものなのか、わかっていないと調べるのも難しいでしょう。仕様書などで理解するのに必要な、最低限の知識を身につけてもらうための章です。なるべく簡単に理解しやすく説明することを意識していますが、名前がややこしかったり、使用方法自体が複雑でわかりづらいため、一回で覚えることに固執せず、わからなくなったら立ち戻って徐々に理解してもらえばよいと思います。

本章で取り扱った内容よりも詳しい内容を学びたい方は、XBRLが参照するXMLの仕様書を読むことを推奨します。

W3C(World Wide Web Consortium)の「Extensible Markup Language (XML) 1.0 (Fifth Edition)」

3.2 名前空間

3章で取り扱ったように、XBRLやXMLではタグを自分で作成することができます。XBRLでは、この自分でタグを作成できるという高い柔軟性が提出者別タクソノミの作成など、複雑な財務報告を扱うことを可能にしています。ただし、同じ名前のタグ（要素名）が混在してしまった場合、衝突・重複など、思わぬ挙動を起こしてしまいます。それを防ぐためのものが、名前空間(namespace)です。

たとえば、A社では<address>（要素名）を「お客様の住所」を表すものとして使用し、B社でも<address>を使用していました。しかし、B社の<address>は「出荷先住所」を表しています。このように、どちらも<address>という要素名を使用していますが、それぞれ細部の構造や用途が異なる場合があります。単に両方を<address>として書くと、後から見た人や機械は「この<address>はなにを指しているの？」と混乱する可能性があります。名前空間は、こういった場合にタグを区別するために使用されます。

名前空間を使うには、まず名前空間宣言をします。作成するXMLのタグの中のxmlns属性に、名前空間を宣言します。xmlns属性では、名前空間接頭辞（プレフィックス）と名前空間URI（単にURIと表すこともある）をセットで定義することで区別をします。

名前空間接頭辞ことプレフィックスは文字通り、名前空間宣言をするときに頭につけることで、同じタグでも識別することができます。

URI (Uniform Resource Identifier) とは、インターネット上で特定のひとつを指すときに使われる記述です。この中のひとつにURLがあります。URIはネット上のリソース（ファイルやディレクトリー、サービスなど）を特定する「宛名」「住所」「名前」すべてを含む広い概念です。その一方で、URLはその中の「場所」だけを示すURIの一種です。つまり、URIの中でも”ネット上の

どこにあるか”を示す「道順」や「住所」がURLです。また、誤解を招かないための補足として、名前空間URIは「場所」ではないため、「アクセスできるURL」でない場合が多いです。あくまでURIは「一意に識別するための文字列」であるため、一意に絞り込めさえすれば、実在するサイトでなくても構いません。

リスト3.1: 名前空間の例

```
1: <!-- 名前空間の例 -->
2: <!-- A社の住所情報 -->
3: <cust:address xmlns:cust="http://example.com/customer">
4:   <cust:zip>123-4567</cust:zip>
5:   <cust:city>Tokyo</cust:city>
6: </cust:address>
7:
8: <!-- B社の住所情報 -->
9: <order:address xmlns:order="http://example.com/order">
10:  <order:zip>987-6543</order:zip>
11:  <order:city>Osaka</order:city>
12: </order:address>
13:
```

「プレフィックス + “：“ + 要素名」で、名前空間を表すことができます。

A社の「cust:address」であれば、プレフィックスのcustは「xmlns:cust=” http://example.com/customer”」と結びつけた短縮形を指し、要素名であるaddressと組み合わせることで「http://example.com/customer + address」となります。これを省略した形が名前空間で使われる「cust:address（プレフィックス + “：“ + 要素名）」の形です。つまり、同じaddressというタグを使用していても cust:addressとorder:addressでは結び付いているURIが異なるため、識別可能であるということです。

この仕組みによって、もともとあるタグと独自のタグを織り交ぜて柔軟に運用ができます。EDINETタクソノミの4つの種類ごとにプレフィックスが分かれているため、会計基準によってタグが異なるという問題も、この名前空間を使用することによって区別されています。逆にプレフィックスを見れば、その要素がどのEDINETタクソノミの分類に属しているかもわかります。

3.3 XML Schema

XML Schemaは、XML文書の要素や属性の種類、順序、数、データ型などを詳細に指定することができます。これによりXMLデータの整合性を保つことができます。また、XML Schemaは「xsd」プレフィックスを定義することにより、スキーマファイルを作成します。「xsd」プレフィックスは要素を定義したり、importするのに使用され、XBRLデータにとってとても重要な要素です。

3.3.1 スキーマファイル

スキーマファイルはXML Schemaによって作成されるもので、XBRL用に作成されたものです。内容として、インスタンスの語彙（要素名や属性など）を定義しています。具体的な勘定科目名や注意事項などの項目が定義されます。EDINETでは、スキーマの拡張子は「.xsd」です。スキーマファイルには、タクソノミスキーマ、目次項目アイテムスキーマ、パート要素スキーマ、ロールタイプスキーマなどがあり、タクソノミスキーマの中には語彙スキーマ、廃止要素スキーマがあります。この中でも、今回はより重要な語彙スキーマについて解説します。

3.3.1.1 語彙スキーマ

語彙スキーマ（コアスキーマ）では、提出書類で利用される報告項目又は勘定科目が要素として定義されます。語彙スキーマで定義した要素をリソースとして、拡張リンクで関係を定義する事が多くあります。これについては、XLinkの部分で詳しく説明します。

図3.1: 語彙スキーマの要素



語彙スキーマでは、以上の内容を定義します。

語彙スキーマは、「EDINET タクソノミの分割単位」ごとにそれぞれ存在します。EDINET タクソノミの分割単位とは、たとえば内閣府令タクソノミの場合は「jpcrp」、財務諸表本表タクソノミの場合は「jppfs」などのプレフィックスを指します。これらすべてに語彙スキーマが存在し、ファイル名では「jpcrp_cor_2024_08-01.xsd」のように表記されます。このうち「cor（コア）」の部分が語彙スキーマを示しており、そこからどのような情報がどこで定義されているかを遡ることが可能です。

さらに、EDINET タクソノミでは標準的な項目があらかじめ定義されているため、金融庁が定める項目を変更してはなりません。一方で、EDINET タクソノミの語彙スキーマに含まれない項目を拡張したい場合は、提出者別タクソノミを用いて別途定義できるため、柔軟性は確保されています。

このような理由から、EDINET タクソノミの語彙スキーマを直接編集することは推奨されません。

3.3.2 XLink

XLink (XML Linking Language) は XML 文書同士の関連付け（リンク）を定義するための言語です。XLinkには、リソース（resource）とリンク（link）というふたつの主要概念があります。リソースとは、ファイルや画像など、URIなどで参照可能な情報やサービスのひとつの単位を指します。リンクとは、どのリソースとリソースを繋ぐのか、関連性はどのようなものなのかを定義する役割があります。

XLinkを使ってリンクを定義するには、「xmlns:xlink=http://www.w3.org/1999/xlink」の名前空間宣言が必要になります。名前空間宣言を行うことで、グローバル属性を XML 文書内で定義したどんな要素に対しても利用することが可能になります。

また、XLinkは大きく分けて単純リンクと拡張リンクの2種類があります。単純リンクはハイパーリンクの機能があります。これは普段ウェブページなどで目にするリンク（URL）と同一のものです。ハイパーリンクは主に人に見せるためのリンクです。拡張リンクは、複数のリソースの役割や相互関係を同じリンクの中で一括して扱える機能があります。

単純リンクと拡張リンクについて、より詳しく解説します。

3.3.3 単純リンク（Simple Link）

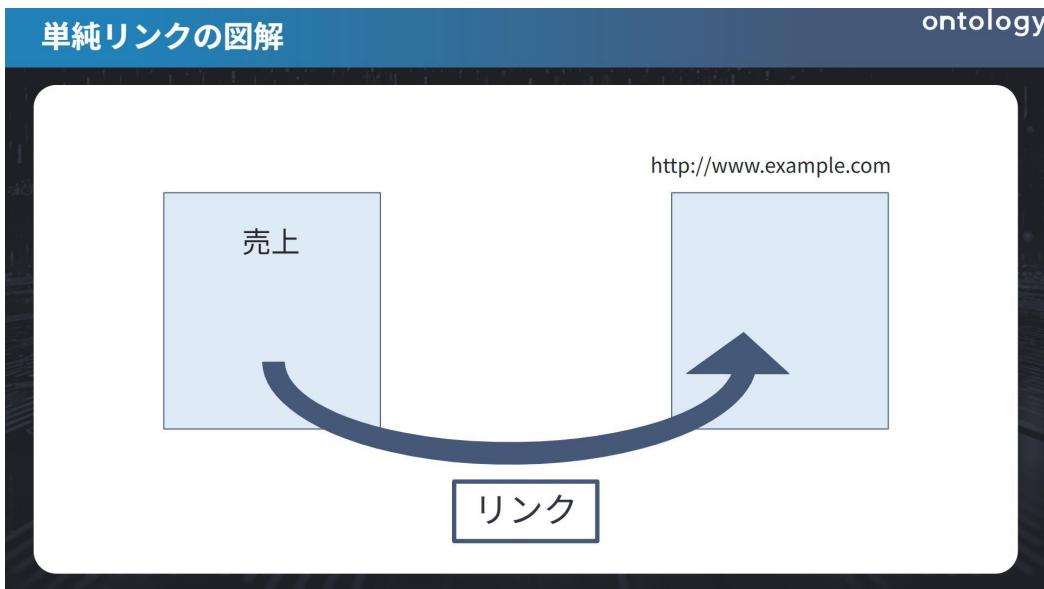
リソース同士を1対1の関係で定義できます。単純リンクは普段ウェブページなどで目にするリンク（URL）やHTMLのaタグに似ており、1対1の一方向のリンクを形作ります。リンクが設置してあるリソースから指定しているリソースへ飛ばす一方向の関係を作ります。

リスト 3.2: 単純リンクの例

```
1: <!-- 単純リンクの例 -->
2: <xbrli:xbrl xmlns:xlink="http://www.w3.org/1999/xlink">
3:   <link xlink:type="simple"
4:     xlink:href="http://www.example.com">
5:     売上
6:   </link>
7: </xbrli:xbrl>
```

単純リンクを定義するにはlink要素にxlink:type="simple"を指定し、xlink:href="任意のURI"のように使用します。

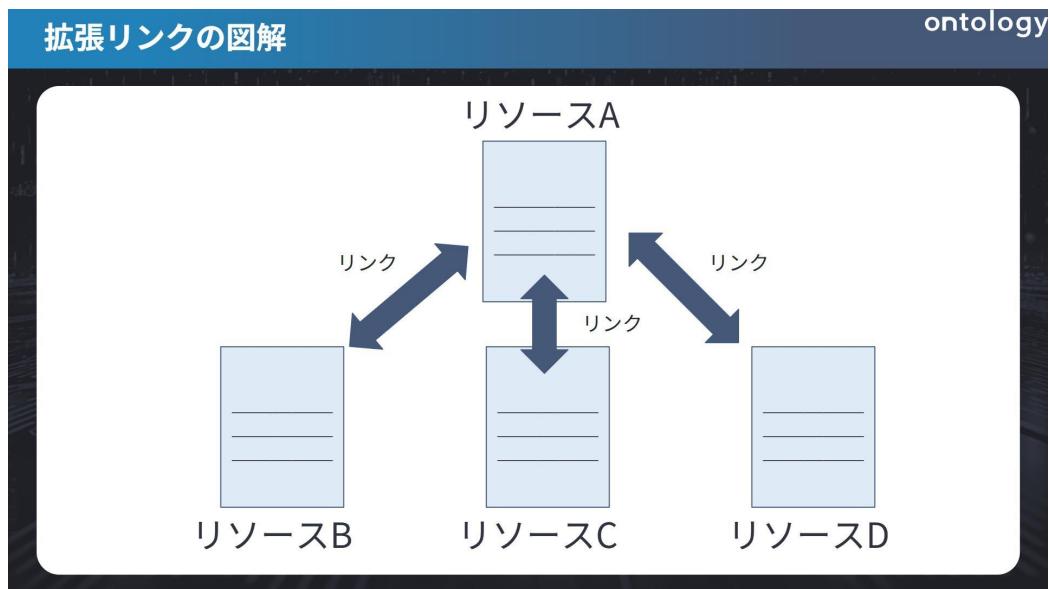
図3.2: 単純リンクの図解



単純リンクのイメージとして、XLinkのコードが書かれたリソースからURI先のリソースに向かってリンク（一方通行）の橋をかけていると考えてください。ハイパーテリンクの機能があります。

3.3.4 拡張リンク (Extended Link)

図3.3: 拡張リンクの図解

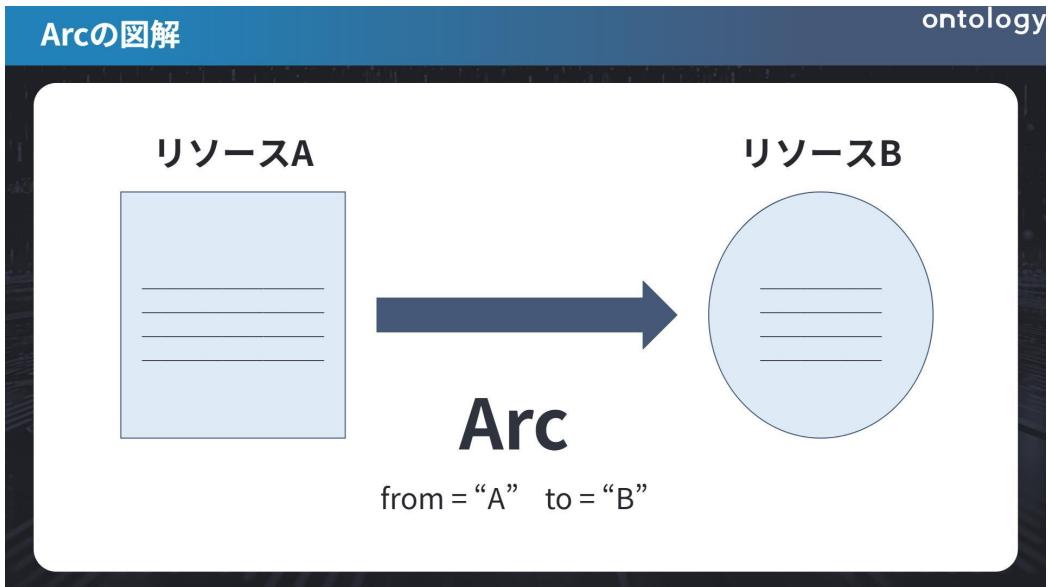


単純リンクでは1対1で一方向のシンプルな関係でしたが、拡張リンクでは、複数のリソース同士をまとめて結び付け、それがどんな関係かを一度に定義できます。双方向のリンクや複雑な関係性を定義できます。多数のリソースを一括して扱えることでリソース同士を一元管理し、その関係性を細かく定義できます。XBRLでは、この拡張リンクを利用して概念とラベルの関係や、概念同士の算術関係などを表現しています。

XLinkでは、`xlink:type="extended"` とすることで拡張リンクを定義できます。extended型要素の配下にlocator型要素、resource型要素、arc型要素をいくつでも定義できるようになります。これらを組み合わせて、複数のリソース間でリンクの設定や双方向リンクなど、複雑な関係の定義ができます。要素としてはtitle型要素というものもありますが、まずは初めの3つが重要です。リンクの対象となるリソースが初めはありません。そのため、リンクにリソースを加える必要があります。リンク要素にリソースを加えることを「リンクにリソースを参加 (participate) させる」と言います。また、リソースにもXLinkのリンク要素内に直接埋め込まれているリソースであるローカルリソースとXLinkのリンク要素外の別の場所にあるリソースであるリモートリソースという2種類があります。

ローカルリソースを参加させる場合には、`xlink:type="resource"` (resource型) を使用します。resource型ではリンクベースファイル内ではlabel要素やreference要素、footnote要素として定義されます。日本語や英語でのラベル(表示名)や参照情報、脚注などのリソースを作ることができます。このあたりのリソースは自分で作成します。リモートリソースを参加させる場合には、`xlink:type="locator"` (locator型) を使用します。リンクベースファイルでlink:loc要素として定義されます。

図3.4: Arcの図解



最後に `xlink:type="arc"` (arc型) はリンクベースファイルで「○○Arc要素」のように定義されます。この要素では、リソース同士をリンクで繋ぐことができます。そのため、resource型要素やlocator型要素同士を繋ぎ、そのリンクの意味もあわせて定義できます。イメージとして画像のようにひとつのリソースからもうひとつのリソースに対してリンクを繋ぎます。1対1の場合、図のようになりますが、1対多や多対多の場合、矢印とリソースの数が増えたり、矢印が双方向になったりします。Arcには"arcrole"という属性があり、Arc同士をどういう役割で結び付けるかを定義できます。たとえば、親子関係やラベル付与などを示す場合に使います。

リスト3.3: 拡張リンクの例

```
1: <!-- 拡張リンクの例 -->
2: <xbrli:xbrl xmlns:xlink="http://www.w3.org/1999/xlink">
3:   <link:loc xlink:type="locator"
4:     xlink:href="../jpigp_cor_2023-12-01.xsd#jpigp_cor_CashAndCashEquivalentsIFRS"
5:     xlink:label="CashAndCashEquivalentsIFRS"/>
6:
7:   <link:label xlink:type="resource"
8:     xlink:label="label_CashAndCashEquivalentsIFRS"
9:     xlink:role="http://www.xbrl.org/2003/role/label"
10:    xml:lang="ja"
11:    id="label_CashAndCashEquivalentsIFRS">現金及び現金同等物</link:label>
12:
13:   <link:labelArc xlink:type="arc"
14:     xlink:arcrole="http://www.xbrl.org/2003/arcrole/concept-label"
15:     xlink:from="CashAndCashEquivalentsIFRS"
```

```
16:         xlink:to="label_CashAndCashEquivalentsIFRS"/>
17:     </xbrli:xbrl>
```

上記のコード例を用いると、まず名前宣言をし、XLinkを使用することを宣言したのち、ひとつ目のブロックで別のファイルから hrefで場所を指定し、「CashAndCashEquivalentsIFRS」というラベル（名称）をつけます。これがひとつのリモートリソースです。ふたつ目のブロックで同じファイル内に存在する idが、「label_CashAndCashEquivalentsIFRS」のものに対し「現金及び現金同等物」というラベルをつけています。これがローカルリソースです。最後にこれらを arc型でリソース同士の関係を定義します。今回であれば、リモートリソースとローカルリソース両方の本質的な意味は同じものであるが、日本語では「現金及び現金同等物」、英語では「CashAndCashEquivalentsIFRS」というラベルを使用するということを定義しています。

このようにひとつずつ定義していくことで、タクソノミのタグを変えずに複数の言語に対応した開示書類を作成しています。

3.3.5 リンクベース（リンクベースファイル）

リンクベースは、XLinkを応用し作成されたXBRL独自の技術です。”リンクのデータベース”の意味を持ち、拡張リンクを集めたファイルです。データベースと言ってもマトリクス的な格納をしているわけではなく、拡張リンクの例で挙げたようなブロックがいくつも続いて、ひとつのファイルになっています。リンクベースファイル（リンクベース）はリソースを拡張リンクを利用し作成され、拡張リンクにより各リソースが多対多の関係で結びつけられています。

リンクベースには、ラベルリンクベース、参照リンクベース、定義リンクベース、計算リンクベース、参照リンクベース、ジェネリックラベルリンクベースといいくつかの種類があります。定義リンクベースは、主にディメンションの定義をします。参照リンクベースは、参照している法令や規則の内閣府令などへの参照情報が定義されています。計算リンクベースは、勘定科目間の計算関係を定義しています。あくまで計算関係を定義しているだけで、実際の計算機能はないのが特徴です。ジェネリックラベルリンクベースは、提出者別タクソノミで各自が追加する拡張リンクロールに対してその英語名称を定義します。これらについてさらに知りたい方は、金融庁が公表している「提出者別タクソノミ作成ガイドライン」を参照すると、より理解が深まると思います。

以降は特に重要な名称リンクベース、表示リンクベース、定義リンクベースについて詳しく解説します。

3.3.5.1 ラベル（名称）リンクベース

ラベルリンクベース（名称リンクベース）は、スキーマで定義される要素に対して名前（ラベル）を定義する役割があるものです。同じ要素に対して複数の言語でラベルを付けることが可能であり、用途に応じて異なるラベルを付与することもできます。その中でも日本語と英語の名称については、どちらも定義する必要があります。先ほど拡張リンクで定義したものが、まさにラベルリンクベースの一部です。

後ほど触れる語彙スキーマと混同しやすいのですが、主な違いとしては、語彙スキーマにない情

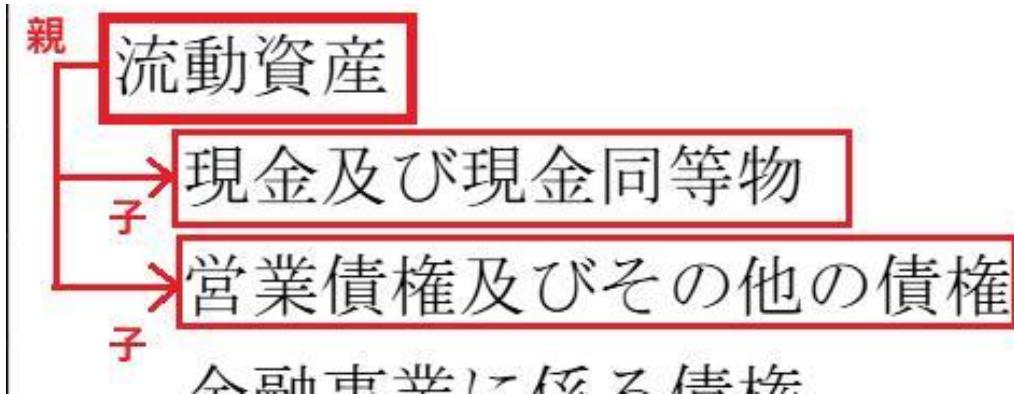
報に対してはラベルリンクベースでは名称をつけることはできないということです。ラベルと言っているため、語彙スキーマにある要素（物体）にラベル（シール）を貼っていると考えると、イメージしやすいかもしれません。

3.3.5.2 表示リンクベース

表示リンクベースには、様式ツリー又は詳細ツリーとして報告項目及び勘定科目の表示上の関係が定義されています。様式ツリーと詳細ツリーについては、後ほど詳しく説明します。

表示リンクは、要素間の親子関係を表現しています。親子関係を表現するために複数のリソースを結び付ける複雑なリンクが必要になり、拡張リンクはその関係を定義する性質を持っています。

図 3.5: 親子関係の例



画像のように、財務諸表では資産の部の中に流動資産という項目があり、その中に「現金及び現金同等物」や「営業債権及びその他の債権」などの勘定科目があります。こういった親子関係の元、財務諸表は成り立っています。表示リンクベースは、この関係を正しく表示するためのものです。また、有価証券報告書の目次項目にも親子関係を用いて表現されています。

図3.6: トヨタ自動車有価証券報告書の目次項目

<u>本文</u>
第一部 <u>企業情報</u>
第1 <u>企業の概況</u>
1 <u>主要な経営指標等の推移</u>
2 <u>沿革</u>
3 <u>事業の内容</u>
4 <u>関係会社の状況</u>
5 <u>従業員の状況</u>
第2 <u>事業の状況</u>
1 <u>経営方針、経営環境及び対処すべき課題等</u>
2 <u>サステナビリティに関する考え方及び取組</u>
3 <u>事業等のリスク</u>
4 <u>経営者による財政状態、経営成績及びキャッシュ・フローの状況の分析</u>
5 <u>経営上の重要な契約等</u>
6 <u>研究開発活動</u>
第3 <u>設備の状況</u>

「第一部 企業情報」の中に第1、第2とされ、さらにその中に1、2、3…と続いていきます。これらも親子関係になっており、第一部と第1が親子関係、第1と1がさらに親子関係になっています。この目次にある親子関係をまとめたものが様式ツリーであり、その中にある勘定科目などの詳細な親子関係まで表したものが詳細ツリーです。表示リンクベースでは、これらの関係を定義しています。

また、表示リンクでは、ラベルリンクベースで定義したラベルの切り替え設定も行うことができます。

3.3.6 拡張リンクロール (Extended Link Role)

拡張リンクロール (ELR) は、構造化データである XBRL データを形作る上で非常に重要な概念です。

拡張リンクロールの「拡張リンク」は、先ほど取り扱った拡張リンクです。拡張リンクロールは拡張リンクの role (`xlink:type="role"`) 属性値です。W3C の XBRL Spec2.1 によると、拡張リンクロールはアプリケーションが、個別の関係ネットワークに分割するために使用する値です。ここで言う個別の関係ネットワークとは、貸借対照表 (Balance Sheet) などを指します。ただし、各貸借対照表には複数の勘定科目がありますが、勘定科目間の親子関係は表示リンクで定義しなくてはなりません。このときに、連結と個別の項目の親子関係を一度に定義するよりも、拡張リンクロールを連結と個別で分けて各ネットワーク内で親子関係を定義したほうが整理しやすくなります。また、

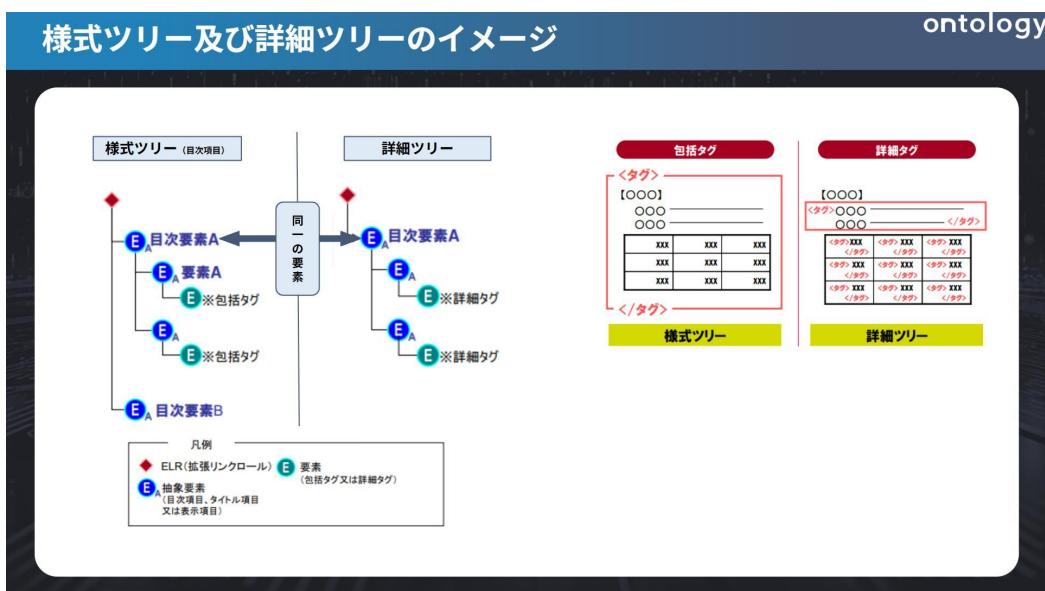
拡張リンクロールには、研究開発や役員の状況、セグメント情報など特定の目次や項目ごとに関するものが設置されています。この中でも、各目次項目を束ねる開示様式を特定するための特徴を持つものは、主に様式ツリーと詳細ツリーの定義に関わります。

3.3.6.1 様式ツリーと詳細ツリー

様式ツリーとは、提出書類全体のツリー構造を表したもので、提出書類様式ごとにあります。イメージとして有価証券報告書の目次項目は、大項目と中項目など比較的重要なツリー構造をまとめたものになっています。これが様式ツリーです。また、提出書類全体の目次項目の中でさらに詳細化が必要な部分については、詳細タグ付けをします。詳細タグ付けをしたものをツリー構造で表したものも詳細ツリーといいます。たとえば、財務諸表本表のツリー目次項目では「経理の状況」の中の「財務諸表等」の中の「財務諸表」であり、その中の売上高などの勘定科目は、財務諸表（目次要素の中の要素）に関わる詳細ツリーと言えます。

要するに、様式ツリー及び詳細ツリーとは、開示書類の全体構造を把握するために親子関係や表示順番を定義し、機会が処理しやすいように構造化しています。

図3.7: 様式ツリー及び詳細ツリーのイメージ



イメージ画像における目次要素は、EDIENTにおける目次項目にあたります。目次項目は府令様式及び財務諸表等規則等様式の隅付き括弧 ([]) で記載される項目に対応する場合が多いとされている項目です。有価証券報告書内でたとえると、【企業の概要】や【事業等のリスク】などがあります。これらは、有価証券報告書の目次に対応する項目となっています。目次項目とそれに対応する要素について、提出書類全体を網羅するツリー構造が様式ツリー、詳細なタグ付け対象の目次事に原則ひとつのみにフォーカスしたツリー構造を詳細ツリーと言います。つまり、詳細ツリーに関しては、目次内の記載すべてを網羅するというわけではありません。

iXBRLでは、様式ツリー・詳細ツリーごとにタグが付いており、このタグにも名前がついていま

す。それぞれ包括タグと詳細タグと言い、包括タグがテキストブロック型の要素に対し、詳細タグはテキストブロックの他に、文字列、文章や金額あるいは数値等を囲むタグが定義されています。XBRLでデータを取得する際はタグがひとつの単位になります。そのため、前述したようにそれぞれの勘定科目ごとに詳細タグが設置されている財務データは、科目ごとにデータ取得が可能です。ただし、テキストデータなどは包括タグで項目ごと括られていることが多いため、特定の文章のみの抽出をするのは困難です。

3.4 XPointer

XPointer (XML Pointer Language) は、XML形式のデータの一部を参照するための技術です。XLinkのリンクベースになくてはならない技術です。特に、語彙スキーマ等で定義した要素を外部から参照するために使用されます。

コード例を用いて説明します。

リスト3.4: XPointerの例（リンクベース）

```
1: <!-- XPointerの例 -->
2: <!-- サンプルファイル.xml（リンクベース） -->
3: <link:presentationLink xlink:type="extended" xlink:role="連結貸借対照表（例）">
4:   <xlink:loc
5:     xlink:type="locator"
6:     xlink:href="xlink:href=" sampleURI.xsd#sampleXPointer">
7: </link:presentationLink>
```

リスト3.5: XPointerの例（スキーマファイル）

```
1: <!-- XPointerの例 -->
2: <!-- sample.xsd（スキーマファイル） -->
3: <xsd:element name="Sample" id="sampleXPointer" ~~~/>
```

XPointerは、コード上では「URI + “#” + XPointer」で構成されています。たとえば「xlink:href=” sampleURI.xsd#sampleXPointer”」となっている場合、「sample.xsd」ファイルの中のid=「sampleXPointer」である部分を参照するという意味になります。また、#とsampleXPointerの部分を合わせてフラグメント識別子と呼びます。URIが相対URIの場合、省略され xlink:href=” フラグメント識別子”となる場合もあります。コード例を用いると、参照対象のデータが同じ書類内にある場合のみ、xlink:href=” #sampleXPointer”のように省略できます。

3.5 まとめ

初めて聞くような単語がとても多く出てきた章だと思います。本章で紹介した内容ですらほぼ紹介のみで、XBRLがどのように構成されているかなどまでは扱えていません。また、XML技術を応用してXBRLでのみ存在する技術もあるため大変混乱しやすく、学習コストも高くなってしまうで

しょう。しかし、これらを知っているのと知らないのでは、実際にデータ取得の幅が全く違ってきます。本章を読み直したり、金融庁から公開されている資料などでゆっくり噛み砕きながら理解していただければと思います。本章と次章で、仕様書を読むことができるようになっていただければいいと思っております。

次章では、コードを用いて、実際にXBRLではどのようにXML技術が応用されているのかを解説します。

第4章 XMLとタクソノミ

4.1 本章の目的

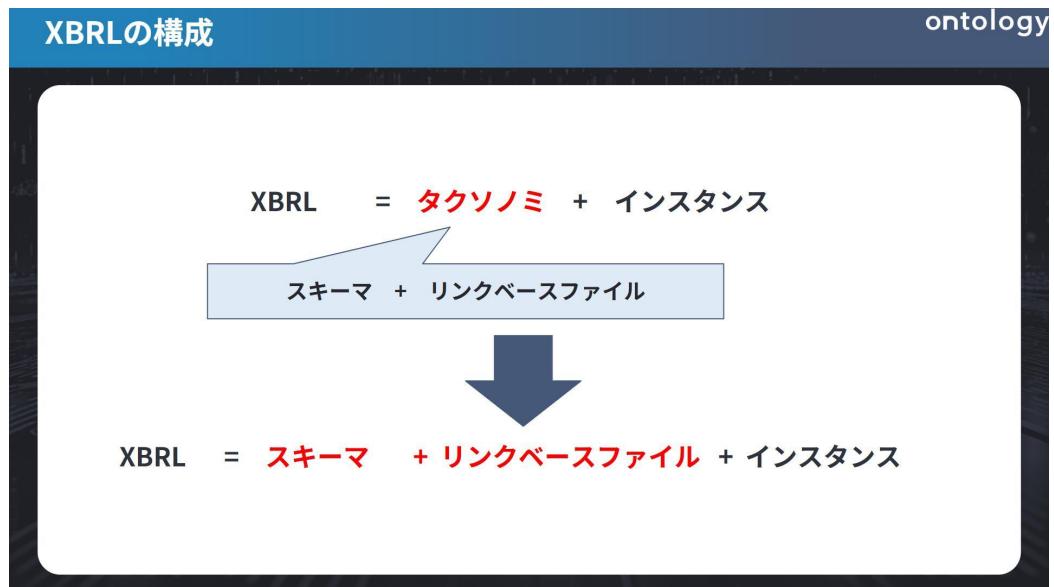
XBRLはXMLで構成されているため、当然タクソノミもXML技術によって成り立っています。前章では、XML技術がそもそもどのような技術なのかについて、解説しました。本章では、それらXML技術がXBRL特にタクソノミでどう応用されているのかを解説します。また、XBRLが柔軟に企業に合わせ、電子財務報告ができている理由は提出者別タクソノミの存在です。提出者別タクソノミがどのように作成されるのかについても解説します。主に裏の仕組みについて取り扱うため、コードを用いて具体的にどういった構造になっているかなど、かなり難しい内容になると思います。前章に引き続きわからないところはあると思いますが、わからなくなったら立ち戻りながら、ゆっくり理解していただくことを推奨します。

本章の内容がわかると、欲しい情報が技術的にどのように定義されているのかなどがわかるようになります。これができると、有価証券報告書に記載されている内容であれば、どんな情報でも取得することができるようになります。

XMLや作成手順など、裏側で行われていることを理解し、XBRLを正しく活用できるようになることを目指します。

4.2 タクソノミの構造

図 4.1: XBRL の構成

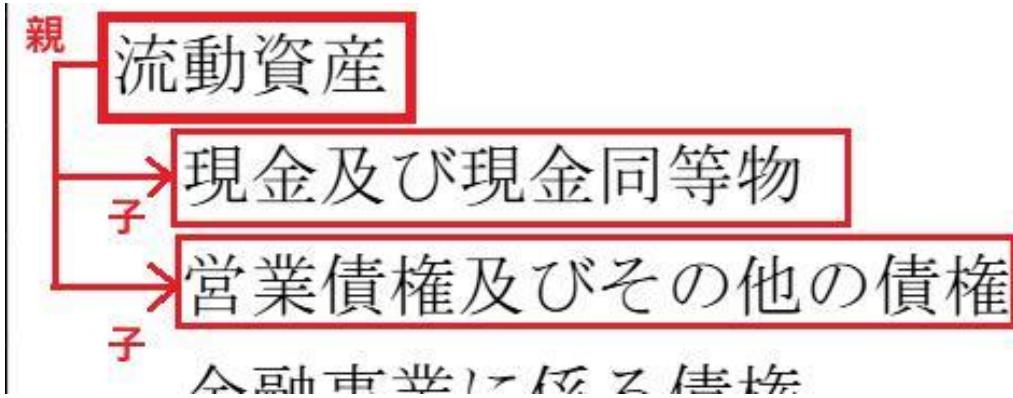


1章で、XBRLはタクソノミとインスタンスにて構成されると解説しました。さらに、各タクソノミはスキーマとリンクベースファイルで構成されます。ひとつのスキーマに対し、複数のリンクベースファイルが参照され、たくさんの情報が定義されることでタクソノミは構成されています。注意点として、スキーマ同士がリンクで結び付くことはありません。これを踏まえると、XBRLはスキーマとリンクベースファイルとインスタンスで構成されると考えることができます。

4.2.1 財務諸表の裏側を知る（仮）

財務諸表を見てみると、ただの表でないことに気づくと思います。これは財務諸表が分類分けされていることを表します。4章で少し触れた、財務諸表の親子関係がこれにあたります。

図4.2: 親子関係の例



たとえば、画像はトヨタ自動車株式会社の2023年度の連結財務諸表の貸借対照表であり、資産の部に関して「流動資産」が親、「現金及び現金同等物」や「営業債権及びその他の債権」が子である親子関係が並んだ構造になっています。このように、連結財務諸表は親子関係などを用いて多次元的な表記をしています。

この親子関係は、いくつものスキーマとリンクベースによって定義されています。

財務諸表の「流動資産」と「現金及び現金同等物」と「営業債権及びその他の債権」の親子関係という3つのみの関係を取っても、3つのファイルを組み合わせて定義されています。なお、データに関しては、上ふたつは実際の企業のXBRLデータをダウンロードしたときに一緒に手に入るタクソノミスキーマ（スキーマファイルなので拡張子が.xsdのもの）と表示リンクベースを見ることで定義されています。表示リンクベースファイルの見分け方として、拡張子が.xmlのもののうち拡張子の直前が「_pre」となっているものが表示リンクベースです。3つ目のEDINETタクソノミの語彙スキーマは、企業ごとにダウンロードするフォルダー内にあるものではなく、EDINETが公表している語彙スキーマです。そのため、XPointerを使用し、参照する必要があります。

図 4.3: EDINET タクソノミのダウンロード

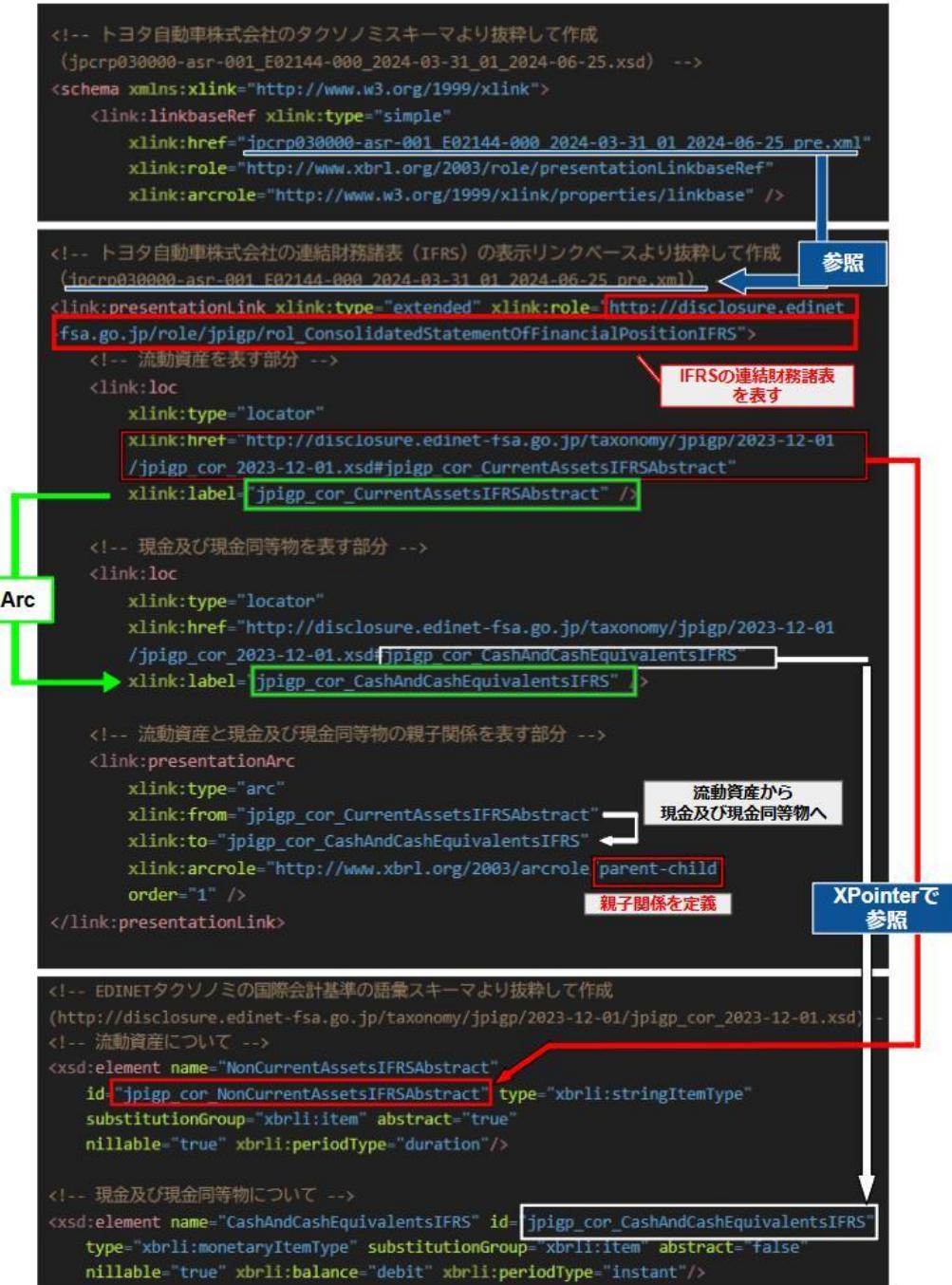
The screenshot shows the EDINET website interface. At the top, there is a logo for EDINET (Electronic Disclosure for Investors Network) and a search bar. Below the search bar, there are tabs for 'トップページ' (Top Page), '書類詳細検索' (Document Detailed Search), '書類全文検索' (Full Text Document Search), and '公告一覧' (Announcement List). A red box highlights the 'EDINETタクソノミ及びコードリスト ダウンロード' (Download of EDINET Taxonomy and Code List) button. On the left, there is a sidebar with a 'ダウンロード' (Download) section containing a link to 'EDINETタクソノミ及びコードリスト'. The main content area shows a list of taxonomy versions:

年次	バージョン	説明	ダウンロード
2025年版	00_会社式一括	2025年3月31日以後に終了する事業年度に係る有価証券報告書等から適用	タクソノミ ダウンロード 399KB
2024年版	01_D.E.I	2024年3月31日以後に終了する事業年度に係る有価証券報告書等から適用	ダウンロード 413KB
2024年版	02_財務消費主義	2023年版EDINETタクソノミ（2023年3月31日以後に終了する事業年度に係る有価証券報告書等から適用）	ダウンロード 459KB
2023年版	03_国際会計基準	2022年版EDINETタクソノミ（2022年3月31日以後に終了する事業年度に係る有価証券報告書等から適用）	ダウンロード

そもそも EDINET タクソノミの語彙スキーマを閲覧するには、EDINET のトップページから「EDINET タクソノミ及びコードリスト ダウンロード」タブの「03.国際会計基準」を開き、最新年（本書は2024年のデータを用いて分析しているため、2024が適切）の EDINET タクソノミデータをダウンロードします。今回は国際会計基準の連結財務諸表の語彙スキーマを見たいので、ダウンロードしたフォルダーを開き、「Taxonomy」→「jpigp」（国際会計基準タクソノミのプレフィックス）→「日付」→「jpigp_cor_年月日.xsd」と開くとアクセスできます。

実際の表示リンクベース抜粋したものを使用し解説します。

図4.4: 表示リンクベースと他ファイルの関係



まず、親子関係がどのように定義されているか解説します。企業のタクソノミスキーマから単純リンクのhref属性で提出者別タクソノミの表示リンクベースファイルを参照してつなげることで、

表示リンクベースで拡張リンクを使えるようにします。

表示リンクベースでは、初めにrole属性で連結財務諸表を指定し、その中だけの関係を定義できるようにします。次にXPointerを用いてEDINET タクソノミの語彙スキーマから要素を参照し、loc要素として定義します。最後にloc要素で定義したもの同士をarcで繋ぎます。arcの中ではformが始点でtoが終点となっており、arcrole属性では始点と終点の関係性を定義します。今回であれば、arcrole属性の最後が「parent-child」となっているため、親子関係を定義できます。

ここまでで、流動資産と現金及び現金同等物の親子関係の定義ができました。実際の財務諸表の中では、流動資産が親である項目は、現金及び現金同等物の項目だけではありません。では次に、親がひとつに対して子が複数ある場合についての表示順序についてです。

図4.5: 表示リンクベースの例

```
<!-- トヨタ自動車株式会社の連結財務諸表（IFRS）の表示リンクベースより抜粋して作成
(jpcrp030000-asr-001_E02144-000_2024-03-31_01_2024-06-25_pre.xml) -->
<link:presentationLink xlink:type="extended" xlink:role="http://disclosure.edinet
-fsa.go.jp/role/jpigg/rol_ConsolidatedStatementOfFinancialPositionIFRS">
    <!-- 流動資産を表す部分 -->
    <link:loc
        xlink:type="locator"
        xlink:href="http://disclosure.edinet-fsa.go.jp/taxonomy/jpigg/2023-12-01
/jpigg_cor_2023-12-01.xsd#jpigg_cor_CurrentAssetsIFRSAbstract"
        xlink:label="jpigg_cor_CurrentAssetsIFRSAbstract" />

    <!-- 現金及び現金同等物を表す部分 -->
    <link:loc
        xlink:type="locator"
        xlink:href="http://disclosure.edinet-fsa.go.jp/taxonomy/jpigg/2023-12-01
/jpigg_cor_2023-12-01.xsd#jpigg_cor_CashAndCashEquivalentsIFRS"
        xlink:label="jpigg_cor_CashAndCashEquivalentsIFRS" />

    <!-- 営業債権及びその他の債権を表す部分 -->
    <link:loc
        xlink:type="locator"
        xlink:href="http://disclosure.edinet-fsa.go.jp/taxonomy/jpigg/2023-12-01
/jpigg_cor_2023-12-01.xsd#jpigg_cor_TradeAndOtherReceivablesCAIFRS"
        xlink:label="jpigg_cor_TradeAndOtherReceivablesCAIFRS" />

    <!-- 流動資産と現金及び現金同等物の親子関係を表す部分 -->
    <link:presentationArc
        xlink:type="arc"
        xlink:from="jpigg_cor_CurrentAssetsIFRSAbstract"
        xlink:to="jpigg_cor_CashAndCashEquivalentsIFRS"
        xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
        order="1" />

    <!-- 流動資産と営業債権及びその他の債権の親子関係を表す部分 -->
    <link:presentationArc
        xlink:type="arc"
        xlink:from="jpigg_cor_CurrentAssetsIFRSAbstract"
        xlink:to="jpigg_cor_TradeAndOtherReceivablesCAIFRS"
        xlink:arcrole="http://www.xbrl.org/2003/arcrole/parent-child"
        order="2" />
</link:presentationLink>
```

新たに
"営業債権及びその他の債権"
の情報をloc要素で定義

Arcも新たに定義

orderの
数字の
若い順で
子の順番
が決まる

loc要素として表示リンクベースに項目を追加したら、arc要素は関係ごとにarcrole属性を付与するため、同じ親子であってもto属性だけを増やすということではなく、ひとつずつ定義します。そのため、同じ子同士の順序に関してはarcrole属性を付与した後に、orderで定義することができます。

orderは昇順で定義することができます。

このような仕組みで、財務諸表で多次元の情報を持つ表を作成しています。

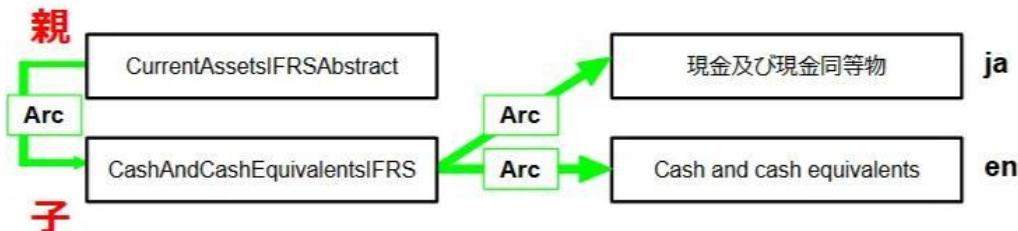
ここまで表示リンクベースを用いて拡張リンクのうち、ファイルの外部の要素をリソースとして参照できるlocator型要素とarc型要素のみを取りました。

次はラベルリンクベースを用いてresource型要素の実例を見て、どのように名称を変えているのか解説します。

EDINETが提供するラベルリンクベースを参照するには、先ほどダウンロードしたEDINET タクソノミデータより、「taxonomy」→「jpigp」（国際会計基準タクソノミのプレフィックス）→「日付」→「label」でラベルリンクベースにアクセスができます。そのうち「jpigp_日付_lab.xml」のものが日本語のラベルリンクベース、「jpigp_日付_lab-en.xml」が英語のラベルリンクベースとなります。こちらのラベルリンクベースファイルを参照して仕組みを解説します。

図4.6: ラベルリンクベースの例

```
<!-- EDINETが提供する財務諸表本表の日本語ラベルリンクベースより抜粋して作成  
(jpigp_2023-12-01_lab.xml) -->  
<link:loc xlink:type="locator"  
xlink:href="../jpigp_cor_2023-12-01.xsd#jpigp_cor_CashAndCashEquivalentsIFRS"  
xlink:label="CashAndCashEquivalentsIFRS"/>  
Arc  
<link:label xlink:type="resource"  
xlink:label="label_CashAndCashEquivalentsIFRS"  
xlink:role="http://www.xbrl.org/2003/role/label"  
xml:lang="ja" id="label_CashAndCashEquivalentsIFRS">現金及び現金同等物</link:label>  
  
language="ja"(日本語)  
を定義  
  
<link:labelArc xlink:type="arc"  
xlink:arcrole="http://www.xbrl.org/2003/arcrole/concept-label"  
xlink:from="CashAndCashEquivalentsIFRS"  
xlink:to="label_CashAndCashEquivalentsIFRS"/>  
  
ラベル付与  
  
<!-- EDINETが提供する財務諸表本表の英語ラベルリンクベースより抜粋して作成  
(jpigp_2023-12-01_lab-en.xml) -->  
<link:loc xlink:type="locator"  
xlink:href="../jpigp_cor_2023-12-01.xsd#jpigp_cor_CashAndCashEquivalentsIFRS"  
xlink:label="CashAndCashEquivalentsIFRS"/>  
Arc  
<link:label xlink:type="resource"  
xlink:label="label_CashAndCashEquivalentsIFRS"  
xlink:role="http://www.xbrl.org/2003/role/label"  
xml:lang="en" id="label_CashAndCashEquivalentsIFRS">Cash and cash equivalents</link:label>  
  
language="en"(英語)  
を定義  
  
<link:labelArc xlink:type="arc"  
xlink:arcrole="http://www.xbrl.org/2003/arcrole/concept-label"  
xlink:from="CashAndCashEquivalentsIFRS"  
xlink:to="label_CashAndCashEquivalentsIFRS"/>  
  
ラベル付与
```



ラベルリンクベースでは、スキーマで定義される要素に対して、名前（ラベル）を定義する役割があります。

日本語・英語どちらのラベルリンクベースでも、label要素はtype="resource"のresource型要素です。resource型要素ではローカルリソースを作成できます。ローカルリソースは、そのファイル内にある情報をリソースとして扱えるため、日本語ラベルリンクベースの場合、「現金及び現金同等物」をローカルリソースとして扱うことができます。Arcでは、XPointerでEDINETタクソノミの語彙スキーマから参照した「現金及び現金同等物」の要素（コンセプト）に対し、arcrole属性でlabel要素で定義したラベルを張り付けるとしています。これを複数言語用意しておくことで、定義してある言語には柔軟に対応できる財務諸表が作成できます。また、言語の区別に関してはlabel要素の中でxml:lang="ja"や"en"で定義しています。

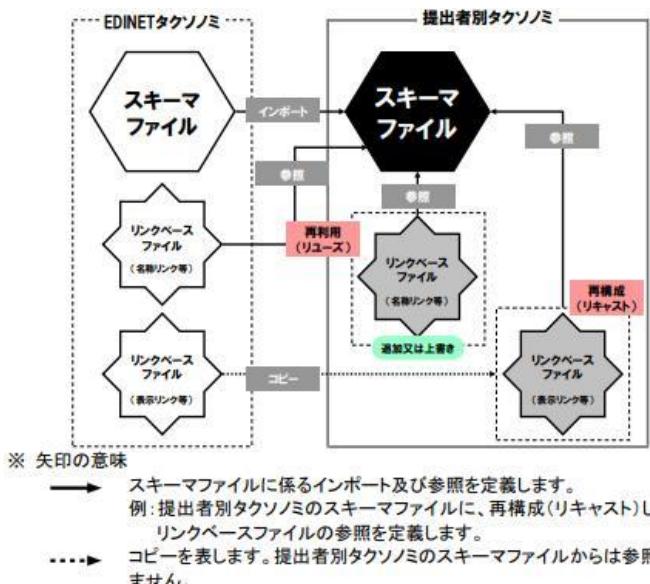
4.3 提出者タクソノミの作成方法

提出者タクソノミもタクソノミである以上、前述した財務諸表の仕組みについては同じです。項目がどのタイミングで追加しているかの違いしかありません。ただし、そもそも準備されていない項目について、どのようにして追加することができているのか知ることは重要です。

企業が各自の業務に合わせ、財務諸表を拡張できることがXBRLが柔軟に電子財務報告できている理由のひとつです。そのため、提出者別タクソノミがどのようにして作成されるのかについて解説します。

EDINETタクソノミは、語彙層と関係層という層に分けて考えることができます。それぞれ、各スキーマとリンクベースを用いて語彙について定義している層と語彙同士の関係について定義している層を指します。ただし、本書では語彙層や関係層で行われることには触れず、もっとシンプルにその仕組みを捉えて解説します。

図4.7: タクソノミの再構成・再利用



提出者別タクソノミ自体は、どの企業の有価証券報告書であっても必ず作成します。ただし、提

出者別タクソノミはEDINET タクソノミを直接修正して作成するわけではありません。EDINET タクソノミで定義されたスキーマファイルをインポートするほか、リンクベースファイルを参照するなどして作成します。また、リンクベースファイルは必要な部分のみコピーし編集して利用することもあります。まずはスキーマファイル、ラベルリンクベースと表示リンクベースの3つを用いて、大まかな作成の流れについて解説します。

提出者別タクソノミは、あくまでEDINET タクソノミにない項目についてタクソノミを作成します。そのため、前提としてEDINET タクソノミにある項目を提出者別タクソノミとして作成しないように注意する必要があります。XBRLでは、これをスキーマファイルのインポートとリンクベース（ラベルリンクベース）の参照によって提出者別タクソノミ内にEDINET タクソノミにある情報を提出者別タクソノミとして定義しないようにしています。

表示リンクベースは参照ではなく、コピーして提出者別タクソノミ内にEDINET タクソノミの情報をすべて写します。コピーはあくまで提出者別タクソノミのスキーマファイルからは参照しません。表示リンクベースが参照ではなくコピーである理由は、EDINET タクソノミの関係と提出者別タクソノミで追加する項目の関係が衝突する可能性を避けるためです。コピーを取り、独自で表示リンクベースを再構築することにより、EDINET タクソノミだけの情報と重複と衝突の両方を避けています。

まとめると、情報を追加するにはラベルリンクベースに追加または上書きをし、新たな情報の関係などを表示リンクベースで再度組み直し、スキーマファイルで参照することで提出者別タクソノミとして項目を作成します。

4.4 まとめ

本章では、XML とタクソノミの関係について、タクソノミ内で XML がどのようにして情報を結び付けているか解説しました。親子関係の定義だけでスキーマとリンクベースの関係性を具体的に説明しましたが、とても複雑であるため、一度での理解は難しいと思います。また、提出者別タクソノミの作成方法についても、何でどのような定義をしているのかということを理解していないと難しいと思います。しかし、これらが理解できると、有価証券報告書をただ見ただけではあまり理解できなかった関係性や、CSV ファイルで探せなかったタクソノミなどがわかるなどの利点があります。有価証券報告書から EDIENT タクソノミで指定されている情報以外のデータなど、細かい情報を取得したいという場合には確実に必要となってくるので、ゆっくりでもよいので理解することを推奨します。

提出者別タクソノミの作成方法について、より詳しく理解したい場合は、EDINET が公表している『提出者別タクソノミ 作成ガイドライン』という資料でより複雑な説明をしているので、そちらからの学習を推奨します。

第5章 分析環境の構築

5.1 本章の目的

本章ではプログラムを使用するために、「プログラミングで利用する EDINET API の準備」と「プログラムを動作させるためのソフトウェアの環境構築」について説明します。

もうすでに EDINET API の利用方法をご存じの方・Python が利用可能な状態にある方は、本章を読み飛ばしていただいて結構です。

5.2 EDINET API を使えるようにする

まず、有報をダウンロードするために必須の API についてです。API (Application Programming Interface) とはソフトウェアやプログラム、Web サービス間をつなぐインターフェースです。その中でも EDINET API は、使用者が直接 EDINET のウェブページに行くのではなく、プログラムを介して EDINET のデータベースから効率的にデータを取得できる API です。EDINET API により、EDINET 利用者は効率的に開示情報を取得することが可能となります。

そんな EDINET API は以下の 2 種類です。

- ・書類一覧 API
- ・書類取得 API

書類一覧 API は、提出された書類を把握するための API です。こちらは EDINET に提出された書類の一覧を取得する API となっています。書類一覧 API を用いることで、日付ごとに分類された提出書類の基本情報を取得できます。

書類取得 API は、提出された書類を取得するための API です。この API を活用すると、取得書類の種類の指定、ファイル名を企業名にしてダウンロード、フィルタリングした企業の書類だけなどの情報を取得できます。

EDINET API の使用には、アカウントの作成と API キーの発行が必要となります。

5.2.1 注意点

アカウント作成、API キーの発行にはいくつか注意点があります。この注意点の項目ができていないと、アカウント作成や API キー発行がうまくいかないこともあるため、注意が必要です。

公式の EDINET API 仕様書では、Microsoft Edge での利用方法が詳しく記載されていましたが、Google Chrome での設定方法の記載がなく利用者も多いと思うので、今回は Chrome の説明をします。

Microsoft Edge ご利用の方は、以下の URL からダウンロードできる EDINET_API 仕様書を参考にしてください。

▼ EDINET 操作ガイド EDINET API 仕様書(Version2)

<https://disclosure2.dledinet-fsa.go.jp/guide/static/disclosure/WZEK0110.html>

5.2.1.1 ポップアップの設定

図5.1: chromeの設定を開く



Google Chromeを開き、右上の三点リーダーから「設定」を開きます。

図5.2: サイトの設定を開く



画面の左のサイドバーから「プライバシーとセキュリティ」→「サイトの設定」の順で開きます。

図5.3: ポップアップとリダイレクトを開く



次に「ポップアップとリダイレクト」を開きます。

図5.4: ポップアップサイトの追加



動作のカスタマイズで「ポップアップの送信やリダイレクトの使用を許可するサイト」を開きます。

図5.5: EDINET API のサイト URL の追加



「サイトの追加」画面でEDINET APIのサイトリンク「<https://api.edinet-fsa.go.jp>」を入力します。
これでポップアップの設定は完了です。

5.2.1.2 JavaScriptの設定

図5.6: プライバシーとセキュリティの設定を開く



再度、Chromeの設定から「プライバシーとセキュリティ」を開きます。

図5.7: v8オプティマイザーを許可



次に「V8のセキュリティを管理する」を開きます。

「サイトでのV8オプティマイザーの使用を許可する」にチェックを入れてください。

以上で、JavaScriptの設定は完了です。

5.2.1.3 Cookie の設定

図 5.8: サードパーティ Cookie の設定を開く



再度、設定画面のサイドバーから「プライバシーとセキュリティの設定」→「サードパーティCookie」の順で開きます。

図 5.9: サードパーティの Cookie を許可する

サイトがブラウジング中のユーザーのトラッキングに使用できる情報の種類を管理します。

サードパーティの Cookie を許可する

サイトは、Cookie を使用して閲覧の利便性を高めることができます（ログイン状態の維持、ショッピングカートの中身の保存など）

サイトは、Cookie を使用して別のサイトでのあなたの閲覧アクティビティを確認できます（広告のパーソナライズなどが行えます）

シークレットモードでサードパーティ Cookie をブロックする

サードパーティの Cookie をブロックする

詳細設定

閲覧トラフィックと一緒に「Do Not Track」リクエストを送信する

すべてのサイトデータと権限を表示

サードパーティの Cookie の使用が許可されたサイト

「サードパーティ Cookie を許可する」にチェックを入れます。

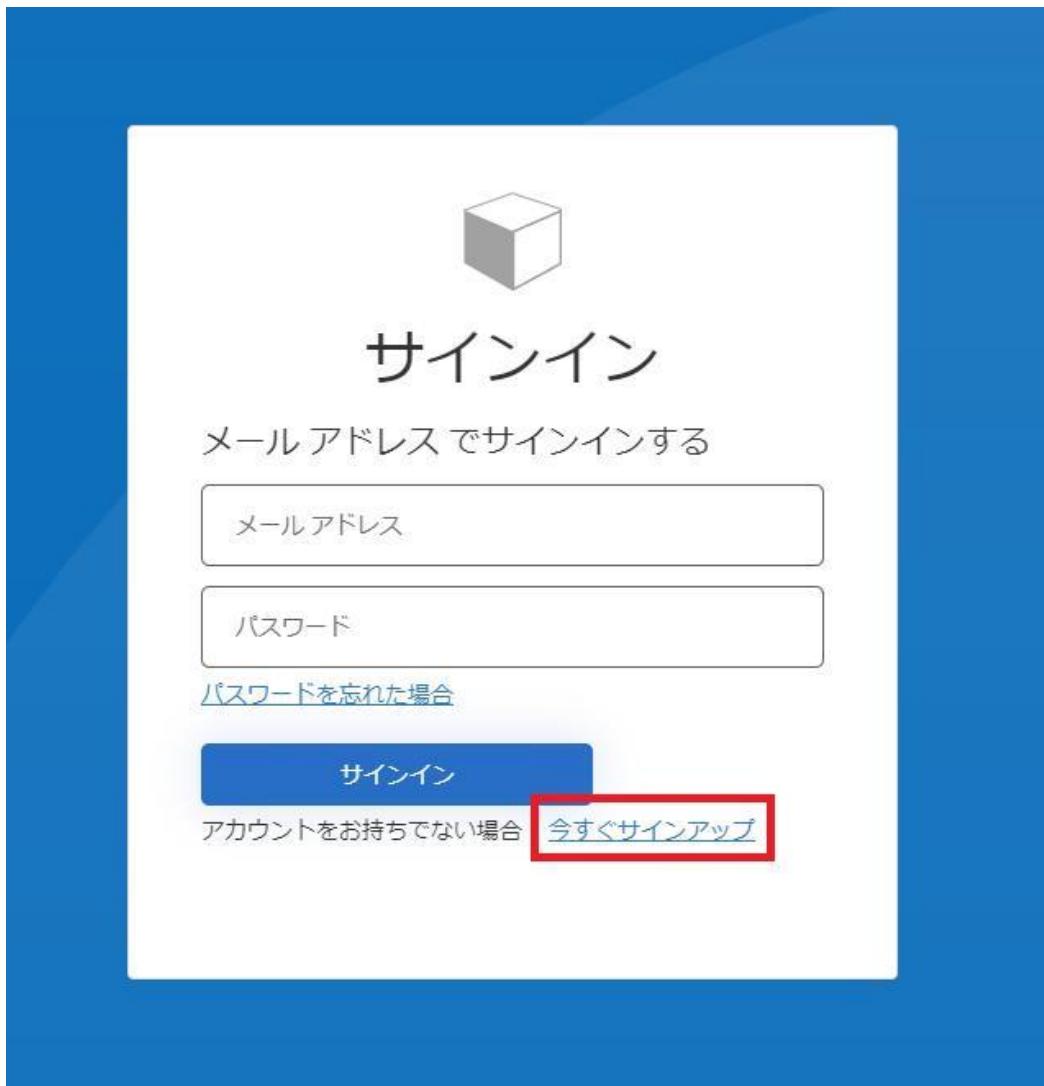
以上で、Cookie の設定及びChrome の設定は完了となります。

5.2.2 アカウント作成

アカウントを作成する場合には、以下のURLから、サインイン画面を表示させます。

<https://api.edinet-fsa.go.jp/api/auth/index.aspx?mode=1>

図5.10: サインイン画面



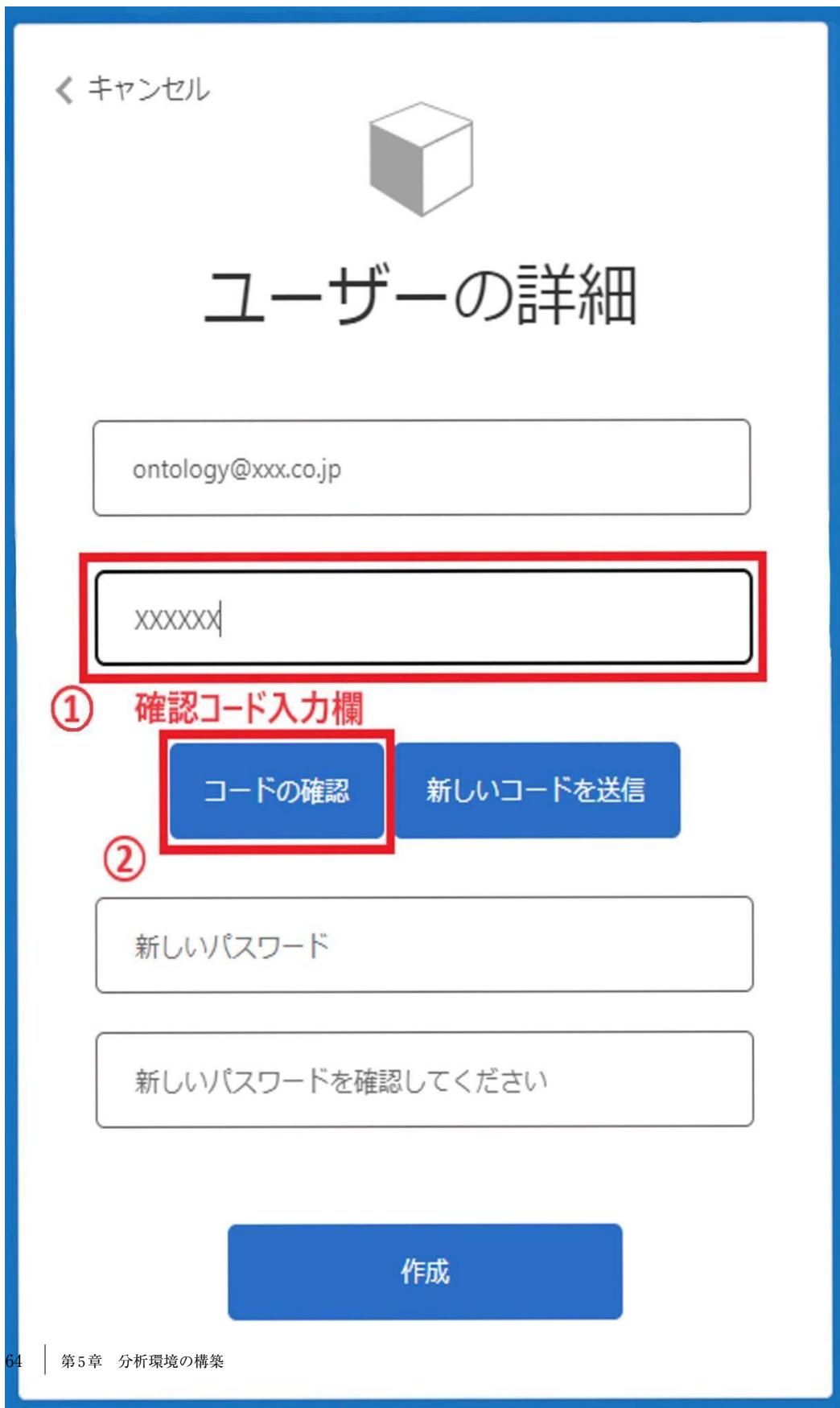
下の「今すぐサインアップ」からアカウントを作成します。

図5.11: メールアドレスを入力し確認コードを受信する



サインアップ画面の上のフォームにメールアドレスを入力し、「確認コードを送信」をクリックします。

図5.12: 確認コードを入力



確認コードを入力し、「コードの確認」へ進みます。

図 5.13: ユーザーパスワードを入力



コードが正しく受理されると、確認コード入力欄がなくなります。次に、下段のパスワードを設定していきます。「新しいパスワード」と「新しいパスワードを確認してください」のフォームには、同一のパスワードを設定します。

図 5.14: 多要素認証を行う

← キャンセル

多要素認証

認証用に SMS でコードを受け取る、または電話を受けることを希望する番号を以下に入力してください。

国番号

① Country/Region

② 電話番号

③ コードの送信

③' 電話する

パスワードが受理されると、次に多要素認証に移ります。「国コード」と「電話番号」をそれぞれ入力したのち、以下のいずれかの多要素認証を行います。

1. SMSによる本人確認 SMSにて通知される確認コードを多要素確認コード入力画面に入力して認証を行う。
2. 自動音声による本人確認 登録した電話番号に発信される自動音声通話を受け取り、音声ガイドに従い、テンキーで「#」を入力することにより認証を行う。
3. SMSか電話でコードを受け取る。

図 5.15: 多要素認証確認コードを入力



SMSか電話にて受け取った確認コードを入力すると、APIキー発行画面が表示されアカウント作成が完了となります。

図 5.16: API キー発行画面

APIキー発行画面(API key issuance screen)

連絡先 (Contact)	所属(Company)	①
	氏名(Full name)*	
	電話番号(Phone)*	

*は必須項目です。 (Required)

②

多要素認証クリア(MFA Clear) **連絡先登録(Save)**

氏名と電話番号は必須
所属は任氏で入力し、
Saveしてください

続けて API キーの発行についてです。多要素認証でのサインイン後に、API キー発行画面がポップアップ画面で表示されます。

連絡先を入力し、「連絡先登録 (Save)」をクリックします。

※2回目以降は、以下の画像のように「連絡先変更 (Save)」となります。

図 5.17: 連絡先保存確認画面

api.edinet-fsa.go.jp の内容

EE0092C : 連絡先を保存します。よろしいですか？(Save?)



図 5.18: 連絡先保存完了画面

api.edinet-fsa.go.jp の内容

EE0102I : 連絡先を保存しました。(Saved.)



確認画面がふたつ表示されるので、どちらも「OK」を押します。

図 5.19: API キーを取得

EDINET - Google Chrome
api.edinet-fsa.go.jp/WEEE0090.aspx

A P I キー発行画面(API key issuance screen)

連絡先 (Contact)	所属(Company) 氏名(Full name)* 電話番号(Phone)*	
------------------	---	--

*は必須項目です。 (Required)

多要素認証クリア(MFA Clear) 離線先変更(Save)

※「連絡先変更」をクリックしても A P I キーは変更されません。
(Clicking Save does not change the API key.)

A P I キー
(API key) XXXXXXXXXXXXXXXXXXXXXXX
A P I キー再発行(Reissue)

※ A P I キーは「A P I キー再発行」ボタンクリック時のみ再発行されます。
(The API key is reissued only when you click the Reissue button.)

API 発行画面に API キーが表示されます。この API キーはコードの中で使用するため、メモしておいてください。

5.3 プログラムを扱う準備をする

5.3.1 VSCode と Python を使えるようにする

コードエディターは様々なもののが存在しますが、本書では広く使われている「VSCode」を用いて、プログラミング言語は「Python」を使用します。インストーラーをダウンロードし、セットアップを行いましょう。なお、本書では Windows でのセットアップを想定しています。

5.3.2 インストーラーをダウンロード

5.3.2.1 VSCode

以下の URL から「Download for Windows」をクリックしてください。

<https://code.visualstudio.com/>

「VSCodeUserSetup-x64-1.94.0.exe」というファイルがダウンロードできれば、成功です。

5.3.2.2 Python

以下のサイトから「Python 3.12.3」の「Python installer(64-bit)」をダウンロードしてください。筆者が現時点でのプログラム動作確認済みのバージョンです。以下のリンクを踏むことで、直接ダウンロードが行われます。

<https://www.python.org/ftp/python/3.12.3/python-3.12.3-amd64.exe>

その後はそれぞれファイルを開き、画面の指示に従いインストールを行ってください。

インストールが終了した後に以下を行うことを推奨します。

※行わなくても問題なく使用できます。

- ・VSCode 内でエディターの日本語化を行う拡張機能を追加

- ・Python における仮想環境の構築

また、本書の趣旨から外れるため、詳細な説明は省略させていただきます。詳しくは他の参考文献をご参照ください。

5.4 まとめ

以上で、プログラミングをするための事前準備が完了しました。プログラミングを利用することで、より効率的に分析を行うことが可能になります。

次の章では、データを分析するための大量のデータを収集してみます。大量の有価証券報告書を自動で取得できるようになります。

第6章 大量の有価証券報告書を自動でダウンロードする

6.1 本章の目的

本章では、EDINET API を用いて多くの有価証券報告書のXBRLファイルを集める方法について解説します。

5章までで、XBRLと有価証券報告書の知識について解説を行いました。企業の比較分析などを行うためには、多くの元データが必要となります。この問題に対しては、前章で取り扱ったEDINET APIを用いたプログラムでデータ取得すると有用でしょう。コードについてはGitHub上に掲載されておりますが、可能であれば自ら手を動かしてコードを書き学習することを推奨します。

6.2 取得するコードを書く

利用者の環境に合わせて変更が必要な部分があるため、まずはそこについて解説します。

6.2.0.1 (1) 日付の指定

リスト6.1: 日付を指定する

```
1: """
2: 日付の指定
3: """
4: start_date = datetime.date(2024, 5, 1) # 開始日付 随時変更
5: end_date = datetime.date(2024, 5, 31) # 終了日付 随時変更
6: day_list = make_day_list(start_date, end_date)
```

main()の中で日付のリストを作成します。そのリストに入る値をstart_dateとend_dateで取得期間を指定します。

あまりに長いとサーバーに負担をかけてしまい、時間がかかるだけでなくエラーになってしまふ場合もあるため、1ヵ月単位などで取ってくることを推奨します。また、6月は前年度の有報の提出期限であるため提出数が特に多く、細かく指定するなどの対処を推奨します。

6.2.0.2 (2) 保存先の指定

リスト6.2: 保存先を指定する

```
1: """
2: 保存先の指定
3: """
4: def download_xbrl_in_zip(securities_report_doc_list, number_of_lists):
5:     # ダウンロードする有報を保存しておく場所を指定。もしなければフォルダーを作成する。
```

```
6:     save_dir = "/path/to/download/directory/" # あなたの保存先のパスに変更してください。
```

download_xbrl_in_zip()のsave_dirに保存先のパスを指定してください。

6.2.0.3 (3)API キーの設置

リスト 6.3: make_doc_id_list() に API キーを設定

```
1: """
2: DocID（有価証券報告書の番号）を取得するときに使うAPIキーの設置
3: """
4: def make_doc_id_list(day_list):
5:     securities_report_doc_list = []
6:     for index, day in enumerate(day_list):
7:         url = "https://disclosure.edinet-fsa.go.jp/api/v2/documents.json"
8:         params = {"date": day.strftime("%Y-%m-%d"),
9:                   "type": 2,
10:                  "Subscription-Key": "your_subscription_key"
11:                  # Subscription-KeyはあなたのAPIキーを使用
12:                 }
```

リスト 6.4: download_xbrl_in_zip() に API キーを設定

```
1: """
2: ダウンロードするときに使うAPIキーの設置
3: """
4: for index, doc_id in enumerate(securities_report_doc_list):
5:     print(doc_id, ":", index + 1, "/", number_of_lists)
6:     url = f"https://disclosure.edinet-fsa.go.jp/api/v2/documents/{doc_id}"
7:     params = {"type": 1,
8:               "Subscription-Key": "your_subscription_key"
9:               # Subscription-KeyはあなたのAPIキーを使用
10:              }
```

make_doc_id_list() と download_xbrl_in_zip() のふたつの関数の中の params という変数の中で前章で取得した API キー（Subscription-Key）を指定します。

コードの your_subscription_key の部分をご自身の API キーに変更します。なお「””」は消さずに中に書きます。

6.2.0.4 関数の解説

コードの中で使用する関数について解説します。

6.2.0.5 書類を取得する関数

リスト 6.5: 書類を取得する関数

```
1: """
2: DocID（有価証券報告書の番号）を取得する関数
3: """
4: def make_doc_id_list(day_list):
5:     securities_report_doc_list = []
6:     for index, day in enumerate(day_list):
7:         url = "https://disclosure.edinet-fsa.go.jp/api/v2/documents.json"
8:         params = {"date": day.strftime("%Y-%m-%d"), "type": 2,
9: "Subscription-Key":"your_subscription_key"}
10:        res = requests.get(url, params=params)
11:        json_data = res.json()
12:
13:        if "results" in json_data:
14:            for num in range(len(json_data["results"])):
15:                ordinance_code = json_data["results"][num]["ordinanceCode"]
16:                form_code = json_data["results"][num]["formCode"]
17:                docInfoEditStatus = json_data["results"][num][num]["docInfoEditStatus"]
18:
19:                if ordinance_code == "010" and form_code == "030000" and
20: docInfoEditStatus != 2:
21:                    print(json_data["results"][num]["filerName"],
22: json_data["results"][num]["docDescription"],
23:                     json_data["results"][num]["docID"])
24:                    securities_report_doc_list.append(json_data["results"][num]["docID"])
25:
26: return securities_report_doc_list
```

make_doc_id_list() では、各日付について EDINET API を呼び出し、有価証券報告書の固有番号である DocID を収集します。EDINET API を呼び出す際に API キーが必要となるため、取得してもらいました。params の “type” は取得情報の指定をしており、2 は提出一覧およびメタデータの取得を指します。これにより、後続の処理で有価証券報告書を特定できます。また、途中で指定している ordinance_code と form_code はそれぞれ府令コードと様式コードを指すものであり、今回は有価証券報告書を取得したいため、以下のように指定をしております。

- ordinance_code=010
- form_code=30000

docInfoEditStatus は書類情報修正区分を示すものであり、財務局員が書類を修正した場合に、修正前の書類にはこの書類情報修正区分の数値に 2 が割り当てられます。最新の書類を取得したいため、docInfoEditStatus != 2 で修正前の書類を取得せずに処理を行うようにしています。該当した提出書類の DocID（書類管理番号）を取得し、securities_report_doc_list（有報のリスト）に格納します。

6.2.0.6 ZIP 形式でダウンロードする関数

リスト6.6: ZIP形式でダウンロードする関数

```
1: """
2: ZIP形式でダウンロードする関数
3: """
4: def download_xbrl_in_zip(securities_report_doc_list, number_of_lists):
5:     # ダウンロードする有価証券報告書を保存しておく場所を指定。もしなければフォルダーを作成。
6:
7:     save_dir = "/path/to/download/directory"# あなたの保存先のパスに変更。
8:     if not os.path.exists(save_dir):
9:         os.makedirs(save_dir)
10:
11:    for index, doc_id in enumerate(securities_report_doc_list):
12:        url = f"https://disclosure.edinet-fsa.go.jp/api/v2/documents/{doc_id}"
13:        params = {"type": 1, "Subscription-Key":"your_subscription_key"}
14:        filename = os.path.join(save_dir, f"{doc_id}.zip")
15:        res = requests.get(url, params=params, stream=True)
16:
17:        try :
18:            if res.status_code == 200:
19:                with open(filename, 'wb') as file:
20:                    for chunk in res.iter_content(chunk_size=1024):
21:                        file.write(chunk)
22:                print(f"Downloaded and Saved: {filename}")
23:            except Exception as e:
24:                print(f"Failed to download file {doc_id}, status code: {e}")
```

`download_xbrl_in_zip()`では、各ドキュメントIDについてAPIを呼び出し、XBRLファイルをZIP形式で指定された場所にダウンロードします。paramsの”type”は取得情報の指定をしており、1はXBRLデータの取得を指します。つまり、提出された書類の本文と監査報告書を取得することができます。resのステータスコードが200はリクエスト成功を意味し、成功したものののみXBRLファイルをダウンロードしています。このダウンロード時に各ファイルはZIP形式で保存され、ファイル名はドキュメントIDになります。

6.3 ソースコード

リスト6.7: ソースコード

```
1: import requests
2: import datetime
3: import os
4:
```

```

5: """
6: 日付を取得する関数
7: """
8: def make_day_list(start_date, end_date):
9:     print("start_date:", start_date)
10:    print("end_date:", end_date)
11:
12:    period = end_date - start_date
13:    period = int(period.days)
14:    day_list = []
15:    for d in range(period + 1):
16:        day = start_date + datetime.timedelta(days=d)
17:        day_list.append(day)
18:
19:    return day_list
20:
21: """
22: DocID（有価証券報告書の番号）を取得する関数
23: """
24: def make_doc_id_list(day_list):
25:     securities_report_doc_list = []
26:     for index, day in enumerate(day_list):
27:         url = "https://disclosure.edinet-fsa.go.jp/api/v2/documents.json"
28:         params = {"date": day.strftime("%Y-%m-%d"),
29:                   "type": 2,
30:                   "Subscription-Key": "your_subscription_key"
31:                   # Subscription-KeyはあなたのAPIキーを使用
32:                   }
33:
34:         res = requests.get(url, params=params)
35:         json_data = res.json()
36:         print(day)
37:
38:         if "results" in json_data:
39:             for num in range(len(json_data["results"])):
40:                 ordinance_code = json_data["results"][num]["ordinanceCode"]
41:                 form_code = json_data["results"][num]["formCode"]
42:                 docInfoEditStatus = json_data["results"][num]["docInfoEditStatus"]
43:
44:                 if ordinance_code == "010" and form_code == "030000" and
docInfoEditStatus != 2:

```

```

45:             print(json_data["results"][num]["filerName"],
json_data["results"][num]["docDescription"],
46:                     json_data["results"][num]["docID"])
47:                     securities_report_doc_list.append(json_data["results"][num]["docID"]))
48:
49:     return securities_report_doc_list
50:
51: """
52: ZIP形式でダウンロードする関数
53: """
54: def download_xbrl_in_zip(securities_report_doc_list, number_of_lists):
55:     # ダウンロードする有価証券報告書を保存しておく場所を指定。もしなければフォルダーを作成。
56:
57:     save_dir = "/path/to/download/directory/" # あなたの保存先のパスに変更。
58:     if not os.path.exists(save_dir):
59:         os.makedirs(save_dir)
60:
61:     for index, doc_id in enumerate(securities_report_doc_list):
62:         print(doc_id, ":", index + 1, "/", number_of_lists)
63:         url = f"https://disclosure.edinet-fsa.go.jp/api/v2/documents/{doc_id}"
64:         params = {"type": 1,
65:                   "Subscription-Key": "your_subscription_key"
66:                   # Subscription-KeyはあなたのAPIキーを使用
67:                   }
68:         filename = os.path.join(save_dir, f"{doc_id}.zip")
69:         res = requests.get(url, params=params, stream=True)
70:
71:         try :
72:             if res.status_code == 200:
73:                 with open(filename, 'wb') as file:
74:                     for chunk in res.iter_content(chunk_size=1024):
75:                         file.write(chunk)
76:                         print(f"Downloaded and Saved: {filename}")
77:             except Exception as e:
78:                 print(f"Failed to download file {doc_id}, status code: {e}")
79:
80: 日付の指定をしたうえで上記の関数をまとめて起動するメインの関数
81: """
82: def main():
83:     # 集める期間

```

```
84:     start_date = datetime.date(2024, 5, 1) # 開始日付
85:     end_date = datetime.date(2024, 5, 31) # 終了日付
86:     day_list = make_day_list(start_date, end_date)
87:
88:     securities_report_doc_list = make_doc_id_list(day_list)
89:     number_of_lists = len(securities_report_doc_list)
90:     print("number_of_lists: ", number_of_lists)
91:     print("get_list: ", securities_report_doc_list)
92:
93:     download_xbrl_in_zip(securities_report_doc_list, number_of_lists)
94:     print("download finish")
95:
96:
97: if __name__ == "__main__":
98:     main()
```

6.3.1 実行結果

以下のログのように、2024/5/1~2024/5/31までの一か月間に提出された有報をダウンロードした結果、筆者の場合、今回は226件の有報が15分足らずで取得できました。

もっと多くの件数をダウンロードする場合は、サーバーに負荷をかけるためエラーを返されることがあります。一度に取得する期間を絞るか、コード内のdownload_xbrl_in_zip()の中にsleep()などを用いてクールタイムを設けるようにしてみるとよいでしょう。

6.3.2 ターミナル上でのログの一部

リスト6.8: ターミナル上でのログの一部

```
1: start_date: 2024-05-01
2: end_date: 2024-05-31
3: 2024-05-01
4: 2024-05-02
5: 2024-05-03
6: 2024-05-04
7: 2024-05-05
8: 2024-05-06
9: 2024-05-07
10: 2024-05-08
11: 2024-05-09
12: 2024-05-10
13: 2024-05-11
```

```
14: 2024-05-12
15: 2024-05-13
16: 株式会社あさひ 有価証券報告書－第49期(2023/02/21－2024/02/20) S100TCYC
17: 2024-05-14
18: 2024-05-15
19: 株式会社西松屋チェーン 有価証券報告書－第68期(2023/02/21－2024/02/20) S100TF9K
20: 株式会社オークワ 有価証券報告書－第55期(2023/02/21－2024/02/20) S100TEV9
21: 2024-05-16
22: 株式会社 セキチュー 有価証券報告書－第73期(2023/02/21－2024/02/20) S100TG3H
23: 2024-05-17
24: パレモ・ホールディングス株式会社 有価証券報告書－第39期(2023/02/21－2024/02/20)
S100TG6Z
25: 株式会社フジ 有価証券報告書－第57期(2023/03/01－2024/02/29) S100TGC6
26: 株式会社 平和堂 有価証券報告書－第67期(2023/02/21－2024/02/20) S100TGDM
27: 株式会社瑞光 有価証券報告書－第61期(2023/02/21－2024/02/20) S100TGHZ
28: 2024-05-18
29: 2024-05-19
30: 2024-05-20
31: 株式会社サンデー 有価証券報告書－第50期(2023/03/01－2024/02/29) S100TG0R
32: 株式会社しまむら 有価証券報告書－第71期(2023/02/21－2024/02/20) S100TGLZ
33:
34: .
35: .
36: (省略)
37: .
38: .
39: S100TJL9 : 226 / 226
40: Downloaded and Saved: /path/to/download/directory/S100TJL9.zip
41: Downloaded and Saved: /path/to/download/directory/S100TJL9.zip
42: download finish
43: download finish
44:
```

ディレクトリーでは、以下のように書類ごと保存されています。

図 6.1: ディレクトリー画像

名前	更新日時	種類
S100T8C7	2024/07/19 13:26	ファイル フォルダー
S100T8C7.zip	2024/07/19 10:03	圧縮 (zip 形式) フォルダー
S100TCYC.zip	2024/07/19 10:00	圧縮 (zip 形式) フォルダー
S100TEV9.zip	2024/07/19 10:00	圧縮 (zip 形式) フォルダー
S100TF9K.zip	2024/07/19 10:00	圧縮 (zip 形式) フォルダー
S100TFVA.zip	2024/07/19 10:04	圧縮 (zip 形式) フォルダー
S100TFVY.zip	2024/07/19 10:05	圧縮 (zip 形式) フォルダー
S100TG3H.zip	2024/07/19 10:00	圧縮 (zip 形式) フォルダー
S100TG6G.zip	2024/07/19 10:04	圧縮 (zip 形式) フォルダー
S100TG6Z.zip	2024/07/19 10:00	圧縮 (zip 形式) フォルダー
S100TGC6.zip	2024/07/19 10:00	圧縮 (zip 形式) フォルダー
S100TGDM.zip	2024/07/19 10:00	圧縮 (zip 形式) フォルダー
S100TGHZ.zip	2024/07/19 10:00	圧縮 (zip 形式) フォルダー
S100TGLA.zip	2024/07/19 10:06	圧縮 (zip 形式) フォルダー
S100TGLZ.zip	2024/07/19 10:01	圧縮 (zip 形式) フォルダー
S100TGMY.zip	2024/07/19 10:01	圧縮 (zip 形式) フォルダー
S100TGO1.zip	2024/07/19 10:01	圧縮 (zip 形式) フォルダー
S100TGOR.zip	2024/07/19 10:00	圧縮 (zip 形式) フォルダー
S100TGSN.zip	2024/07/19 10:01	圧縮 (zip 形式) フォルダー
S100TGUF.zip	2024/07/19 10:01	圧縮 (zip 形式) フォルダー
S100TGV5.zip	2024/07/19 10:01	圧縮 (zip 形式) フォルダー

実際に分析する際は、この zip ファイルを解凍し使用します。

ダウンロードしたファイルが有報のドキュメント ID でラベリングされています。しかし、これではファイルをいちいち開かないと、どの企業のものかわかりません。そのような場合は、書類一覧 API で企業名を取得しファイルを保存する際、企業名で保存できるようプログラムを変更するのも有効でしょう。

6.4 コラム：有価証券報告書以外の書類の集め方

本書では、有価証券報告書をまとめてダウンロードしましたが、有価証券報告書以外の書類を自動でダウンロードすることもできます。その際には、`ordinance_code`, `form_code` を変更します。

有価証券報告書を集めたい場合には、`ordinance_code == 010`, `form_code == 030000` を指定しました。

以下の表であるように、ほかの一般的な EDINET の書類は以下のような組み合わせで取得できます。

表 6.1: 書類種別

書類名	ordinance_code	form_code
四半期報告書	010	043000
半期報告書	010	050000
訂正有価証券報告書	010	030001

`ordinance_code` は内閣府令を意味し、`form_code` は内閣府令の何号様式かを意味します。

EDINET API仕様書(version2)を参照して、内閣府令に対応したordinance_codeを選択します。また、タクソノミ要素リストを参照して、内閣府令の何号様式化に対応したform_codeを選択します。

以上を適切に施託することで、有価証券報告書以外の書類もダウンロードすることができます。

引用：EDINET API仕様書（version2）p85

図6.2: 府令コード

引用：<https://www.fsa.go.jp/search/20231211.html>よりタクソノミ要素リスト.xlsx

図6.3: 様式コード

書類が異なれば記述される内容も異なります。同じようにタクソノミも異なるため、注意が必要です。

6.5 まとめ

本章では、一度に大量のデータをダウンロードすることについて解説しました。毎回EDINETからダウンロードして中身を確認する手間が省けるため、プログラムを回す方が圧倒的に楽だと思います。もし興味があれば、このコードを使いややすく改良してみてください。コードはGitHubにも置いてあります。

さて次章からこのデータを使用して、ついにデータ分析をしていこうと思います。

第7章 連結財務諸表から営業利益を自動で取得する

7.1 本章の目的

本章から実際に、有価証券報告書の中からデータの取得をしていきます。今回は、売上高・費用、資産・負債といった財務データをXBRLファイルから取得していきます。財務データを分析することで、投資意思決定に役立たせることができます。分析するために必要なデータとして、本章では営業利益の数値を例に取得します。

7.2 1社から取得

財務データを取得する際には、以下の項目が統一されている必要があります。

- ・会計基準
- ・連結財務諸表か個別財務諸表か
- ・勘定科目

上記の3項目が統一されていないと、同じコードでデータを一括取得することは困難になります。統一されていない場合、それらの対応のためにコードの条件分岐を実施したりするなど、XBRLを利用する際の苦労はここに集約されます。

本章はトヨタ自動車株式会社を例に、データの取得をします。基本的にグループ会社の場合、グループ全体を示す連結財務諸表を用いて分析することが一般的です。トヨタ自動車株式会社の連結財務諸表は、国際会計基準(IFRS)で作成されています。したがって、本章では国際会計基準に則って作成された連結財務諸表の営業利益を取得するためのコードを書きます。

一度に何社も取得するとコードのどの部分でデータを取得しているのかが理解しづらいため、まずはトヨタ自動車1社から営業利益のデータを取得し、コードの理解に努めます。

7.2.1 タクソノミの確認

データを取得するには、タクソノミの指定が必要です。財務データの取得には「タクソノミのタグ」と「コンテキストID(期間などを表す指標)」のふたつを指定します。タクソノミは、3章で少し触れた要素リストにて確認できます。本章では国際会計基準の連結財務諸表を対象にするため、国際会計基準タクソノミ要素リストを参照し、営業利益のタクソノミを確認します。

まずは、国際会計基準タクソノミ要素リストへのアクセス方法です。以下のリンクよりEDINETの「各種情報検索サービス」のページにアクセスしてください。

<https://www.fsa.go.jp/search/index.html>

図7.1: EDINET各種情報検索サービス

The screenshot shows the homepage of the EDINET Information Search Service. At the top, there is a blue header bar with the text "各種情報検索サービス". Below the header, there is a sidebar on the right containing several sections with icons and text:

- 各種窓口のご案内
- 金融サービス利用者相談室
- 金融行政モニター
- 情報公開等
- パブリックコメント
- 申請・届出・照会
- 入札公告等
- 採用情報

The main content area has a section titled "EDINET(電子開示)" with the following sub-sections:

- EDINETについて
 - 【閲覧サイト】
 - 【提出サイト】
- EDINETに関するお問い合わせ
- EDINETの稼働状況について
- EDINETタクソノミの知的所有権について

Below this is a "New Information Disclosure Service" section with links to:

- 金融庁ソーシャルメディアアカウント
- 関連リンク
- JSEA 証券取引等監視委員会
- CPAOB 公認会計士・監査審査会

A red box highlights the "公表済のEDINETタクソノミ" section, which contains a list of documents:

- 2025年版EDINETタクソノミ [2024年(令和6年)11月12日]
- 2024年版EDINETタクソノミ [2023年(令和5年)12月11日]
- 2023年版EDINETタクソノミ [2022年(令和4年)11月8日]
- 2022年版EDINETタクソノミ [2021年(令和3年)11月9日]
- 2021年版EDINETタクソノミ [2020年(令和2年)11月10日]

At the bottom of this section, it says "これ以前のものも含むEDINETタクソノミの一覧は、こちら。"

At the very bottom of the page, there is a small note: "EDINET小委員会に賛同する会員企業実績付き" and a footer note: "最新版のEDINETタクソノミのページを開きます。" followed by "ページ中央までスクロールし、「3. 公表資料」から「国際会計基準タクソノミ」のファイルをクリックしてダウンロードします。"

図 7.2: 國際会計基準タクソノミから「営業利益」を検索

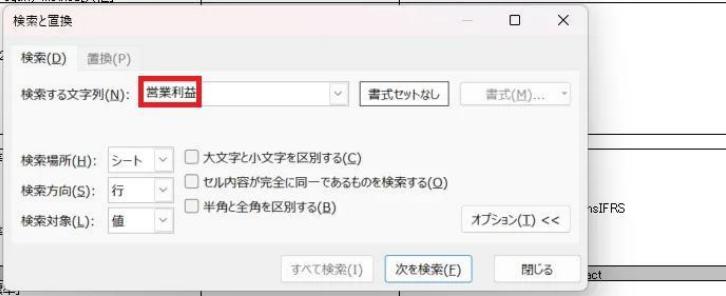
	標準ラベル(日本語)	冗長ラベル(日本語)		
187	損益計算書	損益計算書(IFRS)【タイトル項目】	Statement	
188	損益計算書	損益計算書(IFRS)【表】	Statement	
189	損益計算書	損益計算書(IFRS)【表示項目】	Statement	
190	維持事業	維持事業(IFRS)【タイトル項目】	Continuing	
191	売上収益	売上収益(IFRS)	Revenue	
192	売上高	売上高(IFRS)	Net sales	
193	収益	収益(IFRS)	Revenue	
194	売上原価	売上原価(IFRS)	Cost of sales	
195	売上総利益	売上総利益(IFRS)	Gross profit	
196	販売収益及び一般管理費	販売収益及び一般管理費(IFRS)	Selling expenses	
197	販売費	販売費(IFRS)	Selling expenses	
198	一般管理費	一般管理費(IFRS)	General expenses	
199	研究開発費	当期に費用認識した研究開発費(IFRS)	Research expenses	
200	企業結合に伴う再測定による利益	企業結合に伴う再測定による利益(IFRS)	Gain on business combination	
201	その他の営業収益	その他の営業収益(IFRS)	Other operating income	
202	その他の営業費用	その他の営業費用(IFRS)	Other operating expenses	
203	その他の収益	その他の収益(IFRS)	Other gains	
204	その他の費用	その他の費用(IFRS)	Other losses	
205	持分法による投資収益(△は損失)	その他の費用(IFRS)	Other losses	
206	営業利益(△損失)	営業利益(△損失)(IFRS)	Operating profit	
207	金融収益	金融収益(IFRS)	Financial income	
208	金融費用	金融費用(IFRS)	Financial expenses	
209	税引前利益(△損失)	検索と置換		
210	法人所得税費用	<input type="button" value="検索(D)"/> <input type="button" value="置換(P)"/> 検索する文字列(N): <input type="text" value="営業利益"/> <input type="button" value="書式セットなし"/> <input type="button" value="書式(M)..."/> 検索場所(H): <input type="button" value="シート"/> <input checked="" type="checkbox"/> 大文字と小文字を区別する(C) 検索方向(S): <input type="button" value="行"/> <input checked="" type="checkbox"/> セル内容が完全に同一であるものを検索する(O) 検索対象(L): <input type="button" value="値"/> <input checked="" type="checkbox"/> 半角と全角を区別する(B) <input type="button" value="オプション(I) <<"/>		
211	維持事業からの当期利益(△損失)	<input type="button" value="すべて検索(I)"/> <input type="button" value="次を検索(E)"/> <input type="button" value="閉じる"/>		
212	非維持事業			

ファイルを開いて「詳細ツリー」シートの中で今回取得する、「営業利益」を検索します。

標準ラベル（日本語）の列で営業利益にヒットする行が206行目にありました。

図 7.3: 営業利益の要素名を確認

	用語別ラベル及び代替ラベル(英語)	名前空間プレフィックス	要素名
187	Consolidated statement of profit or loss[連結] [標準]	jipcp.cor	StatementOfProfitOrLossIFRSAbstract
188	Condensed semi-annual consolidated statement of profit or loss[中間] [標準]	jipcp.cor	StatementOfProfitOrLossIFRSTable
189	Condensed semi-annual statement of profit or loss[中間] [標準]	jipcp.cor	StatementOfProfitOrLossIFRSLineItems
190	Condensed semi-annual statement of profit or loss[中間] [標準]	jipcp.cor	ContinuingOperationsIFRSAbstract
191	Total[合計]	jipcp.cor	RevenueIFRS
192	Total[合計]	jipcp.cor	NetSalesIFRS
193	Total[合計]	jipcp.cor	Revenue2IFRS
194	Total[合計]	jipcp.cor	CostOfSalesIFRS
195	Total[合計]	jipcp.cor	GrossProfitIFRS
196	Total[合計]	jipcp.cor	SellingGeneralAndAdministrativeExpensesIFRS
197	Total[合計]	jipcp.cor	SellingExpensesIFRS
198	Total[合計]	jipcp.cor	GeneralAndAdministrativeExpensesIFRS
199	Total[合計]	jipcp.cor	ResearchAndDevelopmentExpenditureRecognizedAsExpense
200		jipcp.cor	GainOnRemeasurementRelatingToBusinessCombinationsIFRS
201		jipcp.cor	OtherOperatingIncomeIFRS
202		jipcp.cor	OtherOperatingExpensesIFRS
203		jipcp.cor	OtherIncomeIFRS
204		jipcp.cor	OtherExpensesIFRS
205	Total[合計]	jipcp.cor	OperatingProfitIFRS
206	Operating profit[正値]	jipcp.cor	OperatingProfitLossIFRS
207	Operating loss[負値]	jipcp.cor	FinanceIncomeIFRS
208	Total[合計]	jipcp.cor	FinanceCostsIFRS
209	Share of profit of investments accounted for using equity method[正値]	jipcp.cor	ShareOfProfitLossOfInvestmentsAccountedForUsingEquityMe
210	Share of loss of investments accounted for using equity method[負値]	jipcp.cor	
211	Profit (loss) before tax[代替標準]		
	Profit before tax[代替正値]		
	Profit before tax[代替負値]		
	Profit before tax from continuing operations[代替]		
	Loss before tax[代替負値]		
	Loss before tax[代替正値]		
	Profit (loss) before tax[中間標準]		
	Profit before tax[中間正値]		
	Loss before tax[中間負値]		
	Income tax expense[代替標準]		
	Profit (loss) from continuing operations[代替標準]		
	Profit from continuing operations[正値]		
	Profit from continuing operations[代替正値]		
	Loss from continuing operations[負値]		
	Loss from continuing operations[代替負値]		
	Profit (loss) from continuing operations[中間標準]		
	Profit from continuing operations[中間正値]		
	Loss from continuing operations[中間負値]		
212	Profit (loss) from discontinued operations[代替標準]		
213	Profit (loss) from discontinued operations[代替標準]		



タクソノミは要素名という列に格納されているため、206行目のうち要素名の列にかかるセルを探します。すると、「OperatingProfitIFRS」がタクソノミであることがわかりました。

図 7.4: コンテキスト ID の命名規約

図表 5-4-2 コンテキスト ID の命名規約

コンテキスト ID の命名規約	
[相対期間又は時点] [期間又は時点] ((_[メンバーの要素名]) × n) (_{連番} 3 枚)	
※各項目の詳細は「図表 5-4-4 コンテキスト ID の設定」を参照してください。	
※ディメンションを使用する場合、タグ付けする値に関連するメンバーの数だけコンテキストを定義する必要があります。命名規約の「n」は関連するメンバーの数を表す整数です。	
※「[連番 3 枚]」は、コンテキスト ID が重複した場合に二つ目からは「002」から開始する連番を付与します。	
※連番軸以外で開示書類等提出者が追加したメンバーを使用する場合は、「[メンバーの要素名]」を「[名前空間プレフィックス]_[メンバーの要素名]」に置き換えます。	

図7.5: コンテキストIDの設定

図表 5-4-4 コンテキストIDの設定

No	項目	値	説明
1	{相対期間又は時点}	CurrentYear	当年度を意味します。
2		Interim	中間期を意味します。
3		Prior1Year	前年度を意味します。
4		Prior1Interim	前中間期を意味します。
5		Prior2Year	前々年度を意味します。
6		Prior2Interim	前々中間期を意味します。
7		Prior[n]Year	[n]年度前を意味します。
8		Prior[n]Interim	[n]年度前中間期を意味します。
9		FilingDate	提出日を意味します。
10		RecordDate	議決権行使の基準日を意味します。 ^{※1}
11		RecentDate	最近日を意味します。 ^{※2, 3}
12		FutureDate	予定日を意味します。 ^{※3}
13	{期間又は時点}	Instant	時点を意味します。
14		Duration	期間を意味します。
15	{メンバーの要素名}	メンバーの要素名	メンバーの要素名を意味します。

また、財務情報は同じ項目であっても時期で区別している場合があります。期間の指定はコンテキストIDで指定します。コンテキストIDは金融庁が公表している「報告書インスタンス作成ガイドライン」に記載されているコンテキストの命名規則に則って指定されています。そのため、今期を指す場合は「CurrentYearDuration」、前期を指す場合は「Prior1YearDuration」を使用することで、情報を一意に絞って取得することができます。

これらはコード内で使用するため、メモしておいてください。

7.2.2 取得するコードを書こう

本書では、XBRLを読み取るためにArelleというXBRLに認定されたOSS(オープンソースソフトウェア)を利用します。ArelleはXBRLファイルの構造データの構築、検証、処理をするソフトウェアです。こうした解析機能を持つプログラムを一般的にパーサーと呼びます。XBRLからの認定を受けているだけでなく、Pythonのライブラリー機能、無料で利用可能、定期的にメンテナンスが行われているなどの理由から、本書ではArelleを採用します。

リスト7.1: Arelleのインストール

```
1: pip install arelle-release
```

まずXBRLファイルを解析するために、Arelleが利用可能な環境を作ります。ターミナル上で実行し、Arelleをインストールします。

これでArelleが利用可能な状態になります。続いて、コードを細かく分けて解説します。

リスト 7.2: 項目を抜き出す準備

```
1: """
2: 取得する項目をリストとして格納するための関数
3: """
4: def make_edinet_company_info_list(xbrl_file):
5:     edinet_company_info_list = []
6:     company_data = {
7:         "EDINETCODE": "",
8:         "企業名": "",
9:         "営業利益(IFRS)": "",
10:    }
```

まず有価証券報告書の中から、今回取得する項目を抜き出す準備をしています。今回は、営業利益の他に「EDINETCODE」「企業名」を取得し、営業利益の額がどの企業のものなのか一目で確認できるようにします。

また、取得したデータを入れるための箱を作ります。

リスト 7.3: Arelle の準備

```
1: """
2: Arelleの準備
3: """
4: ctrl = Cntlr.Cntlr()
5: model_manager = ModelManager.initialize(ctrl)
6:
7: model_xbrl = model_manager.load(xbrl_file)
```

Arelle を使用するためにコントローラーを初期化し、modelMager.load(xbrl_file) で XBRL データを読み込みます。

リスト 7.4: 実データを探して取得

```
1: """
2: 実データを探して取得
3: """
4: for fact in model_xbrl.facts:
5:
6:     # EDINETコードを探す
7:     if fact.concept.qname.localName == 'EDINETCodeDEI':
8:         company_data["EDINETCODE"] = fact.value
9:         print("EDIENTコード :", fact.value)
10:
11:     # 企業名を探す
```

```

12:         elif fact.concept.qname.localName == 'FilerNameInJapaneseDEI':
13:             company_data["企業名"] = fact.value
14:             print("企業名 :", fact.value)
15:
16:             # 営業利益(IFRS)を探す
17:             elif fact.concept.qname.localName == 'OperatingProfitLossIFRS':
18:                 if fact.contextID == 'CurrentYearDuration' and company_data["営業
利益(IFRS)"] == "":
19:                     company_data["営業利益(IFRS)"] = fact.value
20:                     print("営業利益(IFRS) :", fact.value)
21:
22:             # 見つけたデータをリストに入れる
23:             edinet_company_info_list.append([
24:                 company_data["EDINETCODE"],
25:                 company_data["企業名"],
26:                 company_data["営業利益(IFRS)"],
27:
28:             ])
29:
30:     return edinet_company_info_list

```

具体的な取得したい情報を取得していきます。

for 文で読み込んだデータに対し、「.concept.qname.localName =」でタグを指定し、factに望んだ実データを入れるようにループ処理をします。

該当したものは「fact.value」で取得できるようになっています。

また、二重でif文を書き、「contextID =」でコンテキスト ID を指定することにより、期間なども指定し情報をひとつに絞ることができます。

財務データの多くは今期だけではなく、前期なども記載があり任意のデータを絞り込めない、もしくは間違う可能性があるため、必ず指定しましょう。

最後に取得したインスタンスを先ほど作成したリストに入れます。

リスト 7.5: 全ての処理を実行

```

1: """
2: 元データのパスを指定し、すべての処理を実行する関数
3: """
4: def main():
5:     # あなたのXBRLファイルのパスを指定(ただコピーしても動きません)
6:     xbdl_file = r"\xxx\\ファイル名\XBRL\PublicDoc\XBRLファイル名.xbdl"
7:
8:     company_info = make_edinet_company_info_list(xbdl_file)
9:     for info in company_info:

```

```
10:     print(info)
```

ここまで処理を行う XBRL ファイルをパスで指定します。

パスとはファイルなどの居場所を示すものであり、VS Code で目的のファイルを右クリックすることで「パスをコピー」と出てくるため、そのままコピペしましょう（相対パスでも可）。また、ターミナル上で結果を確認できるように、print 文を記述します。

7.2.3 ソースコード

リスト 7.6: ソースコード

```
1: from arelle import ModelManager
2: from arelle import Cntlr
3: import os
4: import glob
5:
6: """
7: 取得する項目をリストとして格納するための関数
8: """
9: def make_edinet_company_info_list(xbrl_file):
10:     edinet_company_info_list = []
11:     company_data = {
12:         "EDINETCODE": "",
13:         "企業名": "",
14:         "営業利益(IFRS)": "",
15:     }
16:
17:     ctrl = Cntlr.Cntlr()
18:     model_manager = ModelManager.initialize(ctrl)
19:
20:     model_xbrl = model_manager.load(xbrl_file)
21:
22:     # 実データを探して取得
23:     for fact in model_xbrl.facts:
24:
25:         # EDINETコードを探す
26:         if fact.concept.qname.localName == 'EDINETCodeDEI':
27:             company_data["EDINETCODE"] = fact.value
28:
29:         # 企業名を探す
30:         elif fact.concept.qname.localName == 'FilerNameInJapaneseDEI':
31:             company_data["企業名"] = fact.value
```

```

32:
33:         # 営業利益(IFRS)を探す
34:         elif fact.concept.qname.localName == 'OperatingProfitLossIFRS':
35:             if fact.contextID == 'CurrentYearDuration':
36:                 company_data["営業利益(IFRS)"] = fact.value
37:
38:     # 見つけたデータをリストに入れる
39:     edinet_company_info_list.append([
40:         company_data["EDINETCODE"],
41:         company_data["企業名"],
42:         company_data["営業利益(IFRS)"],
43:     ])
44:
45: return edinet_company_info_list
46:
47: """
48: 元データのパスを指定し、すべての処理を実行する関数
49: """
50: def main():
51:     # あなたのXBRLファイルのパスを指定(ただコピーしても動きません)
52:     xbdl_file = r"\xxx\フォルダーネーム\XBRL\PublicDoc\XBRLファイル名.xbdl"
53:
54:     company_info = make_edinet_company_info_list(xbdl_file)
55:     for info in company_info:
56:         print(info)
57:
58:     print("extract finish")
59:
60: if __name__ == "__main__":
61:     main()

```

以下のように、トヨタのデータを取得することができました。

リスト 7.7: 出力結果

```

1: """
2: 出力結果
3: """
4: 営業利益(IFRS) : 5352934000000
5: EDIENTコード : E02144
6: 企業名 : トヨタ自動車株式会社
7: ['E02144', 'トヨタ自動車株式会社', '5352934000000']
8: extract finish

```

7.3 10社から取得してみる

ここからはより実用的にしていくために、今学んだ自動でデータを取得する技術を複数社の書類に対して行います。

今回は、トヨタ自動車株式会社を軸にするため、国際会計基準(IFRS)で連結財務諸表を使用している、営業利益が記載されている企業を対象にファイルパスを追加します。

以上の条件に合う企業を10社ピックアップしました。

- ・キッコーマン株式会社
- ・東レ株式会社
- ・パナソニックホールディングス株式会社
- ・トヨタ自動車株式会社
- ・日鉄ソリューションズ株式会社
- ・株式会社ディー・エヌ・エー
- ・サッポロホールディングス株式会社
- ・ソニーグループ株式会社
- ・アサヒグループホールディングス株式会社
- ・キリンホールディングス株式会社

EDINETから以上の10社を検索し、XBRLのボタンをクリックすることで、手動でXBRLファイルをダウンロードすることができます。

先ほどのコードと異なる点は「ファイルパスを1社から10社に増やすこと」と「それぞれの企業に対して自動でデータを取得し、次の企業に移り同じ処理を行うこと」のふたつです。

基本的には先ほどと同じことをしています。そのため、コードを変えた箇所について確認します。

リスト7.8: 複数社のXBRLファイルを読み込む

```
1: """
2: XBRLファイルを複数社すべての読み込む処理
3: """
4: for index, xbrl_file in enumerate(xbrl_files):
5:     .
6:     (省略)
7:     .
8:     print("XBRLファイルを読み込んでいます", ":", index + 1, "/",
len(xbrl_files))
9:
```

対象企業が増えたため、同じ処理をすべてのXBRLファイルで行うためにfor文でループ処理を行います。

本処理は長時間に及ぶ場合があるため、全体で何社を対象としているのか、現在どの段階まで処理が進んでいるのかを把握できるよう、print文を用いて進捗を出力しています。

リスト7.9: 正規表現でファイルパスを指定

```
1: """
2: main()で指定したXBRLファイルパスを正規表現で指定
3: あなたのXBRLファイルのパスを指定(ただコピーしても動きません)
4: """
5: xbrl_files = glob.glob(r'*\フォルダーネ名\\*\XBRL\\PublicDoc\\*.xbrl')
```

main()のところでは、先ほど同様に対象のXBRLファイルのパスを指定する必要があります。1社ずつパスを書いても正常に動くのですが、対象のファイルが増えるほど面倒であるため、正規表現を使用します。

正規表現とは、直接パスを通した対象だけに処理をするのではなく、「パターンに一致する対象すべて」を対象として処理をするための記述方法です。globモジュールはそういったときに使用され、「r」の中のクオーテーションの中がそのパターンの型になっています。また、「*」はどんな文でも、何文字でも入れることができるという意味です。XBRLファイルはファイルの階層構造がほとんど同じであり、ファイル名で識別されています。そのため、識別部分を「*」に置き換え、残りの共通部分を指定すると、単一の文字列パターンでフォルダー内のXBRLファイルをまとめて処理できるようになります。これにより、分析対象の増減が生じても、コードの大幅に変更することなく柔軟に対応可能です。

正規表現を使用するにしても、パスはそのファイルの居場所であるため、環境により異なります。ソースコードをコピペするなど、同じ通り記述してもパスが異なっていると、エラーになる場合もあります。なるべく、固有番号など、ファイルによって確実に異なるところ以外は記述することを推奨します。

7.3.1 ソースコード

リスト7.10: ソースコード

```
1: from arelle import ModelManager
2: from arelle import Cntlr
3: import os
4: import glob
5:
6: """
7: 取得する項目をリストとして保管するための関数
8: """
9: def make_edinet_company_info_list(xbrl_files):
10:     edinet_company_info_list = []
11:     """
12:     XBRLファイルを複数社すべて読み込む処理
13:     """
14:     for index, xbrl_file in enumerate(xbrl_files):
```

```
15:     company_data = {
16:         "EDINETCODE": "",
17:         "企業名": "",
18:         "営業利益(IFRS)": ""
19:     }
20:
21:     ctrl = Cntlr.Cntlr()
22:     model_manager = ModelManager.initialize(ctrl)
23:
24:     model_xbrl = model_manager.load(xbrl_file)
25:     print("XBRLファイルを読み込んでいます", ":", index + 1, "/",
len(xbrl_files))
26:
27:     # 実データを探して取得
28:     for fact in model_xbrl.facts:
29:
30:         # EDINETコードを探す
31:         if fact.concept.qname.localName == 'EDINETCodeDEI':
32:             company_data["EDINETCODE"] = fact.value
33:
34:         # 企業名を探す
35:         elif fact.concept.qname.localName == 'FilerNameInJapaneseDEI':
36:             company_data["企業名"] = fact.value
37:
38:         # 営業利益(IFRS)を探す
39:         elif fact.concept.qname.localName == 'OperatingProfitLossIFRS':
40:             if fact.contextID == 'CurrentYearDuration':
41:                 company_data["営業利益(IFRS)"] = fact.value
42:
43:         # 見つけたデータをリストに入れる
44:         edinet_company_info_list.append([
45:             company_data["EDINETCODE"],
46:             company_data["企業名"],
47:             company_data["営業利益(IFRS)"],
48:         ])
49:
50:     return edinet_company_info_list
51:
52: """
53: 元データのパスを指定し、すべての処理を実行する関数
54: """
```

```

55: def main():
56:     """
57:         main()で指定したXBRLファイルパスを正規表現で指定
58:         あなたのXBRLファイルのパスを指定(ただコピーしても動きません)
59:     """
60:     xbdl_files = glob.glob(r'*\フォルダー名\\*\*\XBRL\\PublicDoc\*\*.xbrl')
61:
62:     company_info = make_edinet_company_info_list(xbdl_files)
63:     for info in company_info:
64:         print(info)
65:
66:     print("extract finish")
67:
68: if __name__ == "__main__":
69:     main()
70:

```

▼結果

リスト7.11: 出力結果

```

1: """
2: XBRLファイルを読み込んでいます : 1 / 10
3: XBRLファイルを読み込んでいます : 2 / 10
4: XBRLファイルを読み込んでいます : 3 / 10
5: XBRLファイルを読み込んでいます : 4 / 10
6: XBRLファイルを読み込んでいます : 5 / 10
7: XBRLファイルを読み込んでいます : 6 / 10
8: XBRLファイルを読み込んでいます : 7 / 10
9: XBRLファイルを読み込んでいます : 8 / 10
10: XBRLファイルを読み込んでいます : 9 / 10
11: XBRLファイルを読み込んでいます : 10 / 10
12: ['E00435', 'キッコーマン株式会社', '66733000000']
13: ['E00873', '東レ株式会社', '57651000000']
14: ['E01772', 'パナソニックホールディングス株式会社', '360962000000']
15: ['E02144', 'トヨタ自動車株式会社', '5352934000000']
16: ['E05304', '日鉄ソリューションズ株式会社', '35001000000']
17: ['E05460', '株式会社ディー・エヌ・エー', '-28270000000']
18: ['E00393', 'サッポロホールディングス株式会社', '11820000000']
19: ['E01777', 'ソニーグループ株式会社', '1208831000000']
20: ['E00394', 'アサヒグループホールディングス株式会社', '244999000000']
21: ['E00395', 'キリンホールディングス株式会社', '150294000000']
22: extract finish

```

7.4 コラム：ライブラリーとは何か

本書において、プログラムを用いたコードの実践を行いました。その中で、コードの開始の部分に「import re」のような記載がされることがあります。この箇所では、ライブラリーのインポートをしています。

ライブラリーとは、特定のタスクやプロジェクトに関連する便利な機能をまとめたものです。ライブラリーは主に、標準ライブラリーと外部ライブラリーに分類されます。標準ライブラリーはPythonで公式に用意されたもので、datetimeモジュールやosモジュールなどです。それに対して、外部ライブラリーは別の組織か個人が用意したもので、pandasやnumpyなどです。多くは複数のモジュールがまとめられ、パッケージで提供されています。

これらは「import ○○」と記述することでコード内で使用することができ、外部ライブラリーを使用する際はこの前に「pip install ○○」とターミナルにコマンドを入力し、PCにインストールします。ライブラリーを使用することで、すでに開発された安全で質の高いプログラムを利用し、より効率的にプログラムが作成できます。

リスト7.12: モジュールを利用するためのコード

1: # モジュールを利用するためのコード

2: import numpy as np

リスト7.13: ライブラリーをインストールするためのコード

1: # ライブラリーをインストールするためのコード

2: pip install lightgbm

7.5 まとめ

本章では、XBRLを使用し有価証券報告書から企業のEDINETコードと企業名、営業利益を取得しました。財務データを取得できるようになると、同じ業種などで企業の比較などがしやすくなります。タクソノミ要素リストは情報が多く探すのが難しいですが、検索機能を駆使するなどすれば有効に使用できると思います。一方で、会計基準の違いなどによりタグが異なると一気に取得することもできないため、そのような会計のドメイン知識も必要になってきます。また、XBRLのタグは正しく付与されていないことも時々あり、そのせいでデータがうまく取得できないと嘆く人も多いです。多くのXBRLユーザーがXBRLに苦労するところがタグなのです。このタグの操作は実際にやってみないとわからない点が多いので、実際に手を動かしながら試すことが一番の学習となります。

次章では、非財務情報であるテキストデータの取得について解説します。

第8章 事業等のリスクのテキストデータを自動で取得する

8.1 本章の目的

本章では、有価証券報告書から事業等のリスクという項目にあるテキストデータを取得します。有価証券報告書は財務データのみならず、現在の事業の状況や財務データが算出された背景などについての記載があります。こういった文章データはテキストデータと呼ばれ、財務データだけでは把握しきれない企業の実態をより深く理解することができます。近年のAIの発展に伴い、テキストデータの解析技術が向上しているため、有価証券報告書を用いたテキスト分析への注目が高まっています。本章ではこのテキストデータを分析するための情報収集として、10社からそれぞれの企業の事業等のリスクという項目のデータを取得します。

8.2 事業等のリスクのテキストデータを自動で取得する

今回取得するテキストデータの項目は、事業等のリスクです。事業等のリスクは、企業の財政状態や経営状況に重要な影響を与える可能性がある主要なリスクについて記載がある項目です。また、リスクが起こりうる可能性の程度や時期、対応策などの記載があります。企業の特色が出やすいだけでなく、同じ企業でも事業年度ごとにも記載内容が大きく変わることもある項目です。よって、注目度も高く、テキスト分析などがされやすい項目です。

本章でも、トヨタ自動車株式会社を軸にしたデータ取得を試みます。なお、大枠は前章のコードと同じであるため、本章では初めから10社分のデータ取得を試みます。

8.2.1 タクソノミの確認

前章と同様に、タクソノミをタクソノミ要素リストから確認します。

テキストデータに関わる部分は、国際会計基準であっても基本的にタクソノミ要素リストに格納されています。そのため、今回は新たにEDINETから最新のタクソノミ要素リストをダウンロードして、タクソノミを確認します。

図8.1: 要素リスト種別

日本会計基準のタクソノミ

IFRSのタクソノミ

(e) タクソノミ要素リスト (EXCEL:2,861KB)	(非財務データのタクソノミ)
EDINETタクソノミ（財務諸表本表タクソノミ及び国際会計基準タクソノミを除く。）に設定されている要素を一覧表示したものです。	
(f) 勘定科目リスト (EXCEL:1,397KB)	(財務データのタクソノミ)
EDINETタクソノミのうち、財務諸表本表タクソノミに設定されている勘定科目を一覧表示したものです。	
(g) 国際会計基準タクソノミ要素リスト (EXCEL:305KB)	(主に財務データ、一部注記など)
EDINETタクソノミのうち、国際会計基準タクソノミに設定されている勘定科目等の要素を一覧表示したものです。	

前章の参考に、タクソノミ要素リストをダウンロードしてください。

図8.2: タクソノミ要素リスト目次

タクソノミ要素リスト 目次

タクソノミ要素リストについて	
1	DE (jdei)
2	企業内容等の開示に関する内閣府令 第二号様式 有価証券届出書 (jpcrp020000-srs)
3	企業内容等の開示に関する内閣府令 第二号の二様式 有価証券届出書 (jpcrp020200-srs)
4	企業内容等の開示に関する内閣府令 第二号の三様式 有価証券届出書 (jpcrp020300-srs)
5	企業内容等の開示に関する内閣府令 第二号の四様式 有価証券届出書 (jpcrp020400-srs)
6	企業内容等の開示に関する内閣府令 第二号の五様式 有価証券届出書 (jpcrp020500-srs)
7	企業内容等の開示に関する内閣府令 第二号の六様式 有価証券届出書 (jpcrp020600-srs)
8	企業内容等の開示に関する内閣府令 第二号の七様式 有価証券届出書 (jpcrp020700-srs)
9	企業内容等の開示に関する内閣府令 第三号様式 有価証券報告書 (jpcrp030000-asr)
10	企業内容等の開示に関する内閣府令 第二号の二様式 有価証券報告書 (jpcrp030200-asr)
11	企業内容等の開示に関する内閣府令 第四号様式 有価証券報告書 (jpcrp040000-asr)
12	企業内容等の開示に関する内閣府令 第四号の三様式 四半期報告書 (jpcrp040300-qs)
13	企業内容等の開示に関する内閣府令 第五号様式 半期報告書 (jpcrp050000-ssr)
14	企業内容等の開示に関する内閣府令 第五号の二様式 半期報告書 (jpcrp050200-ssr)
15	企業内容等の開示に関する内閣府令 第五号の三様式 臨時報告書 (jpcrp050300-esr)
16	企業内容等の開示に関する内閣府令 第五号の四様式 通期報告書 (jpcrp050400-epr)

ファイルをダウンロードし開くと、目次のシートが表示されます。

目次から「9 企業内容等の開示に関する内閣府令 第三号様式 有価証券報告書 (jpcrp03000-asr)」のシートに移ります。

図 8.3: タクソノミ要素リスト内で検索

冗長ラベル(日本語)	構造(英語)	冗長ラベル(英語)
314 連結会社等【メンバー】	Other investees excluded from consolidated accounting group	Other investees excluded from consolidated accounting group [member]
315 コード及び温室効果ガス排出量【表示項目】	Scope 1 and 2 greenhouse gas emissions	Scope 1 and 2 greenhouse gas emissions [line items]
316 コードの温室効果ガス排出量、コード及び他の温室効果ガス排出量	Gross scope 1 greenhouse gas emissions	Gross scope 1 greenhouse gas emissions, Scope 1 and 2 greenhouse
317 コードの温室効果ガス排出量、コード及び他の温室効果ガス排出量	Gross scope 2 greenhouse gas emissions	Gross scope 2 greenhouse gas emissions, Scope 1 and 2 greenhouse
318 コード及び他の温室効果ガス排出量、コード及び他の温室効果ガス排出量	Gross scope 1 and 2 greenhouse gas emissions	Gross scope 1 and 2 greenhouse gas emissions, Scope 1 and 2 greenhous
319 000-0003-2031-2-01-pre.xml		
320 クロナリティに関する考え方及び範囲【目次項目】	Disclosure of Sustainability-related Financial Information	Disclosure of Sustainability-related Financial Information [heading]
321 コードの温室効果ガス排出量	Gross scope 3 greenhouse gas emissions	Gross scope 3 greenhouse gas emissions
322 コード、2段次の直観的構造の考え方及び範囲	Gross scope 1, 2 and 3 greenhouse gas emissions	Gross scope 1, 2 and 3 greenhouse gas emissions
323 ビジネスリスク	Business risks	Business risks [text block]
324 事業等のリスク【テキストブロック】	Description, analysis and responses to significant events or circumstances related to operating concern risks	Description, analysis and responses to significant events or circumstances related to operating concern risks [text block]
325 重要な事象等の内容、分析及び対応策、事業等のリスク【テキストブロック】	Management analysis of financial position, operating results and cash flows	Management analysis of financial position, operating results and cash flows [text block]
326 重要な位置づけ、経営戦略及びチャレンジ・コードの状況の分析【目次項目】	Management analysis of financial position, operating results and cash flows	Management analysis of financial position, operating results and cash flows [heading]
327 重要な位置づけ、経営戦略及びチャレンジ・コードの状況の分析	Management analysis of financial position, operating results and cash flows	Management analysis of financial position, operating results and cash flows [text block]
328 研究開発活動【目次項目】	Research and development activities	Research and development activities [heading]
329 研究開発活動【テキストブロック】	Research and development activities	Research and development activities [text block]
330 研究開発活動【表】	Research and development expenses	Research and development expenses, Research and development act
331 研究開発活動【目次項目】	Research and development activities	Research and development expenses, Research and development act [heading]
332 研究開発活動【表】	Research and development expenses	Research and development expenses, Research and development act [table]
333 事業セグメント【目次】	Operating segments	Operating segments [axis]
334 研究開発費、研究開発活動【表】	Research and development expenses	Research and development expenses, Research and development act [table]
335		
336 連結会計又は会社合計【メンバー】※ディメンション・オフセット	Group total or entity total [member]	Group total or entity total [member]

検索バーで取得したい「事業等のリスク」項目を検索します。「冗長ラベル（日本語）」という列でいくつかヒットしました。そのうち[テキストブロック]と書かれているものを参照します。[目次項目]などと書かれているものでは、適切にテキストデータを取得できないので注意して下さい。

今回は324行目にヒットしました。

図 8.4: タクソノミ要素リストから参照

要素名	type	substitutionGroup
1 名前空間プレフィックス	nonnum:domainItemType	xbrl:item
314 cor OtherInvesteesExcludedFromConsolidatedAccountingGroupMember	nonnum:domainItemType	xbrl:item
315 cor Scope1And2GreenhouseGasEmissionsLineItems	xbrl:stringItemType	xbrl:item
316 cor GrossScope1GreenhouseGasEmissionsScope1And2GreenhouseGasEmissions	attr	xbrl:item
317 cor GrossScope2GreenhouseGasEmissionsScope1And2GreenhouseGasEmissions	attr	xbrl:item
318 cor GrossScope1And2GreenhouseGasEmissionsScope1And2GreenhouseGasEmissions	attr	xbrl:item
319 cor		xbrl:item
320 cor DisclosureOfSustainabilityrelatedFinancialInformationHeading	xbrl:stringItemType	iod:identifierItem
321 cor GrossScope3GreenhouseGasEmissions	attr	xbrl:item
322 cor GrossScope12and3GreenhouseGasEmissions	attr	xbrl:item
323 cor BusinessRiskTextBlock	ubl:tableRowType	iod:identifierItem
324 cor BusinessRisksTextBlock	nonnum:textBlockItemType	xbrl:item
325 cor MaterialMattersRelatingToGoingConcernEtcBusinessRisksTextBlock	nonnum:textBlockItemType	xbrl:item
326 cor ManagementAnalysis	emType	iod:identifierItem
327 cor ManagementAnalysis	tBlockItemType	xbrl:item
328 cor CriticalContract	emType	iod:identifierItem
329 cor CriticalContract	tBlockItemType	xbrl:item
330 cor ResearchAndDevelopment	emType	iod:identifierItem
331 cor ResearchAndDevelopment	tBlockItemType	xbrl:item
332 cor ResearchAndDevelopment	emType	iod:identifierItem
333 cor ResearchAndDevelopment	tBlockItemType	xbrl:item
334 cor ResearchAndDevelopment	emType	iod:identifierItem
cor OperatingSegments	emType	xbrld:hypercubeItem
335		
336 cor EntityTotalMember	mainItem	xbrl:item

タクソノミは、要素名という列に格納されているため、324行目のうち要素名に関わるセルを探すと「BusinessRisksTextBlock」がタクソノミであることがわかりました。

図8.5: コンテキストIDの設定

図表 5-4-4 コンテキストIDの設定

No	項目	値	説明
1	[相対期間又は時点]	CurrentYear	当年度を意味します。
2		Interim	中間期を意味します。
3		Prior1Year	前年度を意味します。
4		Prior1Interim	前中間期を意味します。
5		Prior2Year	前々年度を意味します。
6		Prior2Interim	前々中間期を意味します。
7		Prior{n}Year	[n]年度前を意味します。
8		Prior{n}Interim	[n]年度前中間期を意味します。
9		FilingDate	提出日を意味します。
10		RecordDate	議決権行使の基準日を意味します。 ^{※1}
11		RecentDate	最近日を意味します。 ^{※2,3}
12		FutureDate	予定日を意味します。 ^{※3}
13	[期間又は時点]	Instant	時点を意味します。
14		Duration	期間を意味します。
15	[メンバーの要素名]	メンバーの要素名	メンバーの要素名を意味します。

テキストデータではコンテキストIDを指定して期間を明確に指定しなくても取得はできますが、しておいた方がより正確に取得ができます。なお、財務データとは異なり、事業等のリスクでは提出日時点での事業等のリスクという扱いになるため、コンテキストIDは「FilingDateInstant」となります。

8.2.2 取得するコードを書こう

大まかなコードの形は前章と変わりません。

取得する対象が財務データである「営業利益(IFRS)」からテキストデータである「事業等のリスク」に代わっているため、タクソノミを変える必要があります。

タクソノミを先ほど調べた、以下のものに書き変えましょう。

タグ(要素ID)がBusinessRisksTextBlock、コンテキストIDがFilingDateInstantを指定します。

リスト 8.1: 事業等のリスクを探す

```

1: # 事業等のリスクを探す
2: if fact.concept.qname.localName == 'BusinessRisksTextBlock':
3:     if fact.contextID == 'FilingDateInstant':
4:         company_data["事業等のリスク"] = fact.value

```

3章で少し触れたように、XBRLファイル内のテキストデータ(インスタンス)には、実データのみならず、付加情報が付いています。テキストデータで言えば、HTML情報が付いているため、そのまま取得するだけでは私たちはうまく読むことができません。そのため、データクレンジングをします。

リスト8.2: データクレンジングをする

```
1: """
2: Beautiful Soupとreモジュールを使用してデータクレンジングをする
3: """
4: soup = BeautifulSoup(company_data["事業等のリスク"], "html.parser")
5: company_data["事業等のリスク"] = soup.get_text()
6:
7: company_data["事業等のリスク"] = re.sub(r'\s', '', company_data["事業等のリス
ク"]).strip()
```

今回はBeautiful Soupというライブラリーと、reモジュールの両方を使用してデータクレンジングをします。ふたつを使用することで、より私たちが読むことができる状態のデータを取得できます。Beautiful SoupはHTMLタグの除去する役割を、reモジュールは書類ごとで空白や改行を全て除去し統一する役割を担っています。

これらどちらか片方のみを使用しても、似たようなことはできます。しかし、Beautiful Soupだけだと空白や改行がたくさん残ってしまったり、reモジュールだけではBeautiful Soupほど高精度でタグやネストを処理できなかったりといった問題が残ります。両者のライブラリーを利用することで双方の弱点を補完し、綺麗にテキストデータを取得することが可能になります。

8.2.3 ソースコード

リスト8.3: ソースコード

```
1: from arelle import ModelManager
2: from arelle import Cntlr
3: import os
4: import glob
5: import re
6: from bs4 import BeautifulSoup
7:
8: """
9: 取得する項目をリストとして格納するための関数
10: """
11: def make_edinet_company_info_list(xbrl_files):
12:     edinet_company_info_list = []
13:     for index, xbrl_file in enumerate(xbrl_files):
14:         company_data = {
15:             "EDINETCODE": "",
16:             "企業名": "",
17:             "事業等のリスク": "",
18:         }
```

```
19:
20:     ctrl = Cntlr.Cntlr()
21:     model_manager = ModelManager.initialize(ctrl)
22:
23:     model_xbrl = model_manager.load(xbrl_file)
24:     print("XBRLファイルを読み込んでいます", ":", index + 1, "/",
25:           len(xbrl_files))
26:
27:     # 実データを探して取得
28:     for fact in model_xbrl.facts:
29:
30:         # EDINETコードを探す
31:         if fact.concept.qname.localName == 'EDINETCodeDEI':
32:             company_data["EDINETCODE"] = fact.value
33:
34:         # 企業名を探す
35:         elif fact.concept.qname.localName == 'FilerNameInJapaneseDEI':
36:             company_data["企業名"] = fact.value
37:
38:         # 事業等のリスクを探す
39:         elif fact.concept.qname.localName == 'BusinessRisksTextBlock':
40:             if fact.contextID == 'FilingDateInstant':
41:                 company_data["事業等のリスク"] = fact.value
42:                 """
43:                 BeautifulSoupとreモジュールを使用してデータクレンジングをする
44:                 """
45:                 soup = BeautifulSoup(company_data["事業等のリスク"],
46:                                     "html.parser")
47:                 company_data["事業等のリスク"] = soup.get_text()
48:
49:                 company_data["事業等のリスク"] = re.sub(r'\s', '',
50:                     company_data["事業等のリスク"]).strip()
51:
52:                 # 見つけたデータをリストに入れる
53:                 edinet_company_info_list.append([
54:                     company_data["EDINETCODE"],
55:                     company_data["企業名"],
56:                     company_data["事業等のリスク"],
57:                 ])
```

```
56:     return edinet_company_info_list
57:
58: """
59: 元データのパスを指定し、すべての処理を実行する関数
60: """
61: def main():
62:     # あなたのXBRLファイルのパスを指定(ただコピーしても動きません)
63:     xbdl_files = glob.glob(r'*\*\*\*\\XBRL\\PublicDoc\\*.xbdl')
64:
65:     company_info = make_edinet_company_info_list(xbdl_files)
66:     for info in company_info:
67:         print(info)
68:
69:     print("extract finish")
70:
71: if __name__ == "__main__":
72:     main()
```

こちらでは、事業等のリスクに変更しているだけなので、すぐに10社分のデータを集めるコードを提示しています。表示量が多いので工夫をしたり、CSVファイルで出力など工夫をしてみてください。

8.3 まとめ

本章はテキストデータの取得について学びました。取得した事業等のリスクは有価証券報告書の中でも目にしやすい項目のひとつです。このように欲しいテキストデータのみを取得できると、企業の比較や分析など様々な用途で便利に使用することができます。きれいなデータを手元に持たないと分析ができないため、データ収集として一時データから取得するこのやり方は非常に有効だと言えます。何度も試してやり方を覚えていただけるといいと思います。

次章は、企業が独自でタクソノミを振った提出者別タクソノミの項目から情報の取得に臨みます。

第9章 提出者別タクソノミのデータを取得する～ソニーのコンテンツ価値を取得する～(新規)

9.1 本章の目的

本章では、提出者別タクソノミのデータ取得について取り扱います。本章では、ソニーグループ株式会社の有価証券報告書に記載されている「コンテンツ資産」という勘定科目を取得していきます。EDIENT タクソノミにない項目のデータを取得することで、企業の特徴を知ることができます。本章では、提出者別タクソノミの扱い方を理解することを目的とします。

9.2 提出者別タクソノミの注意点

提出者別タクソノミの項目も、EDINET タクソノミと同様に、タクソノミをパーサーに読み込ませることで情報を取得します。しかし、提出者別タクソノミの情報は以下の点に注意する必要があります。

- ・タクソノミを確認することが難しい
- ・同じ意味の項目でもタクソノミが一致しない可能性がある

タクソノミの確認が難しい理由は、これまで使用したタクソノミ要素リストでは EDINET タクソノミしか含まれておらず、提出者別タクソノミを確認できないためです。そこで提出者別タクソノミを確認する方法として、主に次のふたつが挙げられます。

まず、EDINET で XBRLとともに提供されている CSV ファイルを用いる方法です。この方法では CSV ファイルから該当する提出者別タクソノミを確認するため、比較的扱いやすいことが利点です。一方でデータが欠落している場合があり、確実にタクソノミを確認できない欠点があります。もうひとつの方は、XBRL 内に定義されたラベルリンクベースを用いる方法です。この方法では実際に使われるタクソノミ情報を参照できるため、提出者別タクソノミを確実に確認できることが利点です。一方で XML や XBRL の知識が前提となるため、学習コストが高く、すぐに活用が難しいことが欠点です。

どちらの方法も無視できない欠点が存在し、提出者別タクソノミを確認することは容易ではありません。

加えて、提出者別タクソノミでは同じ意味の項目であっても、企業ごとに異なるタクソノミが付けられる場合があります。これは 3 章で述べたように、各企業が独自に提出者別タクソノミを作成していることに起因しています。そのため、各企業ごとに提出者別タクソノミの情報を確認する必要があり、非常に大変な作業となります。

以上のように、提出者別タクソノミを扱うことは非常に煩雑で手間のかかるものです。使用したい項目はどこなのか、それらに対応するタクソノミはどれか、しっかり調べることが重要になります。

9.3 取得する勘定科目について

今回取得するのは、ソニーグループ株式会社の「コンテンツ資産」です。有価証券報告書の注記欄の引用によると、コンテンツ資産は「繰延映画製作費、テレビ放映権、ミュージック・カタログ、アーティスト・コントラクト、音楽配信権及びゲームコンテンツ」を合わせた資産であり、連結財政状態計算書にて表示されている勘定科目です。

この項目から、ソニーグループ株式会社が保有するコンテンツにどれくらいの価値がついているのか知ることができます。

9.4 タクソノミを確認

先ほども紹介したように、タクソノミ要素リストや勘定科目リストではEDINETタクソノミの一覧が見られますが、それらでは提出者別タクソノミが記述されていないため、確認することができません。そのため、本書では確実に調べられるラベルリンクベースを使用して提出者別タクソノミの確認をします。

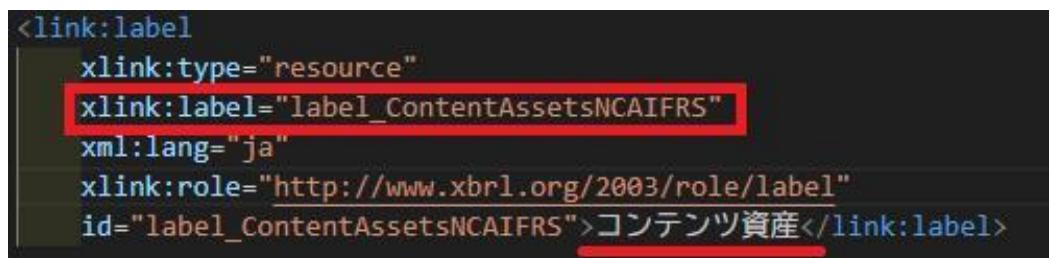
ラベルリンクベースでは、スキーマで定義される要素に対して名前（ラベル）を定義しています。4章で紹介したように、タクソノミはスキーマとリンクベースで構成されているため、それらを確認することで確実にタクソノミのタグを把握できます。タクソノミと今まで呼んでいたものはXMLの目線から見ると、スキーマで定義されている要素名です。それをXBRLを扱う上で、やや柔らかい言いまわしを使用しているにすぎません。

つまり、タクソノミを指定するということは、要素名を指定することと同義です。8、9章でタクソノミを調べるときにタクソノミ要素リスト「要素名」のカラムを参照したのは、このためです。

よって、ラベルリンクベース内で日本語名称の検索をかけることで、タクソノミ（要素名）を見つけ出すことができます。

ラベルリンクベースを参照するには、ダウンロードしたXBRLファイルのフォルダー内から「○○_lab.xml」のような拡張子がxmlであり、lab(ラベル)を表すファイル名を探します。

図9.1: ラベルリンクベースに記載されるコンテンツ資産



ラベルリンクベースでは、画像のような`<link:label></link:label>`で囲われたものが多く存在しており、タクソノミの名称が定義されています。

その中で `xlink:label="で"` に囲われているものからタクソノミの要素名を探します。ラベルリンクベースに書かれているタクソノミにはすべてlabel_などの単語が加えられており、実際のXBRL

ファイルではそのような単語がつけられていないため、注意が必要です。

また、似た名称のラベルが複数ヒットすることがあります。提出者別タクソノミは人の手により新たに追加されたものであり、複雑なものになってしまいます。対処法として要素名を翻訳して、どのような内容のタクソノミなのかを推察し、XBRLファイルで一度検索をかけることを推奨します。

図9.2: ファイル内で検索をかける

```
</xbrli:unit> > ContentAssetsNCAIFRS  
<jpcrp030000-asr_E01777-000:InvestmentsAndAdvancesInTheFinancialServicesSegmentN  
<jpigg_cor:PropertyPlantAndEquipmentIFRS contextRef="Prior2YearInstant" decimals  
<jpigg_cor:PropertyPlantAndEquipmentIFRS contextRef="Prior1YearInstant" decimals  
<jpigg_cor:PropertyPlantAndEquipmentIFRS contextRef="CurrentYearInstant" decimal  
<jpigg_cor:RightOfUseAssetsIFRS contextRef="Prior2YearInstant" decimals="-6" uni  
<jpigg_cor:RightOfUseAssetsIFRS contextRef="Prior1YearInstant" decimals="-6" uni  
<jpigg_cor:RightOfUseAssetsIFRS contextRef="CurrentYearInstant" decimals="-6" un  
<jpigg_cor:GoodwillIFRS contextRef="Prior2YearInstant" decimals="-6" unitRef="JP  
<jpigg_cor:GoodwillIFRS contextRef="Prior1YearInstant" decimals="-6" unitRef="JP  
<jpigg_cor:GoodwillIFRS contextRef="CurrentYearInstant" decimals="-6" unitRef="J  
<jpcrp030000-asr_E01777-000:ContentAssetsNCAIFRS contextRef="Prior2YearInstant"  
<jpcrp030000-asr_E01777-000:ContentAssetsNCAIFRS contextRef="Prior1YearInstant"  
<jpcrp030000-asr_E01777-000:ContentAssetsNCAIFRS contextRef="CurrentYearInstant'  
<jpigg_cor:OtherIntangibleAssetsIFRS contextRef="Prior2YearInstant" decimals="-6  
<jpigg_cor:OtherIntangibleAssetsIFRS contextRef="Prior1YearInstant" decimals="-6
```

XBRLファイル内でタクソノミの存在やcontext_refの値などを確認することで、確実にタクソノミを探せます。

コンテンツ資産についてプレフィックスを含めたタクソノミが「jpcrp030000-asr_E01777-000:ContentAssetsNCAIFRS」、コンテキストIDが当期のものであれば「CurrentYearInstant」であることが確認できました。

実際にコードで使用するタクソノミのタグは「ContentAssetsNCAIFRS」であり、コンテキストIDは「CurrentYearInstant」です。

9.5 コンテンツ資産のデータを取得

ソニーグループ株式会社では、コンテンツ資産が2022年に提出されたものから記載されているため、その時点から現在まで過去3年分の取得を行います。また、同じ企業の情報を時系列別に取得するためグラフを描画し、視覚的にわかりやすくします。

大まかな構造は8章で取り扱ったものと同様のものとなります。しかし、今回はソニーグループ株式会社の情報のみの取得のため、企業名は取得しません。

よって、8章で使用したコードにおいて変更のある部分を解説します。

リスト9.1: タクソノミとインスタンスを指定

```
1: if fact.concept.qname.localName == 'ContentAssetsNCAIFRS' :  
2:     if fact.contextID == 'CurrentYearInstant':  
3:         company_data["コンテンツ資産"] = fact.value
```

先ほど取得したタクソノミとインスタンスを指定します。

リスト9.2: 会計年度データを追加

```
1: if fact.concept.qname.localName == 'FiscalYearCoverPage':  
2:     company_data["会計年度"] = fact.value
```

また、過去3年のデータを扱うため、会計年度も追加で取得します。会計年度のタクソノミは「FiscalYearCoverPage」です。

リスト9.3: グラフを描画

```
1: def graph_plot(data):  
2:     df = pd.DataFrame(data, columns=["会計年度", "コンテンツ資産"])  
3:     df["コンテンツ資産"] = pd.to_numeric(df["コンテンツ資産"], errors='coerce')  
4:     df = df.dropna()  
5:  
6:     plt.figure(figsize=(10, 5))  
7:     plt.plot(df["会計年度"], df["コンテンツ資産"], marker='o')  
8:     plt.title("コンテンツ資産の推移")  
9:     plt.xlabel("会計年度")  
10:    plt.ylabel("コンテンツ資産")  
11:    plt.grid(True)  
12:  
13:    # 数値を3桁区切りで表示  
14:    ax = plt.gca()  
15:    ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, _:  
f"{{x:.0f}}"))  
16:  
17:    plt.show()
```

グラフ描画ライブラリーであるmatplotlibを使用します。

pd.DataFrame()でmatplotlib専用の型にデータを整形し、plot・xlabel・ylabelなどのモジュールでグラフの詳細を設定しています。

さらにgcaモジュールを使用し、グラフに金額が描画された際に3桁区切りで表示されるように設定しています。

9.5.1 ソースコード

リスト 9.4: ソースコード

```
1: from arelle import ModelManager
2: from arelle import Cntlr
3: import os, time, glob, re
4: from bs4 import BeautifulSoup
5: import pandas as pd
6: import matplotlib.pyplot as plt
7: import matplotlib.ticker as ticker
8: import japanize_matplotlib
9:
10: """
11: 提出者別タクソノミを含めた企業情報を取得する関数
12: """
13: def make_edinet_company_info_list(xbrl_files):
14:     edinet_company_info_list = []
15:     for index, xbrl_file in enumerate(xbrl_files):
16:         company_data = {"会計年度": "", "コンテンツ資産": ""}
17:
18:         ctrl = Cntlr.Cntlr()
19:         model_manager = ModelManager.initialize(ctrl)
20:         model_xbrl = model_manager.load(xbrl_file)
21:         print("XBRLファイルを読み込んでいます", ":", index + 1, "/",
len(xbrl_files))
22:
23:         for fact in model_xbrl.facts:
24:             if fact.concept.qname.localName == 'FiscalYearCoverPage':
25:                 company_data["会計年度"] = fact.value
26:             elif fact.concept.qname.localName == 'ContentAssetsNCAIFRS' :
27:                 if fact.contextID == 'CurrentYearInstant':
28:                     company_data["コンテンツ資産"] = fact.value
29:
30:             edinet_company_info_list.append([
31:                 company_data["会計年度"],
32:                 company_data["コンテンツ資産"],
33:             ])
34:
35:     return edinet_company_info_list
36:
37: def graph_plot(data):
```

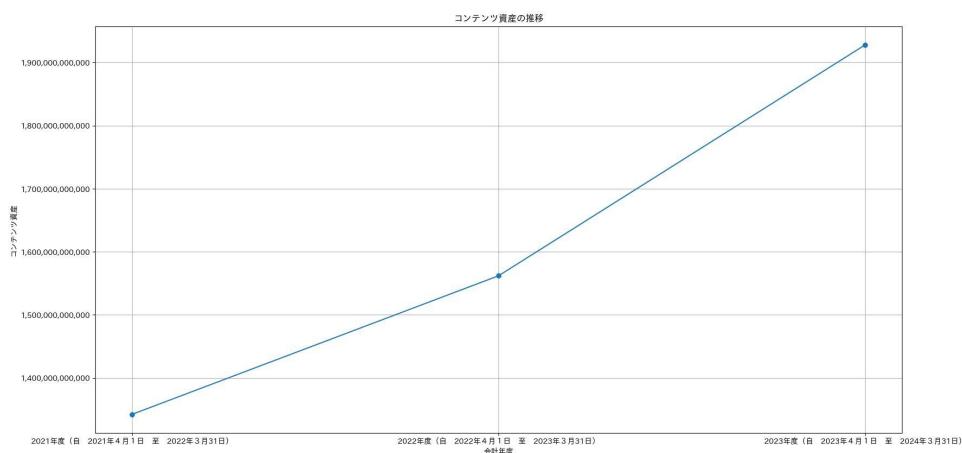
```
38:     df = pd.DataFrame(data, columns=["会計年度", "コンテンツ資産"])
39:     df["コンテンツ資産"] = pd.to_numeric(df["コンテンツ資産"], errors='coerce')
40:     df = df.dropna()
41:
42:     plt.figure(figsize=(10, 5))
43:     plt.plot(df["会計年度"], df["コンテンツ資産"], marker='o')
44:     plt.title("コンテンツ資産の推移")
45:     plt.xlabel("会計年度")
46:     plt.ylabel("コンテンツ資産")
47:     plt.grid(True)
48:
49:     # 数値を3桁区切りで表示
50:     ax = plt.gca()
51:     ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, _:
f"{x:,.0f}"))
52:
53:     plt.show()
54:
55: def main():
56:     # あなたのXBRLファイルのパス(ただコピーしても動きません)
57:     xbdl_file = glob.glob(r'*\フォルダ名\*\*\*\XBRL\PublicDoc\*.xbrl')
58:
59:     company_info = make_edinet_company_info_list(xbdl_file)
60:     for info in company_info:
61:         print(info)
62:
63:     graph_plot(company_info)
64: if __name__ == "__main__":
65:     main()
```

9.5.2 実行結果

リスト 9.5: 実行結果

```
1: ['2021年度（自\u30002021年4月1日\u3000至\u30002022年3月31日）', '1342046000000']
2: ['2022年度（自\u30002022年4月1日\u3000至\u30002023年3月31日）', '1561882000000']
3: ['2023年度（自\u30002023年4月1日\u3000至\u30002024年3月31日）', '1928113000000']
```

図 9.3: コンテンツ資産の推移



9.6 まとめ

今回は、提出者別タクソノミの扱い方について学びました。提出者別タクソノミは同じ項目でも企業によりタクソノミの名称が異なり、そもそもタクソノミを見つけられないなどデメリットもあります。しかし、その企業特有の詳細な情報を得ることができる貴重な情報源です。同じ企業のデータを年度別に比較するなど、分析することで企業について深く知ることができます。

次章では、これまで取得してきたデータを個人のデータベースに貯めていきます。

第10章 自前データベースを持つ（新規）

10.1 本章の目的

本書では、これまでXBRLから様々な情報を取得する方法を紹介しました。取得した情報はさらにデータベースに保存しておくことで、今後さらに効率よく情報を活用することができます。

本章では、XBRLで取得したデータを自身でデータベース管理することを目的とします。

10.2 データベースとは

データベース（DB）は、データを保存し管理するための仕組みです。一定のデータを集めて規則に沿ってまとめます。つまり、ただデータを入れているものではなく、ルールに則り整理されています。昔は紙で管理されていた有価証券報告書も、広義ではデータベースの一例と考えることができます。現在では、コンピューター上にデータベースを構築し、多くの場面で活用されています。紙では管理しきれない量のデータであっても、データベースで管理することにより、目的のデータに簡単にアクセスできたり、データ紛失のリスクを避けるといったことが望めます。

また、その他にもデータベースを使用するメリットは、以下のようなものがあります。

- ・データの一元管理ができる
- ・データの一貫性を保てる
- ・保存できるデータ量が多い
- ・既存のシステムとの連携が可能

10.3 使用する技術

- ・使用言語：Python
- ・主な対象：XBRL インスタンス文書
- ・データベース：SQLite
- ・ライブラリー：SQLAlchemy

SQLiteは一般的なデータベースとは違い、ファイル形式でデータを保存するデータベースです。ファイルのため扱いがわかりやすく、学習に適しています。

SQLAlchemyはSQLiteやその他データベースをより簡単に操作するためのライブラリーです。可読性が高く、より感覚的にプログラムすることができます。また、SQLAlchemyはコードをほとんど変えずにSQLite以外のリレーションナルデータベース（RDB）に変更して、同じ内容の処理を行うことができるため、非常に便利です。SQLite以外のリレーションナルデータベースを使用してみたい、という方も、本章のコードは参考にしてください。

10.4 実際にDBを作つてみよう

実際にデータベースを作成します。今回は、データベースの作成手順を理解するために、あえて小規模のものを作成していきます。一方で、データベースは扱うデータが多いほど真価を發揮し、より大きなメリットを得ることができます。

本書で紹介する内容はあくまで一例ですので、自身の状況に合わせて必要な情報を蓄積し、大規模なデータベースの運用にもぜひ取り組んでください。

10.4.1 想定する状況

今回は第8章で使ったプログラムを用いて10社の営業利益（IFRS）を取得し、データベースに保存します。

10.4.2 データベース設計

以下の表のようなテーブルが、ひとつのリレーショナルデータベースを作成します。

表 10.1: データベース設計

EDINETCODE	企業名	営業利益（IFRS）
E001	株式会社 A	1,000,000
E002	株式会社 B	1,500,000

10.4.3 コードの解説

リスト 10.1: データベースのモデル定義と基本設定

```
1: """
2: データベースのモデル定義と基本設定
3: """
4: DATABASE_URL = "sqlite:///company_data.db"
5: Base = declarative_base()
6:
7: class Company(Base):
8:     __tablename__ = 'company'
9:     edinet_code = Column(String, primary_key=True)
10:    name = Column(String, nullable=False)
11:    OperatingProfitLoss = Column(Integer, nullable=False)
```

データベースの基礎となる骨組みを設定します。

DATABASE_URLでは、接続するデータベースがどの種類のデータベースなのか、どの場所にあるのかを定義します。

declarative_base()を使用し、sqlalchemyの基底クラスを作成します。これはテーブルを定義す

る際の元の骨格のようなものであり、Base を定義してそれを継承したテーブルを作成することで、 SQLAlchemy の機能を使用することができるようになります。

class としてテーブルの形を設定します。Class() の引数に先ほど作成した Base をいれて、 SQLite にこれから定義するテーブルの形を認識させます。`_tablename_` はテーブル名を定義し、それより下のコードで実際のテーブル内でどのようなカラム（列）を用意するのか定義します。

今回は EDINETCODE、企業名、営業利益（IFRS）の3つの列を持たせます。Column() の引数では、その列で課す制約を定義することができます。引数内で初めに書かれている Integer や String はそれぞれ数値や文字列を表しており、そこに書かれた型以外のデータは列に登録することができません。また、EDINET_CODE は企業ごとに付けられている一意な値のため、primary_key に設定します。企業名と営業利益（IFRS）については nullable=False としているため、データがなかった場合だとデータベースにはそれに関わるデータが登録されずエラーが出てしまいます。今回は使用していませんが、これら以外にも多くの制約を定義することができます。

リスト 10.2: データベースのエンジンとセッションをセットアップする関数

```
1: """
2: データベースのエンジンとセッションをセットアップする関数
3: """
4: def setup_database():
5:     engine = create_engine(DATABASE_URL)
6:     Base.metadata.create_all(engine)
7:     Session = sessionmaker(bind=engine)
8:     return Session()
```

主にデータベースとの接続に関するものを設定します。

`create_enjin()` を使用し、SQLite との接続を管理するエンジンというものを作成します。エンジンは実際のデータベースとの接続を管理するだけでなく、プログラムの命令を受けデータを操作する役割も担います。ここでエンジンは DATABASE_URL を読み込み、company_data.db という名前の SQLite ファイルに接続すればいいと判断します。

セッションの作成では、`session` と呼ばれるプログラム上でデータベース内のデータを操作するための関数を定義します。これがないとデータベースからデータを抽出することもできなければ、新しいデータを入れることもできません。

リスト 10.3: 企業データをデータベースに生成する関数

```
1: """
2: 企業データをデータベースに生成する関数
3: """
4: def create_company_data(session, company_list):
5:     for company_info in company_list:
6:         time.sleep(2)
7:         print("企業データを生成しています", ":", company_info[1])
```

```
8:         new_company = Company(
9:             edinet_code=company_info[0],
10:            name=company_info[1],
11:            OperatingProfitLoss=int(company_info[2]) if company_info[2] else
0
12:        )
13:        session.add(new_company)
14:        session.commit()
```

8章で使用したプログラムを使用し、1社ごとにレコード（行）のデータをSQLiteに新しく追加します。企業データは辞書型で保持されているため、データひとつごとに抜き出し変数に格納してから、session.add()を使用してデータ追加を行います。最後にsession.commit()をしなければ、追加したものが確定されずしっかりと保存できないので、忘れないように注意してください。

リスト 10.4: XBRL ファイルを解析し、データベースに生成する関数

```
1: """
2: メイン処理: XBRLファイルを解析し、データベースに生成する関数
3: """
4: def main():
5:     session = setup_database()
6:     xbrl_files = glob.glob(r"XBRL_files\*\XBRL\PublicDoc\*.xbrl")
7:     edinet_company_info_list = make_edinet_company_info_list(xbrl_files)
8:     insert_company_data(session, edinet_company_info_list)
9:     session.close()
```

今まで作成してきた関数を順番に実行し、データベースの作成からデータの登録まで行います。また、最後に作成したセッションをsession.close()を使用し、SQLiteとの接続を完全に切断します。このコードを実行しなければSQLite側でプログラムとの接続がされたままの判定になってしまい、接続数の上限に達した場合、データベースへの接続が行えなくなってしまう可能性があります。また、その他にも多くの問題が発生する可能性があるため、注意してください。

10.4.4 ソースコード

リスト 10.5: ソースコード

```
1: from sqlalchemy import create_engine, Column, Integer, String
2: from sqlalchemy.ext.declarative import declarative_base
3: from sqlalchemy.orm import sessionmaker
4: from arelle import ModelManager
5: from arelle import Cntlr
6: import os, time, glob
7:
```

```
8: """
9: データベースのモデル定義と基本設定
10: """
11: DATABASE_URL = "sqlite:///company_data.db"
12: Base = declarative_base()
13:
14: class Company(Base):
15:     __tablename__ = 'company'
16:     edinet_code = Column(String, primary_key=True)
17:     name = Column(String, nullable=False)
18:     OperatingProfitLoss = Column(Integer, nullable=False)
19:
20: """
21: データベースのエンジンとセッションをセットアップする関数
22: """
23: def setup_database():
24:     engine = create_engine(DATABASE_URL)
25:     Base.metadata.create_all(engine)
26:     Session = sessionmaker(bind=engine)
27:     return Session()
28:
29: """
30: 8章で使用した、10社の企業情報を抽出する関数
31: """
32: def make_edinet_company_info_list(xbrl_files):
33:     edinet_company_info_list = []
34:     for index, xbrl_file in enumerate(xbrl_files):
35:         company_data = {"EDINETCODE": "", "企業名": "", "営業利益(IFRS)": ""}
36:
37:         ctrl = Cntlr.Cntlr()
38:         model_manager = ModelManager.initialize(ctrl)
39:         model_xbrl = model_manager.load(xbrl_file)
40:         print("XBRLファイルを読み込んでいます", ":", index + 1, "/",
len(xbrl_files))
41:
42:         for fact in model_xbrl.facts:
43:             if fact.concept.qname.localName == 'EDINETCodeDEI':
44:                 company_data["EDINETCODE"] = fact.value
45:             elif fact.concept.qname.localName == 'FilerNameInJapaneseDEI':
46:                 company_data["企業名"] = fact.value
47:             elif fact.concept.qname.localName == 'OperatingProfitLossIFRS'
```

```
and fact.contextID == 'CurrentYearDuration':
48:         company_data["営業利益(IFRS)"] = fact.value
49:
50:     edinet_company_info_list.append([
51:         company_data["EDINETCODE"],
52:         company_data["企業名"],
53:         company_data["営業利益(IFRS)"],
54:     ])
55:     print(edinet_company_info_list)
56: return edinet_company_info_list
57:
58: """
59: 企業データをデータベースに生成する関数
60: """
61: def create_company_data(session, company_list):
62:     for company_info in company_list:
63:         time.sleep(2)
64:         print("企業データを生成しています", ":", company_info[1])
65:         new_company = Company(
66:             edinet_code=company_info[0],
67:             name=company_info[1],
68:             OperatingProfitLoss=int(company_info[2]) if company_info[2] else
0
69:         )
70:         session.add(new_company)
71:         session.commit()
72:
73: """
74: メイン処理: XBRLファイルを解析し、データベースに生成する関数
75: """
76: def main():
77:     session = setup_database()
78:     xbrl_files = glob.glob(r'*\フォルダーネーム*\*\*\XBRL\PublicDoc\*.xbrl')
79:     edinet_company_info_list = make_edinet_company_info_list(xbrl_files)
80:     insert_company_data(session, edinet_company_info_list)
81:     session.close()
82:
83: if __name__ == "__main__":
84:     main()
```

10.4.5 成果物

最終的には以下のようなデータベースが完成します。

今後は自身で必要な数だけ、レコードやカラムを調節して活用してください。

図 10.1: データベースの中身を確認

Rows: 10			
	edinet...	name	Operating...
	Filter	Filter...	Filter
1	E00394	アサヒグループホールディングス株式会社	244999000000
2	E00395	キリンホールディングス株式会社	150294000000
3	E00393	サッポロホールディングス株式会社	11820000000
4	E05304	日鉄ソリューションズ株式会社	35001000000
5	E00873	東レ株式会社	57651000000
6	E05460	株式会社ディー・エヌ・エー	-28270000000
7	E00435	キッコーマン株式会社	66733000000
8	E02144	トヨタ自動車株式会社	5352934000000
9	E01772	パナソニックホールディングス株式会社	360962000000
10	E01777	ソニーグループ株式会社	1208831000000

なお、SQLiteのデータを確認するには、SQLite Viewerという拡張機能を使用すると、エディターからデータを閲覧できます。

データベースが完成したら、それからデータを抜き出すことも可能です。そして、データベースの基本操作としてデータを「生成」「読み込み」「更新」「削除」する4つの操作が存在します。今回はすべての操作方法においてデータベースと接続するため、sessionを使用してこれらの操作を行います。

データの生成は先ほど説明したため、その他3つのやり方について説明します。以下の操作をしたい場合、上記のソースコード内の「if __name__ == "__main__":」よりも上に関数を追記することで、任意の操作が可能となります。また操作を行う際は、「if __name__ == "__main__":」内のmain()を使用したい関数に書き換えてご活用ください。ある程度は手動での操作となりますが、自分のDBを作成・操作できます。

10.4.6 データの読み込み

リスト 10.6: データの読み込み

```
1: """
2: データを読み込む関数
3: """
4: def read_company():
```

```

5:     session = setup_database()
6:     # テーブルに存在するデータをすべて取得
7:     companies = session.query(Company).all()
8:     for company in companies:
9:         print(f"EDINET Code: {company.edinet_code}, Name: {company.name},
Operating Profit Loss: {company.OperatingProfitLoss}")
10:    session.close()
11:
12:    .
13:    .
14:    (省略)
15:    .
16:    .
17:    if __name__ == "__main__":
18:        # main()から読み込むためのread_company()に変更する
19:        read_company()

```

`read_company()`は「`session.query(Company).all()`」と記述することで、`Company`テーブルすべての企業情報を読み込むことができます。そのため、「`if __name__ == "__main__":`」内に記述される`read_company()`の引数には何も入れなくて大丈夫です。

また、「`session.query(Company).filter(Company.edinet_code=1)`」と`filter()`を使用することで、任意の情報を絞り込み読み込むことも可能です。

読み込んだ情報は“[[1][株式会社○○][100]]”のように、リストの形で取り出されます。取り出したデータはそのまま利用することができますが、データ型がSQLAlchemy独自の型となっているため、注意してください。

10.4.7 データの更新

リスト 10.7: データの更新

```

1: """
2: データを更新する関数
3: """
4: def update_company(edinet_code, new_name=None, new_profit_loss=None):
5:     session = setup_database()
6:     # edinet_codeに任意のEDINETコードを指定することでその行だけ更新
7:     company = session.query(Company).filter(Company.edinet_code ==
edinet_code).first()
8:     if company:
9:         if new_name:
10:             company.name = new_name

```

```

11:         if new_profit_loss is not None:
12:             company.OperatingProfitLoss = new_profit_loss
13:             session.commit()
14:             print(f"データを更新しました: {company.edinet_code}")
15:     else:
16:         print("指定された企業が見つかりません")
17:     session.close()
18:
19:     .
20:     .
21:             (省略)
22:             .
23:             .
24:         if __name__ == "__main__":
25:             # main()から更新するためのupdate_company()に変更する
26:             update_company("E0034", "アサヒグループ", 150000000)

```

update_company()はデータベースにあるひとつレコードの情報を読み込むことで、その内容を更新することができます。そのため、「if __name__ == "__main__":」内に記述されるupdate_company()の引数にはひとつ目に「更新する企業のEDINETCODE」ふたつ目の引数には「新しく更新する企業名」3つ目の引数には「新しく更新する営業利益」を記述してください。

このコードでは読み込んだレコードをcompanyという名前の変数に入れ、「company.name」と書くことによってcompanyテーブルのnameカラムを指定します。company.name = new_nameと新たな値を格納することにより、データの更新を行います。

また、new_nameとnew_profit_lossとは更新時に新しく格納することであり、if文で条件分岐を作成することでどちらか片方しか更新する値がなくても更新される設計となっています。

10.4.8 データの削除

リスト 10.8: データの削除

```

1: """
2: データを削除する関数
3: """
4: def delete_company(edinet_code):
5:     session = setup_database()
6:     # edinet_codeに任意のEDINETコードを指定することでその行だけ削除
7:     company = session.query(Company).filter(Company.edinet_code ==
edinet_code).first()
8:     if company:
9:         session.delete(company)

```

```

10:     session.commit()
11:     print(f"データを削除しました: {company.edinet_code}")
12: else:
13:     print("削除する企業が見つかりません")
14: session.close()
15:
16:     .
17:     .
18:             (省略)
19:     .
20:     .
21: if __name__ == "__main__":
22:     # main() から削除するためのdelete_company()に変更する
23:     delete_company("E0034")

```

`delete_company()`は企業一社のデータを読み込み、それらのデータを削除することができます。そのため、「`if __name__ == "__main__":`」内に記述される`delete_company()`の引数には、「削除する企業のEDINETCODE」を記述してください。

初めにデータの読み込みを行うため、データを`filter()`を使用しデータを絞り込み、`first()`でひとつのレコードに絞り込んでいます。絞り込んだデータを`company`という変数に入れ、「`session.delete()`」の引数に`company`を入れることによりデータを削除します。

10.5 まとめ

本章では、小規模なデータベースの作成について解説しました。自分でデータベースを持つことで、必要なデータをいつでも自由に使用することができるようになります。本章では、一期10社分のデータを使用したため、EDINETコードだけでデータを一意に絞ることができました。しかし、さらに大規模なDBとして保有したい場合は提出日などと結びつけてデータを一意に絞るなど、工夫が必要になってきます。XBRLの活用方法は様々存在しますが、データベースを最大限活用し、より効率的なXBRL解析を行ってください。

次章では、XBRL解析をより自分好みに行うことができるようするために必要な独自のパーサーを開発する方法を紹介します。

第11章 独自のパーサーを作成する（新規）

11.1 本章の目的

本書では、ここまでパーサーにArelleを使用してきました。ですが解析の目的によってはかゆいところに手が届かない場合があったり、公開されているパーサーをうまく使いこなせないケースがあります。そのため、パーサーを自作し使いやすくしている人も存在します。

本章ではとりあえずパーサーを自作し、学びを得ることを目的とします。

11.2 Arelleと独自パーサーの違い

Arelleのように公開されているパーサーは様々な機能が提供されています。しかし、情報が少ないとため使いこなすには実際にコードを見て学ぶことが必要となり、学習コストが高く感じます。そういう場合、機能を絞った小規模なパーサーを自身で作成する方もいます。パーサーを自作することで、自身に合わせた機能を柔軟に分析に役立てることができます。

11.3 使用される技術

XML解析には、その複雑な情報をツリー構造で表現し読み込むモデルが使用されています。

Arelleでは、このモデルの適用として読み込みにlxmlというPythonのライブラリーを使用しています。

今回は小規模なパーサーを作成するため、lxmlとともに操作性を重視しているBeautifulSoupというライブラリーを使用します。BeautifulSoupではライブラリー内においてツリー構造で処理をしており、その技術を用いて判別した情報を文字列や数値で出力しています。その文字列や数値はlxmlのように複雑な関係が付与されたデータではないため、注意が必要です。

11.4 独自パーサーを作る

今回はパーサーをライブラリーとして実装します。ライブラリーとは、8章コラムでも紹介したように事前に作成した関数をファイルとして保存し、その関数を別ファイルで呼び出して利用することにより効率的にプログラミングを行うための仕組みです。

本章では学習目的のため、ファイルひとつ・関数3つという必要最低限の機能のみを備えたパーサーを作成します。

11.4.1 パーサーの機能要件

今回は以下の機能のパーサーを作成します。

- ・タクソノミを指定すると、そのタクソノミのインスタンス情報が取得できる
- ・タクソノミを指定する際に同時にcontext_ref等の情報で条件を絞るようにする
- ・データクレンジング処理をデフォルトする
- ・よく使用する情報を一括で取得できる

11.4.2 機能の解説

では、先ほどの機能要件を満たすようなライブラリーを試しに作成してみましょう。
ここでは3つの機能に分けて実装し、ひとつずつ解説していきます。

11.4.2.1 1. fact を取得する機能

「XBRLファイルのpath」「タクソノミ」「context_ref」「unit_ref」の4つの情報を使用し、特定のタクソノミに合致するfactを取得できるようにします。
この機能で機能要件のひとつ目とふたつ目を満たすことができます。

リスト 11.1: タグを指定して抽出を行う関数

```

1: """
2: タグを指定して抽出を行う関数
3: """
4: def get_fact(xbrl_file:str,tag:str,context_ref:str=None,unit_ref:str=None):
5:     with open(xbrl_file, encoding="utf-8") as doc:
6:         soup = BeautifulSoup(doc, "lxml-xml")
7:
8:         conditions = {}
9:         if context_ref:
10:             conditions["contextRef"] = context_ref
11:         if unit_ref:
12:             conditions["unitRef"] = unit_ref
13:
14:         # 実際にXBRLから条件に合うインスタンスを探す
15:         fact = soup.find(tag, conditions)
16:         fact = fact.text
17:
18:     return fact

```

ここでは、Pythonの標準ライブラリーであるopen関数でインスタンスファイルの中身を読み取り、Beautiful Soupに読み込ませています。

また、特定のタグ条件に合わせてインスタンスを取得するには、Beautiful Soupのfindメソッドを使用することで対処することができます。context_refとunit_refをconditionsという名前の辞書型変数に入れてfindメソッドに読み込ませることで、条件に合ったfactを取得できます。

11.4.2.2 2. データクレンジング処理機能

テキストデータのインスタンスを取得した際に、必ずついているHTMLタグや余計な空白等の除去を行います。

テキスト分析を行う際や企業ごとにテキストデータを比較したい場合など、テキストデータを扱う際はHTMLタグが不要なことが多くあり、これは何度も多用するクレンジング処理なため、機能として実装します。

リスト11.2: python

```
1: """
2: HTMLタグを除去する関数
3: """
4: def htmltag_remove(text):
5:     # 空白・改行 (\s) の除去
6:     text = re.sub(r'\s+', '', text).strip()
7:     # HTMLタグ (<.*?>) の除去
8:     soup = BeautifulSoup(text, "html.parser")
9:     cleansing_text = soup.get_text()
10:
11:     return cleansing_text
```

今回の除去は、8章で使用したデータクレンジングのコードをそのまま流用しました。具体的に除去したのは3つです。

- ・空白
- ・改行
- ・HTMLタグ

プログラムにおいて空白や改行を含む空白文字全般は「」と表されるため、何もない” ”の状態に置き換えることでそれらを除去しています。また、「<.*?>」では<>とその中身全てを” ”に置き換えることで除去しています。

この3つを除去することで、それぞれの文章がすべて1列でつながっているテキストデータを取得することができます。

図11.1: データクレンジングを行ったテキスト

3【事業等のリスク】以下において、トヨタの事業その他のリスクについて、投資家の判断に重要な影響を及ぼす可能性のある事項を記載しています。ただし、以下はトヨタに関するすべてのリスクを網羅したものではなく、記載されたリスク以外のリスクも存在します。かかるリスク要因のいずれによっても、投資家の判断に影響を及ぼす可能性があります。本項においては、将来に関する事項が含まれていますが、当該事項は有価証券報告書提出日（2024年6月25日）現在において判断したものです。（1）市場および事業に関するリスク①自動車市場の競争激化世界の自動車市場では激しい競争が繰り広げられています。トヨタは、ビジネスを展開している各々の地域で、自動車メーカーとの競争に直面しています。近年、自動車市場における競争はさらに激化しており、厳しい状況が続いています。また、世界の自動車産業におけるCAS

11.4.2.3 3. 自分のよく使用するタクソノミのインスタンスを一括で取得する機能

この機能では事前に自分がよく使用するタクソノミ、本章では例として「企業名」「会計基準」「事業等のリスク」の3つをそれぞれ指定しておくことで、インスタンスを簡単に取得できるようにします。

リスト 11.3: よく使用するタクソノミのインスタンスをゲットする

```
1: """
2: よく使用するタクソノミのインスタンスをゲットする
3: """
4: def use_well_taxonomy(xbrl_file:str):
5:     with open(xbrl_file, encoding="utf-8") as doc:
6:         soup = BeautifulSoup(doc, "lxml-xml")
7:
8:     tags = ["FilerNameInJapaneseDEI", "AccountingStandardsDEI"
9:             , "BusinessRisksTextBlock"]
10:
11:    fact_list = []
12:    for tag in tags:
13:        fact = soup.find(tag)
14:        fact = fact.text
15:        fact = HTMLtag_remove(fact)
16:        fact_list.append(fact)
17:
18:    return fact_list
```

タクソノミをリストで保持し、for文で順番に取り出し、Beautiful Soupから取得しています。テキストデータである事業等のリスクは分析時に活用しやすいように、先ほどのデータクレンジングをデフォルトで行い出力するような処理にしています。

また今回は、対象がどの企業でもインスタンスが取得できるよう、以下のタクソノミを選出しました。

- ・企業名
- ・会計基準
- ・事業等のリスク

タクソノミは1章でも説明したように会計基準ごとに分かれています。勘定科目のタクソノミは業種ごとに分かれています。

情報を取得する対象企業が「IFRS基準を採用している企業のみ」や「建設業の企業のみ」などのように絞ることができればそれに応じたタクソノミを指定することができ、より自分が使いやすいパーサーにすることができます。

11.4.3 ソースコード

リスト 11.4: parser.py

```
1: # parser.py
2: from bs4 import BeautifulSoup
3: import re
4:
5: """
6: タグを指定して抽出を行う関数
7: """
8: def get_fact(xbrl_file:str,tag:str,context_ref:str=None,unit_ref:str=None):
9:     with open(xbrl_file, encoding="utf-8") as doc:
10:         soup = BeautifulSoup(doc, "lxml-xml")
11:
12:     conditions = {}
13:     if context_ref:
14:         conditions["contextRef"] = context_ref
15:     if unit_ref:
16:         conditions["unitRef"] = unit_ref
17:
18:     # 実際にXBRLから条件に合うインスタンスを探す
19:     fact = soup.find(tag, conditions)
20:     fact = fact.text
21:
22:     return fact
23:
24: """
25: htmlタグを除去する関数
26: """
27: def htmltag_remove(text):
28:     # 空白 (\s) 除去
29:     text = re.sub(r'\s', '', text).strip()
30:     # HTMLタグ (<.*?>) の除去
31:     soup = BeautifulSoup(text, "html.parser")
32:     cleansing_text = soup.get_text()
33:
34:     return cleansing_text
35:
36: """
37: よく使用するタクソノミのインスタンスをゲットする
38: """
```

```
39: def use_well_taxonomy(xbrl_file):
40:     with open(xbrl_file, encoding="utf-8") as doc:
41:         soup = BeautifulSoup(doc, "lxml-xml")
42:
43:     tags = ["FilerNameInJapaneseDEI", "AccountingStandardsDEI"
44:             , "BusinessRisksTextBlock"]
45:
46:     instance_list = []
47:     for tag in tags:
48:         instance = soup.find(tag)
49:         instance = instance.text
50:         instance = htmltag_remove(instance)
51:         instance_list.append(instance)
52:
53:     return instance_list
```

11.4.4 ライブラリーの実装方法

作成したパーサーを手軽に使う方法として、パーサーのファイルを同じフォルダー内に置き、import文を使って呼び出す手段があります。しかし、この方法ではライブラリーを毎回コピーしたり移動させなければならないなど手間がかかり、利用しやすいとは言えません。

そこで本書では、pipコマンドやcondaコマンドでインストールしたライブラリーと同様に、自作ライブラリーをインストールして使えるようにする方法を紹介します。この方法を用いることで、パーサーをライブラリー化し、より使いやすくすることができます。

新しく「setuptools」というライブラリーを使用します。setuptoolsはPythonプログラムをライブラリーとして配布することができるツールです。

以下の手順で進めていきます。

1. setup.pyを作成
2. フォルダーを作成し、その中に__init__.pyと作成したライブラリーファイルを作成
3. コマンドを使用しターミナルでインストール

また、このような手順でライブラリーをインストールする分には世界中に公開などされないので、安心してください。

11.4.4.1 1. setup.pyを作成

リスト11.5: setup.py

```
1: # setup.py
2: from setuptools import setup, find_packages
3:
4: setup(
5:     name="xbrl_parser",
```

```
6:     version="1.0.0",
7:     author="ontology",
8:     packages=find_packages(),
9:     install_requires=[
10:         "beautifulsoup4>=4.9.0",
11:         "lxml>=4.6.0",
12:     ],
13: )
```

setup.pyでは「ライブラリの名前」や「配布対象にするモジュールを指定する」など細かいオプションを設定します。今回はできる限り複雑なオプションは設定せず、以下のようなオプション設定です。

- ・ライブラリ名はxbrl_paser
- ・バージョンを1.0.0とする
- ・作成者名はontology
- ・パッケージはfind_packages()でファイル内から自動読み込み
- ・このライブラリーをインストールした際、ふたつのライブラリーを同時にインストールする

11.4.4.2 2. 新しくフォルダーを作成

先ほどの作成したparser.pyと、この後作成する__init__.pyをまとめて入れるフォルダーを作成します。インストール時にはこのフォルダーごと取り込まれ、フォルダー内のファイルをモジュールとして参照できるようになります。また、ファイル名は自分の考えたもので大丈夫です。今回は「xb_parser」というフォルダーナ名にします。

parser.pyをこのフォルダーの中に入れて下さい。

11.4.4.3 __init__.pyを作成

リスト 11.6: __init__.py

```
1: # __init__.py
2: # 作成したフォルダーにparser.pyと__init__.pyを入れる
3: from xb_parser.parser import (
4:     get_fact,
5:     HTMLtag_remove,
6:     use_well_taxonomy
7: )
8:
9: __all__ = [
10:     "get_fact",
11:     "HTMLtag_remove",
12:     "use_well_taxonomy"
13: ]
```

`__init__.py`では、ライブラリーの初期化を行う設定やライブラリー内のモジュールの制御を行うことができます。今回はimport文を記述した際にモジュールからどの関数がimportされるかなど、import周りの整理を主に行っていきます。

2で作成したフォルダーの中に作成して下さい。

11.4.4.4 4. コマンドを使用しターミナルでインストール

ターミナルを起動し「`pip install`」と打ち込み実行してください。

「Successfully installed xbrl_parser-1.0.0」と表示されれば、インストールは完了です。

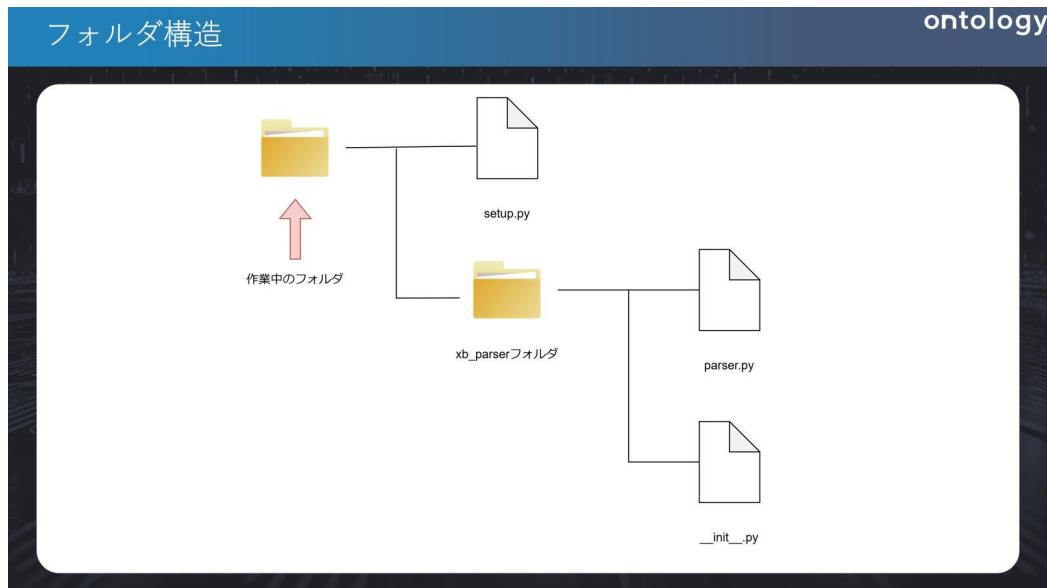
自作ライブラリーを使用したいPythonファイルに、「`from フォルダ名 import 関数名`」と入力することで使用することができます。

▼Pythonでの使用例

リスト11.7: Pythonでの使用例

```
1: from xb_parser import get_fact,HTMLtag_remove,use_well_taxonomy
2:
3: xbrl_file = r"\xxx\\ファイル名\XBRL\PublicDoc\XBRLファイル名.xbrl"
4: tag = "BusinessRisksTextBlock"
5: context_ref = "FilingDateInstant"
6:
7: #XBRLファイルと任意のタクソノミをセット
8: fact = get_fact(xbrl_file,tag,context_ref)
9:
10: #HTMLタグについているデータをセット
11: text = HTMLtag_remove(fact)
12:
13: #対象のXBRLファイルをセット
14: well_fact = use_well_taxonomy(xbrl_file)
```

図 11.2: フォルダ構造



なお、「`pip install .`」を実行する際は、「`xb_parser フォルダ`」のひとつ上の階層のディレクトリーをカレントディレクトリー（作業中のフォルダー）にする必要があるため、注意してください。

11.5 Arelleのようなライブラリーを作るには

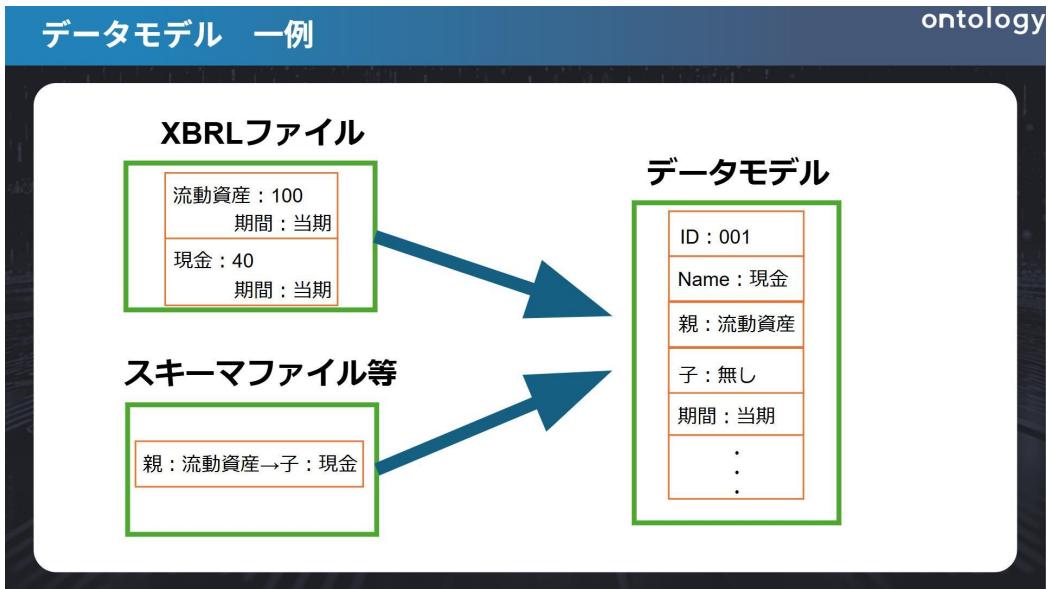
ここまでで小規模なパーサーを作って使用することについて解説しましたが、本当に自分で使用するとなると、汎用的な機能を備えたパーサーが欲しくなると思います。Arelelのような汎用的なライブラリーを作成するには、データモデルを導入することが必要です。

そこで一般公開されている Arelleなどのライブラリーでは、データモデルを導入しプログラムをより見やすく操作しやすい形に簡略化することで、多くの機能の実装を実現しています。

11.5.1 データモデルとは

データモデルとはどのように使用されているのでしょうか？今回は Arelleの一部コードを参照しながら、システムの大まかな形のみ説明をしていきます。

図 11.3: データモデル一例



データモデルとは、事前に必要な値が入る変数を複数用意しておくことで、複雑なデータ構造を統一できるものです。解析のはじめにXBRL内すべてのデータをデータモデルに基づいた形で一度読み込むことで、その後のデータ処理を簡略化することができます。

図 11.4: Arelle のデータモデル一例

```
def init(self, keepViews: bool = False, errorCaptureLevel: str | None = None) -> None:
    self.uuid: str = uuid.uuid1().urn
    self.namespaceDocs: defaultdict[str, list[ModelDocumentClass]] = defaultdict(list)
    self.urlDocs: dict[str, ModelDocumentClass] = {}
    self.urlUnloadableDocs: dict[bool, str] = {} # if entry is True, entry is blocked at
    self.errorCaptureLevel: str = (errorCaptureLevel or logging._checkLevel("INCONSISTENCY"))
    self.errors: list[str | None] = []
    self.logCount: dict[str, int] = {}
    self.arcroleTypes: defaultdict[str, list[ModelRoleType]] = defaultdict(list)
    self.roleTypes: defaultdict[str, list[ModelRoleType]] = defaultdict(list)
    self.qnameConcepts: dict[QName, ModelConcept] = {} # indexed by qname of element
    self.nameConcepts: defaultdict[str, list[ModelConcept]] = defaultdict(list) # contains
    self.qnameAttributes: dict[QName, Any] = {}
```

Arelleでは「インスタンス情報を格納する変数」や「タクソノミ間の関係性を保持する変数」など、他にも多くの変数が画像のようにデータモデルとして定義されています。

このようにインスタンスファイルに載っていないようなスキーマファイルや、そこから判別できる状況などの情報も付与することで、実装できる機能の幅を広げることができます。

これでも一部を抜粋したものにすぎません。Arelleではこれだけの数の情報をひとつひとつのインスタンスに対して付与することで、様々な充実した機能の実装を行っています。

11.6 まとめ

本章では、必要最低限の小規模なパーサーの作成例と汎用的なパーサーを作成するときに必要なデータモデルの紹介をしました。プログラムを工夫することで、小規模なものでも機能内容を自分に合った唯一無二のものにすることで、より快適にXBRLを扱うことができます。

また、データモデルを使用することにより、複雑な関係性を加味してプログラムで解析することが可能になります。XBRLに慣れてきたらそれらの技術にも触れ、より汎用的なパーサーの作成も推奨します。

NextPublishing Sample

著者紹介

○○ ○○ (○○ ○○)

○

○○ ○○ (○○ ○○)

○

◎本書スタッフ

アートディレクター/装丁：岡田章志+GY

編集協力：■■■■

ディレクター：栗原 翔

（表紙イラスト）

■■■■

技術の泉シリーズ・刊行によせて

技術者の知見のアウトプットである技術同人誌は、急速に認知度を高めています。インプレス NextPublishingは国内最大級の即売会「技術書典」(<https://techbookfest.org/>)で発表された技術同人誌を底本とした商業書籍を2016年より刊行し、これらを中心とした『技術書典シリーズ』を開催してきました。2019年4月、より幅広い技術同人誌を対象とし、最新の知見を発信するため『技術の泉シリーズ』へリニューアルしました。今後は「技術書典」をはじめとした各種即売会や、勉強会・LT会などで発表された技術同人誌を底本とした商業書籍を刊行し、技術同人誌の普及と発展に貢献することを目指します。エンジニアの“知の結晶”である技術同人誌の世界に、より多くの方が触れていただくきっかけになれば幸いです。

インプレス NextPublishing
技術の泉シリーズ 編集長 山城 敬

●お断り

掲載したURLは202■年■月1日現在のものです。サイトの都合で変更されることがあります。また、電子版ではURLにハイパーリンクを設定していますが、端末やビューアー、リンク先のファイルタイプによっては表示されないことがあります。あらかじめご了承ください。

●本書の内容についてのお問い合わせ先

株式会社インプレス

インプレス NextPublishing メール窓口

np-info@impress.co.jp

お問い合わせの際は、書名、ISBN、お名前、お電話番号、メールアドレスに加えて、「該当するページ」と「具体的なご質問内容」「お使いの動作環境」を必ず明記ください。なお、本書の範囲を超えるご質問にはお答えできないのでご了承ください。

電話やFAXでのご質問には対応しておりません。また、封書でのお問い合わせは回答までに日数をいただく場合があります。あらかじめご了承ください。

奥付サンプル

201X年X月X日 初版発行Ver.1.0 (PDF版)

著 者 × × × ×
編集人 × × × ×
発行人 × × × ×
発 行 株式会社インプレスR&D

〒101-0051
東京都千代田区神田神保町一丁目105番地
<http://nextpublishing.jp/>

●本書は著作権法上の保護を受けています。本書の一部あるいは全部について株式会社インプレスR&Dから文書による許諾を得ずに、いかなる方法においても無断で複写、複製することは禁じられています。

©201X, All rights reserved.

ISBN978-4-XXXX-XXXX-X



Next Publishing®

●インプレス Next Publishingは、株式会社インプレスR&Dが開発したデジタルファースト型の出版モデルを承継し、幅広い出版企画を電子書籍+オンデマンドによりスピーディで持続可能な形で実現しています。<https://nextpublishing.jp/>

NextPublishing Sample