課長のせいで今日も死んでます。
楽して覚える情報技術

［著］株式会社おもしろテクノロジー

技術書典 18（2025 年夏）新刊
2025 年 5 月 30 日　 ver 1.0

■免責

本書は情報の提供のみを目的としています。
本書の内容を実行・適用・運用したことで何が起きようとも、それは実行・適用・運用した人自身の責任であり、著者や関係者はいかなる責任も負いません。

■商標

本書に登場するシステム名や製品名は、関係各社の商標または登録商標です。
また本書では、™、Ⓡ、Ⓒ などのマークは省略しています。

まえがき / はじめに

本書を手に取っていただき、ありがとうございます。

「課長のせいで今日も死んでます〜無理して覚える IT(情報技術)〜」は、様々な IT 知識についてマンガも交えながらわかりやすく解説した本です。この本を読めば、取り扱う技術の概要がざっくりわかるようになっています。

本書の目的

本書の目的は、様々な IT 技術の概要を学ぶことです。概要を学ぶことで、実際の業務で活用するためのきっかけを得ることができます。本書では、実際の技術の使われ方、メリット・デメリット、関連知識などを解説しています。

ただし、詳細については深く説明をしていませんので、業務で活用する際は、必ず他の資料や書籍を参照してください。なお、可能な限り公的な情報や参考元となった文献も記載していますので、ご活用ください。

マンガを提出しても上司は首を縦に振ってはくれませんが、公的な文章であれば、導入を検討してもらえるかもしれません。

本書の対象読者

本書では次のような人を対象としています。

- IT 知識について興味がある人
- 自分の専門外の分野にも興味がある人
- 専門書はちょっと苦手だけど、マンガだったら何とかなるような気がする人

前提とする知識

本書を読むにあたり、次のような知識が必要となります。

- IT に関する基礎知識
  - 全てを初心者向けに分かる表現にしないことで、既存の初心者向けマンガでは取り扱うことができなかった題材も取り扱っています。
  - 一方で、初学者の方が楽しめないかというとそうではなく、初心者向けの内容から中上級者向けの内容まで幅広く取り扱っています。また、分からない内容があればぜひ調べてみてください。
- 平成のアニメ・マンガ・インターネット関連の知識
  - 本書にはそのあたりのパロディが多く含まれています。
  - パロディが苦手な方はごめんなさい。

問い合わせ先

本書に関する質問やお問い合わせは、以下のアカウントまでお願いします。

- URL: `https://x.com/Norotororo`

謝辞

本書の表紙・マンガはゆず胡椒氏（@citrong_king）に描いていただきました。いつも遅れて原稿をお渡しして作業をお願いしている中で、文句も言わず素晴らしいクオリティの作品を描いていただき、感謝しています。

また、本書は株式会社おもしろテクノロジーの社員にレビューしていただきました。この場を借りて感謝します。ありがとうございました。

凡例

本書では、プログラムコードを次のように表示します。太字は強調を表します。

```
print("Hello, world!\n");          ←太字は強調
```

プログラムコードの差分を表示する場合は、追加されたコードを太字で、削除されたコードを取り消し線で表します。

```
~~print("Hello, world!\n");~~          ←取り消し線は削除したコード
print("Hello, "+name+"!\n");       ←太字は追加したコード
```

長い行が右端で折り返されると、折り返されたことを表す小さな記号がつきます。

```
123456789_123456789_123456789_123456789_123456789_123456789_123456789_123›
›456789_123456789_
```

ターミナル画面は、次のように表示します。行頭の「$ 」はプロンプトを表し、ユーザが入力するコマンドには薄い下線を引いています。

```
$ echo Hello          ←行頭の「$ 」はプロンプト、それ以降がユーザ入力
```

本文に対する補足情報や注意・警告は、次のようなノートや囲み枠で表示します。

ノートタイトル

ノートは本文に対する補足情報です。

タイトル

本文に対する補足情報です。

タイトル

本文に対する注意・警告です。

目次

第 1 章
クラウド

第 1 章では「クラウドサービス」を題材にしています。。普段から使い慣れている人にとっては当たり前のものですが、まだ触れたことのない人には未知の世界かもしれません。

今回の解説では、各社が提供しているコンピューティングやストレージなどの具体的なサービス内容には深入りせず、実際の PC を活用する場合と比較した際の差分を中心に説明していきます。

1.1 あらすじ

新入社員として IT 企業に就職したりんねは、社内で偶然見かけた小さな女の子（=課長）の所属する 4 課に配属される。課長は、AI の開発環境の構築を依頼されており、どうやら自分で PC を組み立てて実現をするらしい。しかし、PC を組み立て終わる際に、PC の電力供給が大きすぎることが原因で、発火してしまい、偶然近くにあったガスボンベが爆発してしまう。りんねは、死んでしまうが、天界で神さまにであい、クラウドサービスの知識を授けられる。神さまはりんねを生き返らせるが、無事にクラウドサービスを活用することができるのか？

「計算リソースを柔軟に調達できるのがクラウドサービスの強みね！ 」

「なるほど、クラウドクラウドってみんな言ってたけど、こういうことだったんだ。」

「計算リソースだけじゃなく、各社いろんな機能を持ったサービスを提供してるから、興味があったらしらべてみてもいいわね。」

「わかりました！ 」

1.2　クラウドの定義

NIST（National Institute of Standards and Technology）では、クラウドコンピューティングを次のように定義しています。

オンデマンド・セルフサービス (On-demand self-service)

ユーザは、各サービスの提供者と直接やりとりすることなく、必要に応じ、自動的に、サーバーの稼働時間やネットワークストレージのようなコンピューティング能力を一方的に設定できる。

幅広いネットワークアクセス (Broad network access)

コンピューティング能力は、ネットワークを通じて利用可能で、標準的な仕組みで接続可能であり、そのことにより、様々なシンおよびシッククライアントプラットフォーム（例えばモバイルフォン、タブレット、ラップトップコンピュータ、ワークステーション）からの利用を可能とする。

リソースの共用化 (Resource pooling)

サービスの提供者のコンピューティングリソースは集積され、複数のユーザにマルチテナントモデルを利用して提供される。様々な物理的・仮想的リソースは、ユーザの需要に応じてダイナミックに割り当てられたり再割り当てされたりする。物理的な所在場所に制約されないという考え方で、ユーザは一般的に、提供されるリソースの正確な所在地を知ったりコントロールしたりできないが、場合によってはより抽象的なレベル（例：国、州、データセンタ）で特定可能である。リソースの例としては、ストレージ、処理能力、メモリ、およびネットワーク帯域が挙げられる。

スピーディな拡張性 (Rapid elasticity)

コンピューティング能力は、伸縮自在に、場合によっては自動で割当ておよび提供が可能で、需要に応じて即座にスケールアウト／スケールインできる。ユーザにとっては、多くの場合、割当てのために利用可能な能力は無尽蔵で、いつでもどんな量でも調達可能のように見える。

サービスが計測可能であること (Measured service)

クラウドシステムは、計測能力を利用して、サービスの種類（ストレージ、処理能力、帯域、実利用中のユーザアカウント数）に適した管理レベルでリソースの利用をコントロールし最適化する。リソースの利用状況はモニタされ、コントロールされ、報告される。それにより、サービスの利用結果がユーザにもサービス提供者にも明示できる。

原文：
`https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf`

IPA による和訳：`https://www.ipa.go.jp/security/reports/oversea/nist/ug65p900`

00019cp4-att/begoj9000000buyd.pdf

個人的には、以下の定義のみ覚えていれば十分だと思います。

> サービスの提供者のコンピューティングリソースは集積され、複数のユーザにマルチテナントモデルを利用して提供される。様々な物理的・仮想的リソースは、ユーザの需要に応じてダイナミックに割り当てられたり再割り当てされたりする。

つまり、どこかでまとまって稼働しているサーバーを、インターネットを通じてオンデマンドで利用できるということです。

1.3　その他関連情報

クラウドの技術に関連した取り扱ったトピックについて解説します。

AI 開発にはグラフィックボード搭載の PC が必要か

A. 時と場合による

解説の都合上「AI = グラフィックボード」という説明をしましたが、必ずしも GPU が必要というわけではありません。ただし、私がこれまで出会った機械学習系の仕事をしている人は、全員 GPU を使用していました。

AI や機械学習が指し示す範囲は広く、開発環境を一つの枠組みで語ることは難しいのが現状です。ただ、近年よく使用されている大規模なモデルを使用する場合については、GPU を利用するケースが多いのではないでしょうか。

GPU 搭載 PC の消費電力について

GPU で計算をする場合、その性能が上がるにつれ消費する電力は比例します。今回イメージした構成の場合、一台あたりの消費電力が 1000W なので、合計の 14000W の消費になりそうです。

例えば、PANASONIC のコンセント (WCH2434H) の許容電力は 1500W ですから、定格を大きく超えているのが分かります。延長ケーブルを使う際には電力の計算を忘れないようにしましょう。

また、今回の題材となっている「クラウドサービス」の場合は電力の確保をどうしているのかというと、AWS の場合は、原子力発電所との交渉を進めるなど、苦難をしているようです。

引用：`https://www.itmedia.co.jp/news/articles/2307/03/news086.html`

AI 開発にはクラウドサービスが必要か

A. 会社による

今回のケースでは、開発側がクラウドを必要としていることが後から明かされましたが、必ずしもすべての企業がクラウドサービスを利用しているわけではありません。自社でサーバーを保有する（オンプレミス）ケースも存在します。基本的には大規模な GPU クラスタを作ろうとした場合、初期費用が多くかかります。規模の小さい企業の場合は、クラウドを使うことが多いのではないでしょうか。なお、このセクションでは AI とクラウドについてのみ説明していますが、クラウドは AI 以外の用途でも広く活用されています。クラウド＝ AI ではないことを念の為補足させてください。

LLM をファインチューニングするために必要な GPU の数

A. モデルサイズ・選択するアルゴリズムに依存する

GPU が必要となる代表的なユースケースとして、近年注目を集めている LLM（Large Language

Model）があります。今回の題材では [GPT-3](https://en.wikipedia.org/wiki/GPT-3) のファインチューニングを想定した数値を例示しています。必要な GPU の性能はモデルサイズによって変化し、GPT-3 の場合は以下のようになります：

PEFT（Parameter-Efficient Fine-Tuning、部分的なファインチューニング）では、約 350GB の VRAM が必要です。

完全なファインチューニングでは、約 1.2TB の VRAM が必要です。

引用：`https://arxiv.org/pdf/2106.09685`

この説明の要点は、このような大規模な GPU リソースを自前で用意する必要はなく、**クラウドサービスを利用することで時間単位での使用が可能**という点です。ちなみに、実際の LLM ファインチューニングでは、RTX4090 より大きな VRAM を搭載した GPU も使用されています。

▼表 1.1: GPU の種類と VRAM 容量・価格

GPU 名	VRAM	発売当初価格（概算）
NVIDIA A100	40GB または 80GB HBM2e	約 1000 万円〜2000 万円
NVIDIA H100	80GB HBM3 (一部 SKU は 94GB)	約 2500 万円〜3500 万円
NVIDIA GeForce RTX 4090	24GB GDDR6X	約 25 万円

「たっ、高い...! 」

「えっ、全然余裕じゃない？ 」

「 (この人お嬢様だった...) 」

「サーバーを増やすとかではなく、時代はクラウド」とは

この元ネタは、元国会議員の蓮舫氏の発言です。

<@href>{https://www.asahi.com/articles/ASQ4W6GJ8Q4WUTFK00B.html}

当時の Twitter では、「クラウドを使えばサーバーがいらなくなる」という意図と捉えられ話題となっていました。この記事によると、オンプレではなくクラウドサービスを使えという意図だったようです。

一方で、クラウドサービスの大手は海外に集中しており、クラウドを利用するたびに円が流出する「デジタル赤字」も近年問題になっています。

最近では、さくらインターネットが国内で初めてガバメントクラウド認証を取得したことで話題になりました。

企業における PC の会計上の取扱いについて

一定価格を超える PC は、時間経過とともに価値が減少するという考え方から、購入時に一括で経費として計上することができず、固定資産として扱う必要があります。

- サーバーの償却期間：5 年
- サーバー以外の PC の償却期間：4 年

https://www.nta.go.jp/taxes/shiraberu/taxanswer/shotoku/pdf/2100_01.pdf

企業によっては固定資産の棚卸のため、毎年 PC の実在確認を行う必要があります。（ので、経理の人に嫌がられます）

一方、クラウドは使用量に応じた経費計上で済むため、会計処理がシンプルです。

第 2 章

IaC (Infrastructure as Code)

第 2 章では「IaC (Infrastructure as Code)」を題材にしています。第 1 章では「クラウドサービス」を題材にしていますが、その発展形としてインフラ構成をコードで管理する = Infrastructure as Code (IaC) という考え方があります。今回の解説では、IaC を中心に、クラウド活用時のよくある誤りや運用イメージについての解説を行います。

2.1 あらすじ

ある日いつも通りにクラウドを使用していたりんねと課長だが、突然インフラ費用の高額請求が届いてしまう。どうやら、クラウド上の高額リソースを消し忘れてしまったらしい。2 人は危険なギャンブルで取り戻そうとするが、上手くいかずまたりんねは天界へ行ってしまうのだった。果たして IaC を駆使して、いい感じのインフラを構築することはできるのでしょうか？

「インフラとコードって関係あるのかな？ 」

「記法は独特ですけど、ちゃんとコードで表現できるみたいですよ！ 」

「コードを見ると確かにそれっぽい！ 」

▼ Terraform のコード例

```
provider "aws" {
  region = "ap-northeast-1"
}

resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "a" {
  vpc_id     = aws_vpc.main.id
  cidr_block = "10.0.1.0/24"
}

resource "aws_instance" "web" {
  ami           = var.ami
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.a.id
}
```

2.2 IaC について

IaC は○○年に登場した技術で、Infrastructure as Code の略です。IaC は、インフラをコードで管理することを指します。コードを実行すると、インフラが自動で構築されます。内部的にはコードを実行環境が解釈して、インフラを構築するための API を呼び出す仕組みになっています。

IaC の嬉しさ

▲図 2.1: とても長い手順書

IaC の嬉しさは、インフラの構成をコードとして管理できることです。これにより、インフラの構成や設定をプログラムとして記述し、バージョン管理や自動化が可能になります。例えば、本番環境と開発環境を同じ構成で作成したい場合、IaC を使うことで、同じコードを実行するだけで簡単に環境を再現できます。また、削除も一括で行うことができ、不要なリソースを漏れなくにクリーンアップできます。

IaC のデメリット

IaC のデメリットはどうしても手間がかかることです。IaC を使うためには、まずはコードを書く必要があり、記述方法に習熟する必要があります。また、IaC を使うことによりクラウドの利用により実現したい、迅速なリソースの立ち上げが実現できなくなる場合があります。その場合は、技術検証（PoC）の場合には、IaC を使わずに、手動でリソースを立ち上げることもあります。

「そうするとまた、リソースの消し忘れが起きちゃうんじゃ… 」

「だから、IaC だけが解決策じゃないってことね」

「どうすればいいんですか？ 」

「コストのアラートだったり、自動停止の設定だったり色々やれることはあるわね」

「なるほど、調べてみます！ 」

クラウドの消し忘れ・コスト増を防ぐために

▼表 2.1: クラウドコスト管理機能比較

サービス種別	AWS	Azure	GCP	説明
コスト監視	AWS Budgets	Cost Management	Cloud Billing	予算設定や使用状況の可視化
コスト分析	Cost Explorer	Cost Analysis	Billing Reports	コスト傾向の分析と将来予測
自動停止	Lambda + CloudWatch	Automation Runbooks	Cloud Scheduler	未使用リソースの自動停止

クラウドコストを効果的に管理するには、上記のような機能を活用することが重要です。特に以下の対策が有効です：

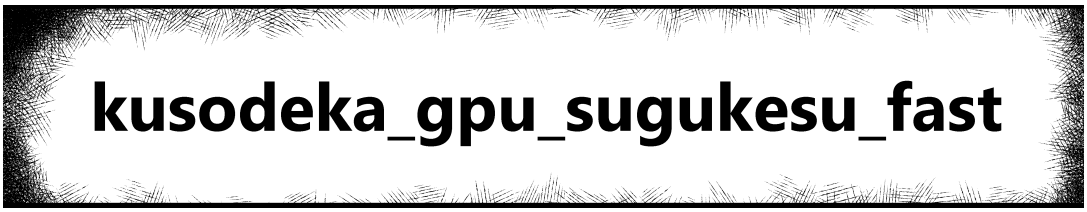
1. 予算アラートの設定：一定のコスト閾値に達した場合に通知 2. タグ付け戦略：部門・プロジェクト別コスト管理 3. 自動シャットダウン：開発環境の夜間・週末停止 4. 定期的なコスト分析：不要リソースの特定と削除

IaC と組み合わせることで、リソースのライフサイクル管理とコスト最適化を両立できます。

2.3　その他関連情報

今回のストーリーで取り扱ったトピックについて解説します。

kusodeka_gpu_sugugesu_fast とは



▲図 2.2: kusodeka_gpu_sugugesu_fast

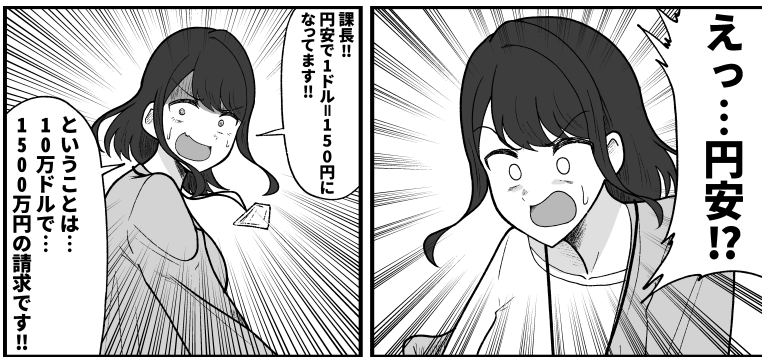
A. エンジニア特有の誇張表現です。

```
kusodeka_gpu = 性能が高いGPUが乗っているクラウドリソースで
sugugesu_fast = 値段も高いからすぐ消さないとヤバい
```

クラウドサービスはリソースを使った時間だけ課金がされます。また、基本的に性能が高ければ高いほど時間当たりの料金は高くなります。GPU 付きのコンピューティングインスタンスは高額なリソースの最たる例です。

```
ex: AWS p5.48xlargeの場合
- NVIDIA H100 GPUを8基搭載
- 1時間あたり約 $123.20 USD（約 ￥18,000〜￥19,000, 東京リージョンでのオンデマンド料金）
```

クラウドと円安について



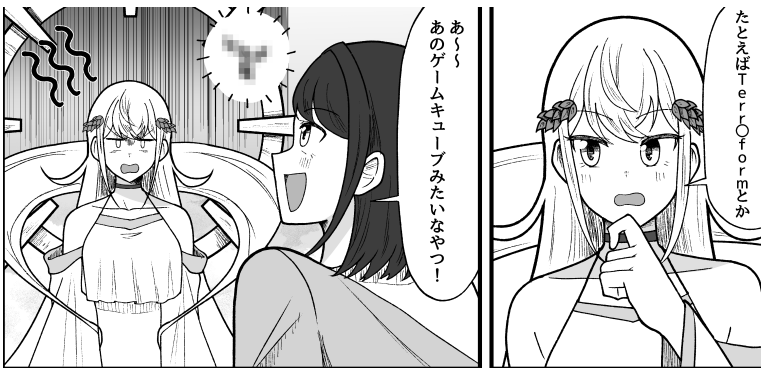
▲図 2.3: 円安の影響を受けるクラウド料金

海外のクラウドサービスを利用する場合、料金はドル建てで請求されます。日本円に換算すると、円安の影響を受けるため、円安が進むとクラウドサービスの料金も高くなります。

劇中では、冒頭では 1 ドル 100 円で計算していました。調べたところ最後に 1 ドル 100 円だったのは 2013 年らしいです。あくまでこの話は計算を分かりやすくするため現実世界と一致はしていません。一週間で 1 ドル 100 円から 150 円になったら日本経済壊れる。

ゲームキューブみたいなロゴ

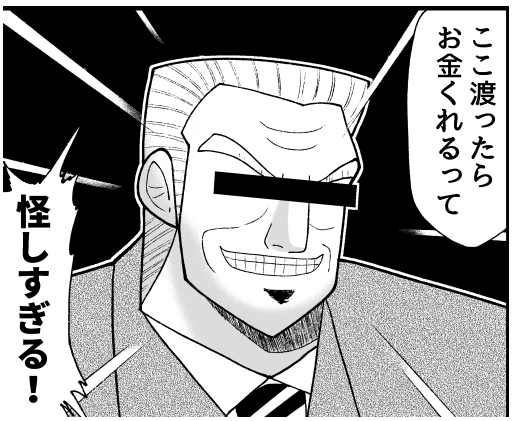
Terraform のロゴを見てゲームキューブを思い出すのは私だけですか？



▲図 2.4: ゲームキューブみたいなロゴ

お金をくれるおじさん

カ○ジのパロディです。



▲図 2.5: お金をくれるおじさん

第 3 章

機械学習工学

第 3 章では「機械学習工学」を題材にしています。機械学習工学は、機械学習を実際のシステムに組み込むための技術や手法を指します。なんでもできると思われがちな機械学習ですが、実際のシステムに組み込むためには、様々な課題があります。今回では、近頃流行っている機械学習 ☒ AI から一歩引いて運用面の解説をします。

3.1　あらすじ

突然暴走したロボットに襲われるりんね。どうやらシステムには AI が搭載されているものの、どうやら正しく制御が出来ていないようだった。どうやらその原因は、課長が入力した学習データにあるらしい。果たしてりんねと課長は正しいデータを準備し、AI を再学習させることができるのか？

「AI とデータってやっぱり切り離せないんですね〜」

「俺は生きる！ 生きてデータと添い遂げる！ 」

「AI ナーーーーー!! 」

3.2　機械学習工学について

機械学習工学は、機械学習を実際のシステムに組み込むための技術や手法を元にした学問体系であり、バイブルである〇〇社「機械学習工学」では、機械学習の本質的な問いを以下のように定義しています。

> 機械学習技術を用いて構築したソフトウェア，そしてそれを中心としたシステムにより，どれだけの価値創造・課題解決が可能ができるか
>
> そのシステムの開発や運用において，品質や効率をどのように高めていくか

つまり、機械学習のある特定のモデルの性能の枠を超えて、産業として活用する場合のうまいやり方を考えることだと解釈しています。今までのソフトウェア開発アプローチでは、ソースコードが全て、つまり人間がソフトウェアの挙動を定義するのに対して、機械学習の場合は、モデル・データが挙動の定義に重要な役割を果たします。機械学習工学では、その機械学習特有の概念である「モデル」「データ」に関する方法論も取り扱います。

また、機械学習工学が取り扱う範囲は以下の領域も含まれます: AI の倫理ほげほげ: 機械学習における知財・契約ふがふが

IaC の嬉しさ

▲図 3.1: とても長い手順書

IaC の嬉しさは、インフラの構成をコードとして管理できることです。これにより、インフラの構成や設定をプログラムとして記述し、バージョン管理や自動化が可能になります。例えば、本番環境と開発環境を同じ構成で作成したい場合、IaC を使うことで、同じコードを実行するだけで簡単に環境を再現できます。また、削除も一括で行うことができ、不要なリソースを漏れなくにクリーンアップできます。

IaC のデメリット

IaC のデメリットはどうしても手間がかかることです。IaC を使うためには、まずはコードを書く必要があり、記述方法に習熟する必要があります。また、IaC を使うことによりクラウドの利用により実現したい、迅速なリソースの立ち上げが実現できなくなる場合があります。その場合は、技術検証（PoC）の場合には、IaC を使わずに、手動でリソースを立ち上げることもあります。

「そうするとまた、リソースの消し忘れが起きちゃうんじゃ... 」

「だから、IaC だけが解決策じゃないってことね」

「どうすればいいんですか？ 」

「コストのアラートだったり、自動停止の設定だったり色々やれることはあるわね」

「なるほど、調べてみます！ 」

クラウドの消し忘れ・コスト増を防ぐために

▼表 3.1: クラウドコスト管理機能比較

サービス種別	AWS	Azure	GCP	説明
コスト監視	AWS Budgets	Cost Management	Cloud Billing	予算設定や使用状況の可視
コスト分析	Cost Explorer	Cost Analysis	Billing Reports	コスト傾向の分析と将来予
自動停止	Lambda + CloudWatch	Automation Runbooks	Cloud Scheduler	未使用リソースの自動停止

クラウドコストを効果的に管理するには、上記のような機能を活用することが重要です。特に以下の対策が有効です：

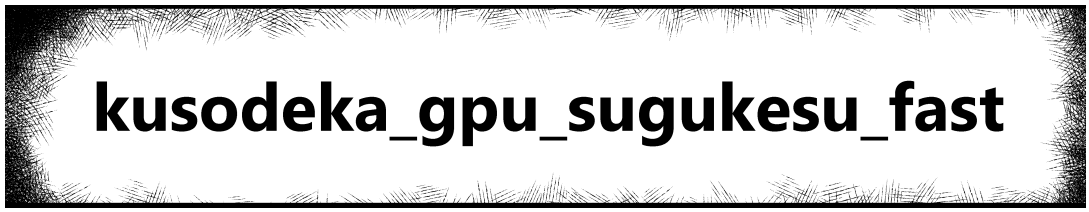
1. 予算アラートの設定：一定のコスト閾値に達した場合に通知 2. タグ付け戦略：部門・プロジェクト別コスト管理 3. 自動シャットダウン：開発環境の夜間・週末停止 4. 定期的なコスト分析：不要リソースの特定と削除

IaC と組み合わせることで、リソースのライフサイクル管理とコスト最適化を両立できます。

3.3　その他関連情報

今回のストーリーで取り扱ったトピックについて解説します。

kusodeka_gpu_sugugesu_fast とは



▲図 3.2: kusodeka_gpu_sugugesu_fast

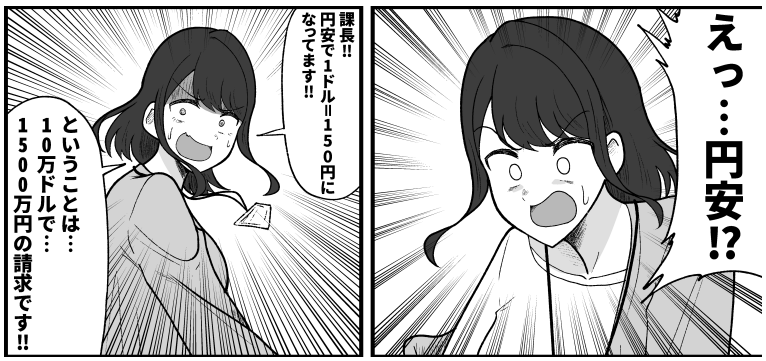
A. エンジニア特有の誇張表現です。

```
kusodeka_gpu = 性能が高いGPUが乗っているクラウドリソースで
sugugesu_fast = 値段も高いからすぐ消さないとヤバい
```

クラウドサービスはリソースを使った時間だけ課金がされます。また、基本的に性能が高ければ高いほど時間当たりの料金は高くなります。GPU 付きのコンピューティングインスタンスは高額なリソースの最たる例です。

```
ex: AWS p5.48xlargeの場合
- NVIDIA H100 GPUを8基搭載
- 1時間あたり約 $123.20 USD（約 ￥18,000〜￥19,000, 東京リージョンでのオンデマンド料金）
```

クラウドと円安について



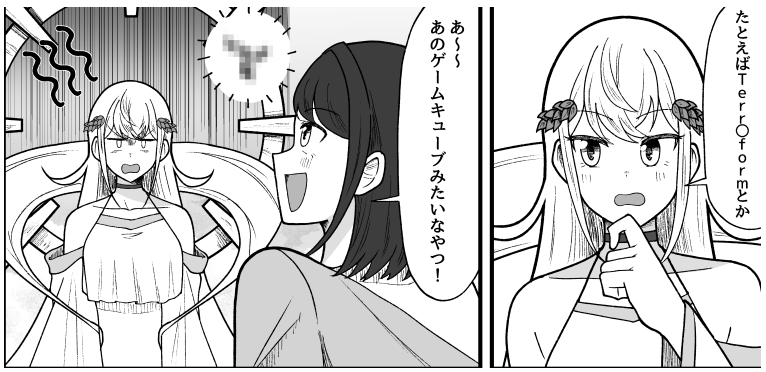
▲図 3.3: 円安の影響を受けるクラウド料金

海外のクラウドサービスを利用する場合、料金はドル建てで請求されます。日本円に換算すると、円安の影響を受けるため、円安が進むとクラウドサービスの料金も高くなります。

劇中では、冒頭では 1 ドル 100 円で計算していました。調べたところ最後に 1 ドル 100 円だったのは 2013 年らしいです。あくまでこの話は計算を分かりやすくするため現実世界と一致はしていません。一週間で 1 ドル 100 円から 150 円になったら日本経済壊れる。

ゲームキューブみたいなロゴ

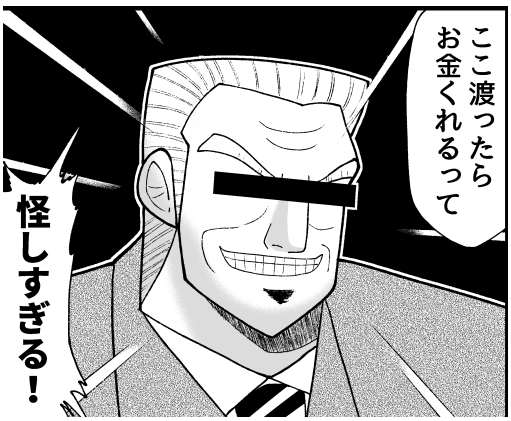
Terraform のロゴを見てゲームキューブを思い出すのは私だけですか？



▲図 3.4: ゲームキューブみたいなロゴ

お金をくれるおじさん

カ○ジのパロディです。



▲図 3.5: お金をくれるおじさん

第 4 章

形式手法

第 4 章では「形式手法」を題材にしています。形式手法は、ソフトウェアの設計や検証に数学的な手法を用いることを指します。中々実現のハードルが難しい技術ですが、考え方だけでもとても参考になります。今回の解説では、形式手法の説明をするとともに、Python でできる形式手法の実装例を紹介します。

4.1 あらすじ

とｔ

「インフラとコードって関係あるのかな？ 」

「記法は独特ですけど、ちゃんとコードで表現できるみたいですよ！ 」

「コードを見ると確かにそれっぽい！ 」

▼Terraform のコード例

```
provider "aws" {
  region = "ap-northeast-1"
}

resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "a" {
  vpc_id     = aws_vpc.main.id
  cidr_block = "10.0.1.0/24"
}

resource "aws_instance" "web" {
  ami           = var.ami
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.a.id
}
```

4.2 IaC について

IaC は○○年に登場した技術で、Infrastructure as Code の略です。IaC は、インフラをコードで管理することを指します。コードを実行すると、インフラが自動で構築されます。内部的にはコードを実行環境が解釈して、インフラを構築するための API を呼び出す仕組みになっています。

IaC の嬉しさ

▲図 4.1: とても長い手順書

IaC の嬉しさは、インフラの構成をコードとして管理できることです。これにより、インフラの構成や設定をプログラムとして記述し、バージョン管理や自動化が可能になります。例えば、本番環境と開発環境を同じ構成で作成したい場合、IaC を使うことで、同じコードを実行するだけで簡単に環境を再現できます。また、削除も一括で行うことができ、不要なリソースを漏れなくにクリーンアップできます。

IaC のデメリット

IaC のデメリットはどうしても手間がかかることです。IaC を使うためには、まずはコードを書く必要があり、記述方法に習熟する必要があります。また、IaC を使うことによりクラウドの利用により実現したい、迅速なリソースの立ち上げが実現できなくなる場合があります。その場合は、技術検証（PoC）の場合には、IaC を使わずに、手動でリソースを立ち上げることもあります。

「そうするとまた、リソースの消し忘れが起きちゃうんじゃ… 」

「だから、IaC だけが解決策じゃないってことね」

「どうすればいいんですか？ 」

「コストのアラートだったり、自動停止の設定だったり色々やれることはあるわね」

「なるほど、調べてみます！ 」

クラウドの消し忘れ・コスト増を防ぐために

▼表 4.1: クラウドコスト管理機能比較

サービス種別	AWS	Azure	GCP	説明
コスト監視	AWS Budgets	Cost Management	Cloud Billing	予算設定や使用状況の可視化
コスト分析	Cost Explorer	Cost Analysis	Billing Reports	コスト傾向の分析と将来予測
自動停止	Lambda + CloudWatch	Automation Runbooks	Cloud Scheduler	未使用リソースの自動停止

クラウドコストを効果的に管理するには、上記のような機能を活用することが重要です。特に以下の対策が有効です：

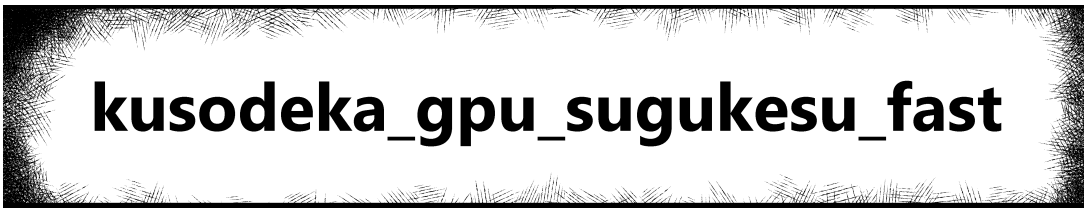
1. 予算アラートの設定：一定のコスト閾値に達した場合に通知 2. タグ付け戦略：部門・プロジェクト別コスト管理 3. 自動シャットダウン：開発環境の夜間・週末停止 4. 定期的なコスト分析：不要リソースの特定と削除

IaC と組み合わせることで、リソースのライフサイクル管理とコスト最適化を両立できます。

4.3　その他関連情報

今回のストーリーで取り扱ったトピックについて解説します。

kusodeka_gpu_sugugesu_fast とは



▲図 4.2: kusodeka_gpu_sugugesu_fast

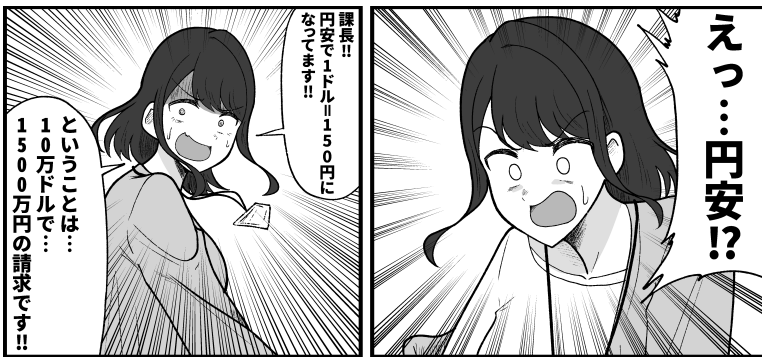
A. エンジニア特有の誇張表現です。

```
kusodeka_gpu = 性能が高いGPUが乗っているクラウドリソースで
sugugesu_fast = 値段も高いからすぐ消さないとヤバい
```

クラウドサービスはリソースを使った時間だけ課金がされます。また、基本的に性能が高ければ高いほど時間当たりの料金は高くなります。GPU 付きのコンピューティングインスタンスは高額なリソースの最たる例です。

```
ex: AWS p5.48xlargeの場合
- NVIDIA H100 GPUを8基搭載
- 1時間あたり約 $123.20 USD（約 ￥18,000〜￥19,000, 東京リージョンでのオンデマンド料金）
```

クラウドと円安について



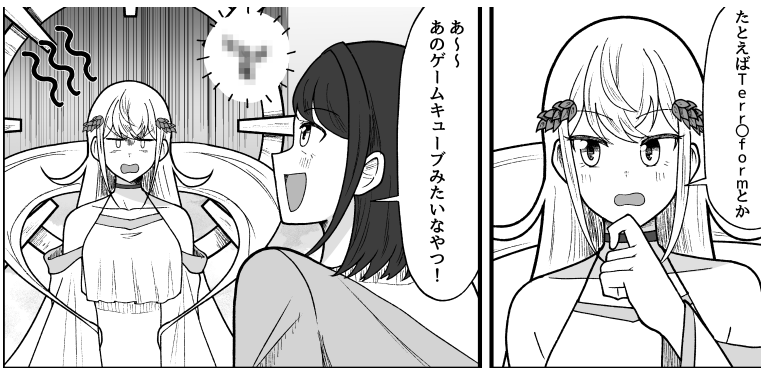
▲図 4.3: 円安の影響を受けるクラウド料金

海外のクラウドサービスを利用する場合、料金はドル建てで請求されます。日本円に換算すると、円安の影響を受けるため、円安が進むとクラウドサービスの料金も高くなります。

劇中では、冒頭では 1 ドル 100 円で計算していました。調べたところ最後に 1 ドル 100 円だったのは 2013 年らしいです。あくまでこの話は計算を分かりやすくするため現実世界と一致はしていません。一週間で 1 ドル 100 円から 150 円になったら日本経済壊れる。

ゲームキューブみたいなロゴ

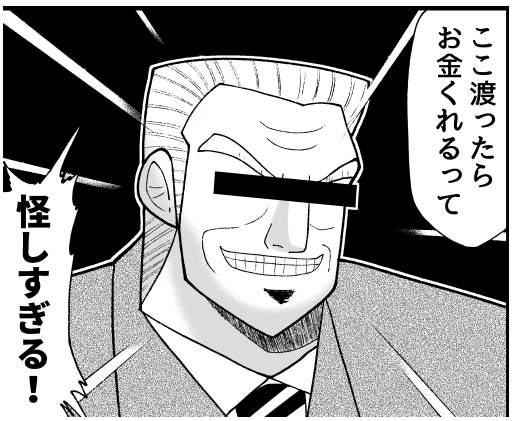
Terraform のロゴを見てゲームキューブを思い出すのは私だけですか？



▲図 4.4: ゲームキューブみたいなロゴ

お金をくれるおじさん

カ○ジのパロディです。



▲図 4.5: お金をくれるおじさん

付録 A
Re:VIEW との差分

Starter における機能追加や修正の一覧です。

A.1　まとめ記事

こちらの記事も参照してください。

- 『技術系同人誌を書く人の味方「Re:VIEW Starter」の紹介』(Qitta)
  `https://qiita.com/kauplan/items/d01e6e39a05be0b908a1`
- 『Re:VIEW Starter の新機能（2019 年夏）』(Qiita)
  `https://qiita.com/kauplan/items/dd8dd0f4a6e0eb539a98`
- 『Re:VIEW Starter の新機能（2019 年冬）』(Qitta)
  `https://qiita.com/kauplan/items/e36edd7900498e231aaf`

A.2　プロジェクト作成

- Starter では、Web サイトにおいて GUI でプロジェクトの作成と初期設定ができます。このおかげで、初心者でもつまづかずにプロジェクトを始められます。

  Re:VIEW ではプロジェクトの作成はコマンドラインでしか行えず、また初期設定を自分で行う必要があるので、初心者にはかなりハードルが高いです。
- Starter では、ダウンロードしたプロジェクトに文章のサンプルが含まれています。そのおかげで、どのように原稿を書けばいいのかが分かります。

  Re:VIEW では、作成したプロジェクトの中身はほとんど空であり、サンプルとしては役に立ちません。

A.3　コメント

- Starter では、範囲コメントが使えます。

  Re:VIEW では、行コメントしか使えません。
- Starter では、行コメントは単に読み飛ばされます。

  Re:VIEW では、行コメントが空行扱いになるため、行コメントを使うと意図せず段落が区切られます。これは明らかに Re:VIEW の仕様バグですが、Re:VIEW 開発チームはこのバグを認めていません。

A.4　箇条書き

- Starter では、順序つきリスト（HTML でいうところの「<ol>」）の項目に、「1.」や「(a)」や「(A-1)」など任意の文字列が使えます。
  Re:VIEW では「1.」しか使えません。
- Starter では、順序つきリストを入れ子にできます。
  Re:VIEW ではできません。
- Starter では、順序つきリストと順序なしリストを相互に入れ子にできます。
  Re:VIEW ではできません。
- Re:VIEW では、箇条書きの項目が複数行のテキストだと、自動的に 1 行に連結されます。そのせいで、本来なら「aaa」と「bbb」と「ccc」の 3 行に分かれていた単語が、「aaabbbccc」のように勝手に連結されてしまいます。
  Starter ではこのようなおかしなバグは起こりません。

A.5　定義リスト

- Re:VIEW では、定義リストの説明文が複数行だと、自動的に 1 行に連結されます。そのせいで、本来なら「aaa」と「bbb」と「ccc」の 3 行に分かれていた単語が、「aaabbbccc」のように勝手に連結されてしまいます。

  Starter ではこのようなおかしなバグは起こりません。
- Starter では、定義リストの説明文に箇条書きが書けます。

  Re:VIEW では書けません。
- Starter では、定義リストとよく似た「説明リスト」が用意されています。説明リストは入れ子のブロック命令で書くので、説明文の中にたとえばプログラムコードを入れられます。またブロック命令にオプション引数を指定すると、太字にする・しないや、コンパクトモードにする・しないが選べます。

A.6　見出し（章、節、項、目）

- Starter では、章や節のタイトルを見栄えのいいデザインにしています。また設定ファイルを変更することで、デザインを選択できます。

  Re:VIEW では LaTeX のデフォルトのデザインを使っているだけです。
- Starter では、章 (Chapter) ごとにタイトルページを作成できます。これは商業誌では一般的なスタイルです。
- Starter では、節 (Section) ごとに改ページできます。これは初心者向け入門書ではよく見かけるレイアウトです。
- Starter では、章や節を参照するとページ番号もつきます。これは特に、項 (Subsection) に番号をつけない設定において、項を参照するときに有用です。

A.7　インライン命令

- Starter では、たとえば「@<code>{foo@<b>{bar}}」のようにインライン命令を入れ子にできます。

  Re:VIEW ではできません。
- Starter では、「@{}」が太字かつゴシック体になります。なぜなら日本語の文章では、強調は太字にするだけでなくゴシック体にすることが望ましいからです。

  Re:VIEW では、「@{}」は太字になるだけです。
- Starter では、「@{}」の省略形として「@<B>{}」を用意しています。これにより、文章の強調が簡単にできます。

  Re:VIEW には、「@<B>{}」はありません。
- Starter では、何もしない命令「@<nop>{}」があります。これを使うと、たとえば「@@<nop>{}<B>{}」とすれば「@<B>{}」と表示されます。これは Re:VIEW を使って Re:VIEW のマニュアルを作るときに重宝します。

  Re:VIEW にはこのようなコマンドはありません。
- Re:VIEW では、たとえば「@<code>{a: b? c! d. e}」と入力すると、「{a: b? c! d. e}」のように半角空白が 2 つ分出力されます。このような意図しない出力になるのは、LaTeX におけるスペーシングの仕様をそのまま引き継いでいるからです。
  Starter ではこの現象を回避しており、等幅フォントでも半角空白が 2 つ分出力されないように修正しています。
- Starter では、ファイル名を表すインライン命令「@<file>{}」が用意されています。
- Starter では、文字サイズを変更するインライン命令「@<small>{}」「@<xsmall>{}」「@<xxsmall>{}」「@<large>{}」「@<xlargel>{}」「@<xxlarge>{}」が用意されています。
- Starter では、「LaTeX」を表示する「@<LaTeX>{}」、「TeX」を表示する「@<TeX>{}」、「♡」を表示する「@<hearts>{}」が用意されています。
- Starter では、リンクの URL を自動的に脚注に表示する機能があります。これは、紙の本ではリンクの URL が表示されない問題に対する解決策です。

A.8　ブロック命令

- Starter では、ブロック命令を入れ子にできます。たとえば「`//note{ ... //}`」の中に「`//list{ .. //}`」や「`//quote{ ... //}`」を入れられます。
- Starter では、章の概要を表す「`//abstract`」が用意されています。

  Re:VIEW にはありませんが、かわりにリード文を表す「`//lead`」が使えます。
- Starter では、縦方向の空きを入れる「`//vspace`」が用意されています。マイナスの値を指定すれば、余分な空きを削除できます。
- Starter では、縦方向のスペースを確保する「`//needvspace`」が用意されています。

  たとえばプログラムのキャプションだけが現在のページ、プログラムコードは次のページに表示されてしまったとします。このとき「`//list`」の前に「`//needvspace[6zw]`」と書くと、全角 6 文字分の高さのスペースがなければ自動的に改ページされます（つまりプログラムのキャプションの前で改ページされる）。結果として、プログラムのキャプションとコードが同じページに表示されます。

A.9　プログラムコード

- Starter では、「1」と「l」、「0」と「O」の見分けがつきやすいフォントを選んでいます。

  Re:VIEW では LaTeX のデフォルトを使っているので、これらの見分けがつきにくいです。
- プログラムコードを表示するとき、Starter では 1 つのブロック命令「`//list`」だけで済みます。

  Re:VIEW では「`//list`」「`//emlist`」「`//listnum`」「`//emlistnum`」の 4 つを使い分ける必要があります。
- Starter では、長すぎる行を自動的に右端で折り返します。また折り返したことが分かるような記号をつけてくれます。

  Re:VIEW にはこのような気の利いた機能がありません。
- Starter では、「`@<del>{}`」で取り消し線を引いた行でも右端で折り返しされます。Starter ではこのような細かいところに配慮が行き届いています。
- Starter では、行番号は目立たないようにグレーで表示します。

  Re:VIEW にはこのような気の利いた機能がありません。
- Starter では、行番号を欄外に表示できます。これにより、行番号の分だけコードの表示範囲が狭まってしまうのを防げます。
- Starter では、飛び飛びの行番号を指定できます。

  Re:VIEW では連続した行番号しか表示できません。
- Starter では、インデントを可視化できます。この機能は、Python のようなインデントで入れ子を表現するプログラミング言語では重要です。
- Starter では、ラベルさえ指定されていればキャプションがなくても番号つきで表示されます。

  Re:VIEW では、「`//list[fib1]`」のようにラベルを指定していても、キャプションがなければ番号がつきません。ラベルの目的を考えれば、これは Re:VIEW の仕様バグです。
- Starter では、「`//list[?]`」のようにすると簡単に番号付きで表示されます。

  Re:VIEW にはこの機能がないので、番号つきで表示するには重複しないラベルを指定する必要があり、数が多いとかなり面倒です。
- Starter では、プログラムコードのフォントサイズを変更できます。「あるプログラムコードだけフォントサイズを小さくする」といったことが簡単にできます。
- Starter では、コード中の注釈を表す「`@<balloon>{}`」がグレー表示されます。これは注釈がコードより目立たないようにという配慮です。

  Re:VIEW にはこのような配慮はありません。
- プログラム中にタブ文字がある場合、Re:VIEW では空白文字に展開されます。しかしこれが「LaTeX コードに変換してからタブ文字を展開する」という仕様のため、表示が崩れます。

  Starter ではタブ文字を展開してから LaTeX コードに変換する」ため、表示が崩れません

（ただし限界はあります）。

- Starter では、全角文字の幅を半角文字 2 つ分に揃えて表示する機能があります。全角と半角が混在しているせいで表示が揃わない（崩れてしまう）場合に使うと大変便利です。
- Starter では、プログラムコードとは別に出力結果を表示するための「`//output`」があります。プログラムコードと出力結果では見た目を変えて表示したい場合に便利です。
- Re:VIEW では、プログラムコードをハイライト表示する機能があります。

  Starter ではまだ未サポートです。

A.10　ターミナル画面

- Starter では、ターミナル画面を表すコマンド命令「`//terminal`」を用意しています。これは使い方が「`//list`」と同じなので、覚えやすいです。

  Re:VIEW で用意しているブロック命令「`//cmd`」は、「`//list`」と使い方が違っているので覚えにくいです。
- Starter の「`//terminal`」はほぼ「`//list`」と同じなので、折返しや行番号など「`//list`」と同じ機能が「`//terminal`」でも使えます。

  Re:VIEW の「`//cmd`」はそうではないので、機能が貧弱です。
- Starter では、ユーザ入力を表すインライン命令「`@<userinput>{}`」があります。これを使うと、ユーザ入力部分に薄い下線が引かれます。ただし LaTeX の制限により、長い行が右端で折り返しされません。
- Starter では、カーソルを表すインライン命令「`@<cursor>{}`」が用意されています。Vim の操作画面を説明するときに便利です。

A.11　ノート、ミニブロック

- Starter では、ノートを表すブロック命令「//note」を上書きし、見栄えのいい表示にしています。

  Re:VIEW ではノートの見栄えは気にされていません。
- Starter では、ノートの中に箇条書きや他のブロック命令を入れられます。
- Starter では、ノートの中で脚注が使えます。
- Starter では、ノートの中に書いたプログラムコードやターミナル画面が、ページまたぎできます。
- Starter では、ノートにラベルをつけてあとから参照できます。これはちょうど、図や表を参照するのと同じような機能です。
- Starter では、補足情報を表す「//info」と注意喚起を表す「//caution」と警告を表す「//warning」にアイコンがつきます。

A.12 図

- Re:VIEW では、たとえば「`//image[][][scale=0.5]`」のようなオプションを指定すると、画像が本文幅の半分の大きさで表示されます。しかしこれは画像の幅が本文幅より大きい場合だけであり、画像の幅が本文幅より小さい場合は画像幅の半分の大きさで表示されます。そのため、画像の幅が本文幅より大きいかどうかを気にする必要があり、面倒です。

  Starter では「`//image[][][width=50%]`」のように指定すると、画像の幅が本文幅より大きいかどうかに関係なく、常に本文幅の半分の大きさで画像が表示されます。
- Starter では「`//image[][][width=40mm]`」のように、画像の表示幅を絶対値で指定できます。つまり画像の表示幅を本文の幅とは関係なく指定できます。
- 画像の幅が本文幅より小さい場合、表示幅を指定せずに画像を表示すると、Re:VIEW では PDF ならそのままの大きさで表示されるのに、HTML では本文幅いっぱいの大きさで表示されてしまいます。

  Starter では、PDF と HTML のどちらでもそのままの大きさで画像が表示されます。
- Starter では、画像の挿入位置を指定できます（現在位置、ページ上部、ページ下部、別ページ）。
- Starter では、図のまわりをグレーの線で囲む機能があります。画像のふちが白い場合はこの機能を使うと、画像の境界を明示できます。
- 画像が次のページに送られたとき、Starter では後続のテキストを現在位置に流し込まれます。

  Re:VIEW では後続のテキストが現在位置に流し込まれないので、現在位置に大きなスペースが空きます。
- 「`//indepimage`」の第 2 引数に説明文字列を指定した場合、Re:VIEW では「図：」という接頭辞がつきます。しかし「`//indepimage`」の使用用途を考えたとき、この接頭辞は邪魔なので、Starter ではつけません。もし必要なら自力で「図：」を入れてください。
- Starter では、「`//sideimage`」を使うと画像の横にテキストを表示できます。

A.13　表

- Starter では、表の挿入位置を指定できます（現在位置、ページ上部、ページ下部、別ページ）。
- Starter では、表のデータを CSV 形式で指定できます。
- Starter では、表のデータを外部ファイルから読み込めます。
- Starter では、行ごとの罫線をオン・オフできます。また行ごとの罫線をオフにしたうえで、指定箇所だけ罫線を引くよう指定できます。
- Starter では、「`//table[label][]`」のようにラベルだけ指定してキャプションがない場合でも、「表 1.1」のような番号がつきます。

  Re:VIEW では、ラベルが指定されてあってもキャプションがなければ番号が表示されません。
- Starter では、「`//table[?][]`」のように指定すると内部でラベルを自動的に割り当ててくれます。そのおかげで、表を番号つきで表示するのが簡単です。

  Re:VIEW ではこのような機能がないので、番号つきで表示するには重複しないラベルを必ず指定する必要があり、面倒です。
- 表のカラムの右寄せや左寄せを指定する「`//tsize`」を、Starter では「`//tsize[latex][|lcr|]`」のように指定できます。

  Re:VIEW では「`//tsize[|latex||lcr|]`」のように指定するため、読みづらいし分かりにくいです。
- Re:VIEW には、項目の区切り文字を変更できる機能があります。

  Starter では未サポートです。

A.14　会話形式

- Starter では、アイコン画像つきの会話形式を用意しています。
- Starter では、アイコン画像なしの会話形式も用意しています。
- Starter では、大量の会話形式を入力するときに便利なように、短縮名を登録して短く書ける機能を用意しています。

A.15　用語、索引

- Starter には、用語を表す「@<term>{}」インライン命令があります。これは「@<idx>{}」と似ていますが、用語をゴシック体で表示する点が違います。用語が初めて登場したときは「@<term>{}」、再登場した場合は「@<idx>{}」を使うことを想定しています。
- Starter では、「@<idx>{}」や「@<term>{}」の中によみがなを指定できます。
- Starter では、親子関係と表示順序が逆転している場合にも「~~~」を使えば対応できます。
- Starter では、索引ページにおいて用語の転送先となる別の用語を指定できます。
- Re:VIEW の索引ページは、LaTeX のデフォルトデザインのままです。Starter では、用語とページ番号の間を連続したドットで埋めたり、用語のグループ化を文字単位から行単位に変更しています。

A.16　表紙、大扉、奥付

- Starter では、表紙と裏表紙は PDF ファイルで指定します。
  Re:VIEW では、表紙は画像ファイル（PNG や JPG）で指定します。
- 本のタイトルが長い場合、Starter ではタイトルを複数行で指定すれば、改行位置を指定できます。このおかげで、大扉では長いタイトルがきれいに改行されて表示されます。
  Re:VIEW ではこのような機能はありません。
- Starter では、大扉の裏に免責事項などが書かれています。これは商業誌では一般的ですが、同人誌では入れ忘れることがほとんどなので、あらかじめ Starter が用意しています。
- Starter では、奥付が必ず偶数ページ（見開きで左側のページ）になるよう調整します。
  Re:VIEW ではこのような機能はありません[*1]。

[*1] TechBooster のテンプレートでは、Starter と同じように奥付が偶数ページになります。

A.17　PDF ファイル

- Starter では、印刷用 PDF では白黒に、電子用 PDF ではカラーにしてくれます。
  Re:VIEW ではこのような機能はありません。
- Starter では、印刷用 PDF では左右の余白幅を切り替えています。これは本文幅を広げるための工夫であり、商業誌でも一般的です。
  Re:VIEW ではこのような工夫はされていません。
- Re:VIEW では、印刷用 PDF にはハイパーリンクがつかないようになっています。これは入稿時のトラブルをできるだけ減らすための仕様です。
  しかし LaTeX ではハイパーリンクのある・なしで組版結果が変わることがあります（hyperref.sty の仕様）。また最近の印刷所では、ハイパーリンクがついた PDF でも入稿でトラブルになることはなさそうです。そこで Starter では、印刷用 PDF でもハイパーリンクを残したままにしています。
- Re:VIEW では、デフォルトで PDF にトンボがつきます。しかしこれは入稿に慣れてないとトラブルのもとです。入稿先が同人誌向けの印刷会社なら、トンボなしのほうが入稿トラブルは少ないです。
  そのため、Starter ではトンボはつけません。
- Re:VIEW と Starter のどちらとも、印刷用 PDF には隠しノンブル（通し番号）がつきます。ただし実装方法は Re:VIEW と Starter とで全く別です。また Starter では、ノンブルの位置や大きさや色や開始番号を設定ファイルで変更できます。
- 印刷用 PDF に自動でノンブルがつくのは、PDF の作成が Re:VIEW だけで完結している場合は便利ですが、たとえば大扉を別ソフトで作るような場合は逆に不便です。
  そこで Starter では、PDF を生成したあとに隠しノンブルをつける Rake タスク「`rake pdf:nombre`」も用意しています。これだと、大扉だけを別ソフトで作るような場合にも対応できます。
- Starter では、PDF ファイルにノンブルをつける別のサービス[*2]も用意しています。
- Starter では、PDF のテキストをマウスで選択したときに、行番号や折り返し記号やインデント記号を選択対象にしない機能が用意されています。ただしこの機能はコンパイル時間を大きく増やしてしまうので、デフォルトではオフです。

[*2] `https://kauplan.org/pdfoperation/`

A.18 LaTeX 関連

- Re:VIEW では、LuaLaTeX と jlreq.cls を積極的に利用しており、採用実績も多いです。

  Starter では、今のところ LuaLaTeX と jlreq には対応していません。今後の課題です。
- Re:VIEW そのものではありませんが、Techbooster の Re:VIEW テンプレートでは tcolorbox を全面的に採用しており、そのおかげでたとえば角丸のブロックを実現できています。

  ただし tcolorbox を使うとコンパイル時間がかかるようになるので、Starter では使わないようにしています。tcolorbox を使うと見栄えのいいデザインができることは確かなので、今後どうするかは検討中です。
- Starter では、環境変数「`$STARTER_CHAPTER`」を設定することで、特定の章だけをコンパイルできます。こうすると全部の章をコンパイルするよりずっと短い時間でコンパイルできるので、表示の確認が迅速にできます。

  Re:VIEW にはこのような機能はありません。
- Starter では、LaTeX のコンパイル回数を減らす工夫をしています。文章を少しだけ変更して再度コンパイルすると、Starter ではコンパイル回数がたいてい 1 回だけで済みます。

  Re:VIEW ではいつも 3 回コンパイルされるので、PDF 生成に時間がかかります。
- Re:VIEW では、PDF ファイルを生成する「`review-pdfmaker`」コマンドや、ePub ファイルを生成する「`review-epubmaker`」コマンドが用意されています。

  Re:VIEW ではこれらは使えません。必ず「`rake pdf`」や「`rake epub`」を使ってください。
- 「`rake pdf`」や「`rake web`」を実行すると、Starter ではコンパイルが必ず実行されます。

  Re:VIEW ではファイルが更新されていなければコンパイルされません。これは一見便利そうですが、LaTeX のスタイルファイルを変更しただけではコンパイルされないため、トラブルのもとになります。
- Starter では、設定ファイル「config-starter.yml」の内容を LaTeX のスタイルファイルから参照できます。設定によって挙動を変えたい場合に便利です。
- Starter では、名前が「`STARTER_`」で始まる環境変数の値を LaTeX のスタイルファイルから参照できます。環境変数によって挙動を変えたい場合に便利です。
- Starter では表示を簡潔にするため、LaTeX コンパイル時の出力を捨てています。コンパイルエラーがあった場合は、出力を捨てないようにコマンドオプションを変更してからもう一度コンパイルすることで、エラーメッセージを表示しています。

  Re:VIEW でも出力は捨てていますが、コンパイルエラーがあった場合はコンパイル時のログファイルを表示しています。しかしこの方法だとエラメッセージが少し上のほうに流れてしまうので、LaTeX に慣れていない人だとエラーメッセージを見逃してしまいます。
- PDF ファイルを生成するとき、Starter ではもとの PDF ファイルを削除せずに上書きする（つまり inode が変わらない）ようにしています。このおかげで、macOS の `open` コマンド

で PDF ファイルを開くと、同じウィンドウのまま表示されます。

　これに対し、Re:VIEW ではもとの PDF ファイルを消してから新しいファイルを作成します（つまり inode が変わる）。そのせいで、macOS の open コマンドで PDF ファイルを開くと毎回新しいウィンドウで表示されてしまい、いちいち前のウィンドウを閉じる必要があるので面倒です。

- Starter では、LaTeX のコンパイル時間を表示します。以前は「/usr/bin/time」コマンドが必要でしたが、現在は必要としません。

A.19　エラメッセージ

- Starter では、原稿ファイル（*.re）コンパイル時のエラーメッセージを赤い字で表示します。初心者はエラーメッセージが出ても気づかないことが多いので、赤く表示することで気づきやすいようにしました。
- Starter では、原稿ファイル（*.re）のコンパイル時にエラーがあると、すぐにコンパイルをすぐに終了するようにしました。初心者にとっては、このほうがエラーに気づきやすいはずです。

  Re:VIEW ではコンパイルエラーがあっても残りのコンパイルを続けるので、エラーメッセージが流れてしまい、初心者はエラーに気づかない可能性が高いです。

A.20　その他

- Re:VIEW では、プロジェクトが使っている Re:VIEW のバージョンをアップデートする機能が用意されています。たとえば Re:VIEW 4 を使っているプロジェクトを Re:VIEW 5 へアップデートしたいときは、ターミナルで「`review-update`」コマンドを実行します。Starter にはプロジェクトをアップデートする機能はありません。
- Starter では関連プロジェクトとして、PDF に隠しノンブルをつけるサービス「PDF Operation[*3]」を提供しています。

[*3] `https://kauplan.org/pdfoperation/`

付録 B
ファイルとフォルダ

プロジェクトの zip ファイル（例：mybook.zip）を解凍してできるファイルとフォルダの解説です。

README.md

プロジェクトの説明が書かれたファイルです。ユーザが好きなように上書きすることを想定しています。

Rakefile

rake コマンドが使用します。コマンドを追加する場合は、このファイルは変更せず、かわりに「lib/tasks/*.rake」を変更してください。

catalog.yml

原稿ファイルが順番に並べられたファイルです。原稿ファイルを追加した・削除した場合は、このファイルも編集します。

config-starter.yml

Starter 独自の設定ファイルです。Starter では設定ファイルとして「cofnig.yml」と「cofnig-starter.yml」の両方を使います。

config.yml

Re:VIEW の設定ファイルです。Starter によりいくつか変更と拡張がされています。

contents/

原稿ファイルを置くフォルダです[*1]。

contents/*.re

原稿ファイルです。章 (Chapter) ごとにファイルが分かれます。

css/

HTML ファイルで使う CSS ファイルを置くフォルダです。ePub で使うのはこれではなくて style.css なので注意してください。

images/

画像ファイルを置くフォルダです。この下に章 (Chapter) ごとのサブフォルダを作ることもできます。

[*1] 原稿ファイルを置くフォルダ名は「config.yml」の「contentdir: contents」で変更できます。

layouts/layout.epub.erb

原稿ファイルから ePub ファイルを生成するためのテンプレートです。

layouts/layout.html5.erb

原稿ファイルから HTML ファイルを生成するためのテンプレートです。

layouts/layout.tex.erb

原稿ファイルから LaTeX ファイルを生成するためのテンプレートです。

lib/hooks/beforetexcompile.rb

LaTeX ファイルをコンパイルする前に編集するスクリプトです。

lib/ruby/*.rb

Starter による Re:VIEW の拡張を行う Ruby スクリプトです。

lib/ruby/mytasks.rake

ユーザ独自の Rake コマンドを追加するためのファイルです。

lib/ruby/review.rake

Re:VIEW で用意されている Rake タスクのファイルです。Starter によって変更や拡張がされています。

lib/ruby/review.rake.orig

Starter によって変更や拡張がされる前の、オリジナルのタスクファイルです。

lib/ruby/starter.rake

Starter が追加した Rake タスクが定義されたファイルです。

locale.yml

国際化用のファイルです。たとえば「リスト 1.1」を「プログラム 1.1」に変更したい場合は、このファイルを変更します。

mybook-epub/

ePub ファイルを生成するときの中間生成ファイルが置かれるフォルダです。通常は気にする必要はありません。

mybook-pdf/

PDF ファイルを生成するときの中間生成ファイルが置かれるフォルダです。LaTeX ファイルをデバッグするときに必要となりますが、通常は気にする必要はありません。

mybook.epub

生成された ePub ファイルです。ファイル名はプロジェクトによって異なります。

mybook.pdf

生成された PDF ファイルです。ファイル名はプロジェクトによって異なります。

review-ext.rb

Re:VIEW を拡張するためのファイルです。このファイルから「lib/ruby/*.rb」が読み込まれています。

sty/

LaTeX で使うスタイルファイルが置かれるフォルダです。

sty/jumoline.sty

LaTeX で使うスタイルファイルのひとつです。

sty/mycolophon.sty

奥付[*2]の内容が書かれたスタイルファイルです。奥付を変更したい場合はこのファイルを編集します。

sty/mystyle.sty

ユーザが独自に LaTeX マクロを定義・上書きするためのファイルです。中身は空であり、ユーザが自由に追加して構いません。

sty/mytextsize.sty

PDF における本文の高さと幅を定義したファイルです。LaTeX では最初に本文の高さと幅を決める必要があるので、他のスタイルファイルから分離されてコンパイルの最初に読み込めるようになっています。

sty/mytitlepage.sty

大扉[*3]の内容が書かれたスタイルファイルです。大扉のデザインを変更したい場合はこのファイルを編集します。

sty/starter.sty

Starter 独自のスタイルファイルです。ここに書かれた LaTeX マクロを変更したい場合は、このファイルを変更するよりも「sty/mystyle.sty」に書いたほうがバージョンアップがしやすくなります。

sty/starter-codeblock.sty

プログラムコードやターミナルの LaTeX マクロが定義されています。

sty/starter-color.sty

色のカスタマイズ用です。

sty/starter-font.sty

フォントのカスタマイズ用です。

sty/starter-headline.sty

章 (Chapter) や節 (Section) や項 (Subsection) の LaTeX マクロが定義されたファイルです。

sty/starter-note.sty

ノートブロックの LaTeX マクロが定義されています。

sty/starter-section.sty

以前の、章や節の LaTeX マクロ定義です。もはや使ってませんが、starter.sty を書き換えれば使えます。

sty/starter-talklist.sty

会話形式の LaTeX マクロが定義されています。

[*2] **奥付**とは、本のタイトルや著者や出版社や版や刷などの情報が書かれたページのことです。通常は本のいちばん最後のページに置かれます。

[*3] **大扉**とは、タイトルページのことです。表紙のことではありません。

sty/starter-toc.sty

目次のカスタマイズ用です。

sty/starter-misc.sty

その他、各種のマクロ定義です。

sty/starter-util.sty

Starter のスタイルファイルで使われるマクロが定義されています。

style.css

ePub で使われる CSS スタイルファイルです。

付録 C
開発の背景

ここでは歴史の記録として、Re:VIEW Starter が開発された背景などを記しておきます。個人的な忘備録として残すものであり、読まなくても本や同人誌の制作にはまったく支障はありません。

開発の経緯

Re:VIEW は ver 2.4 の頃、「A5 サイズの PDF ファイルが生成できない」「フォントサイズも 10pt から変更できない」という致命的な問題を抱えていました。つまり B5 サイズでフォントが 10pt の本しか作れなかったのです。

この頃はまだ、技術同人誌は B5 サイズ 10pt で作るのが主流だったので、Re:VIEW のこの制限はあまり問題にはなりませんでした。しかし同人誌印刷では B5 サイズより A5 サイズのほうが割安なので、一部の人が A5 サイズやより小さいフォントサイズでの PDF 生成を模索し、そして Re:VIEW の制限に苦しみました。

たとえば、フォントサイズを小さくしようとして格闘し、結果としてあきらめた人の証言を見てみましょう[*1]（図 C.1）。

この問題は、「geometry.sty」というスタイルファイルをオプションなしで読み込んでいることが原因です[*2]。具体的な修正方法は Qiita の記事[*3]に書いています。読めば分かりますが、かなり面倒です。

この不具合はバグ報告したものの、開発陣の反応は芳しくなく、すぐには修正されなさそうでした[*4]。当時は「技術書典」という同人イベントの開催が迫っていたので、これは困りました。仕方ないので Re:VIEW 側での修正に期待せず、誰でも簡単に A5 サイズの同人誌が作れるための別の方法を考えることにしました。

[*1] https://www.slideshare.net/KazutoshiKashimoto/nagoya0927-release の p.21 と p.22。

[*2] 簡単に書いてますけど、原因が geometry.sty であることを突き止めるのには多くの時間がかかり、正月休みが潰れました。今でも恨んでます。この苦労を知らずに「Re:VIEW では昔から A5 の PDF が簡単に生成できた」と言い張る歴史修正者にはケツバットの刑を与えてやりたいくらいです。

[*3] https://qiita.com/kauplan/items/01dee0249802711d30a6

[*4] 実際、Re:VIEW Ver 2 の間は修正されず、修正されたのは Ver 3 になってからでした。当然、技術書典には間に合いませんでした。

▲図 C.1: フォントやページサイズを変更できなかった人の証言

そうした経緯で誕生したのが、Re:VIEW Starter です。Starter では初期設定が GUI でできる簡単さが売りのひとつですが、もともとは A5 サイズ 9pt の同人誌を簡単に作れることが開発動機だったのです。

開発の転機

Starter は、当初は Re:VIEW との違いが大きくならないよう、GUI で設定した状態のプロジェクトをダウンロードできるだけに留めていました。つまりユーザが手作業で設定するのを、GUI で簡単に設定できるようにしただけでした。

しかし Re:VIEW は、ドキュメント作成ツールとしての基本機能が足りておらず、そのうえ開発速度が遅くて技術書典といったイベントには新機能追加が間に合いません。またバグ報告をしても「仕様だ」と回答されたり、互換性を理由に却下されたり（けど他の人が報告すると取り入れられたり）といったことが積み重なって、ある日を境に Starter で独自の機能を追加することを決心しました。

そこからは少しずつ Re:VIEW の機能を上書きし、足りない機能を追加していきました。

現在の Starter では、Re:VIEW のソースコードを広範囲に上書きしています[*5]。特にパーサの大部分は Starter 側で上書きしています。このおかげで、インライン命令やブロック命令を入れ子対応にしたり、箇条書きの機能を大きく拡張したりできています。

Re:VIEW とのかい離はずいぶんと大きくなりました。

[*5] 実は Re:VIEW のソースコードを広範囲に上書きしているせいで、ベースとなる Re:VIEW のバージョンを 2.5 から上げられていません。しかし Re:VIEW の開発速度が遅く目ぼしい新機能は追加されてないので、デメリットはありません。

開発の障害

Starter を開発するうえで大きな障害になったのが、LaTeX と、Re:VIEW のコード品質です。

「LaTeX」はフリーの組版ソフトウェアであり、Re:VIEW や Starter が PDF ファイルを生成するときに内部で使っています。LaTeX は出力結果がとてもきれいですが、使いこなすには相当な知識が必要です。

特に LaTeX のデバッグは困難を極めます。エラーメッセージがろくに役立たないせいでエラーを見ても原因が分からない、どう修正すればいいかも分からない、直ったとしてもなぜ直ったのか分からない、分からないことだらけです。この文章を読んでいる人にアドバイスできることがあるとすれば、「LaTeX には関わるな！」です。

また Re:VIEW のコード品質の悪さも、大きな障害となりました。ソースコードを読んでも何を意図しているのか理解しづらい、コードの重複があちこちにある、パーサで行うべきことを Builder クラスで行っている、…など、基礎レベルでのリファクタリングがされていません。信じたくはないでしょうが、同じような感想を持った人が他にもいたので紹介します。

`https://np-complete.gitbook.io/c86-kancolle-api/atogaki`:

> っと上の文章を書いてから 2 時間ほど Re:VIEW のコードと格闘していたんですが、なんですかこのクソコードは・・・これはマジでちょっとビビるレベルのクソコードですよ。夏コミが無事に終わったら冬に向けて Re:VIEW にプルリク送りまくるしかないと思いました。

「クソコード」は言い過ぎですが、そう言いたくなる気持ちはとてもよく分かります。

また、パースしたあとに構文木を作っていないのは、Re:VIEW の重大な設計ミスといえるでしょう。Re:VIEW や Markdown のようなドキュメント作成ツールでは、一般的に次のような設計にします。

1. パーサが入力テキストを解析して、構文木を作る。
2. 構文木をたどって必要な改変をする。
3. Visitor パターンを使って、構文木を HTML や LaTeX のコードに変換する。

このような設計であれば、機能追加は容易です。特に HTML や LaTeX のコードを生成するより前にすべての入力テキストがパース済みなので、たとえ入力テキストの最後に書かれたコマンドであろうと、それを認識して先頭のコードを柔軟に変更できます。

しかし Re:VIEW は構文木を作らず、パースしながら HTML や LaTeX のコードに変換するため、しなくてもいいはずのハックが必要となることがあります。

Starter は将来的に Re:VIEW のパーサをすべて上書きして、構文木を生成するタイプのパーサへ置き換えることになるでしょう。

開発方針の違い

Re:VIEW と Starter では、開発方針に大きな違いがあります。

開発速度

Re:VIEW では、リリースが年 3 回と決まっています。またリリース時期も（過去を見る限り）2 月末、6 月末、10 月末に固定されています。そのため「次の技術書典までにこの新機能が必要だ」と思っても、固定されたリリース時期にならないと新機能はリリースされません。言い方を変えると、ユーザが必要とする開発速度についてこれていません。

Starter は主に、技術書典をはじめとした同人イベントに合わせて新機能が開発されます。また 2〜3 週間ごとにリリースされるので、新機能の追加とバグ修正が急ピッチで進みます。つまり、ユーザが必要とする速度で開発されています。

機能選定

Re:VIEW は商業誌での利用実績が多いこともあって、「日本語組版処理の要件[*6]」への対応が重視されます。そのため jlreq クラスファイル[*7]が標準でサポートされており、少なくない開発リソースがその対応に割かれています。そのせいか、「範囲コメントを実装する」「インライン命令やブロック命令を入れ子に対応させる」「順序つき箇条書きで数字以外を使えるようにする」などの基本機能が、未だに実装されていません。

しかし「日本語組版処理の要件」は、読んでみると分かりますが、重箱の隅をつつくような内容がほとんどです。プロユースでは必要なのかもしれませんが、同人誌では「禁則処理がきちんとできていれば充分」というユーザがほとんどでしょう。開発リソースをそんな細かいところに割くよりも、もっとユーザが必要とする機能の開発に割くべきです。

Starter では日本語組版の細かい要件は気にせず、ユーザが必要とする機能を重点的に開発しています。たとえば、範囲コメントを実装したり、インライン命令やブロック命令を入れ子に対応させたり、順序つき箇条書きでアルファベットも使えるようにしたり、章や節のタイトルを見栄えのいいデザインにしたりといった、ユーザにとって必要な機能を優先して開発しています[*8]。

バグ対応

今まで、Re:VIEW には約 20 個ほどのバグ報告や Pull Request を出しました。取り入れられたものも多いですが、理不尽な理由で拒絶されたものも多いです。

明らかなバグを仕様だと言い張る

たとえば、Re:VIEW では箇条書きの項目が複数行だと勝手に結合されるというバグがあります。

▼サンプル

```
* AA AA
  BB BB
```

[*6] https://www.w3.org/TR/jlreq/ja/

[*7] https://github.com/abenori/jlreq

[*8] ここに挙げた機能のうち、見た目のデザインを変更するのは（LaTeX の知識さえあれば）ユーザでも変更できますし、実際に Techbooster テンプレートでは見た目のデザインを変更しています。しかしそれ以外の機能は Re:VIEW のソースコードを変更しなければ実現できず、ユーザが簡単に行えることではありません。

```
CC CC
```

▼表示結果

- AA AABB BBCC CC

「AABB」や「BBCC」のように英単語が結合されていますよね？ どう見ても Re:VIEW のバグなのですが、報告しても「これは仕様だ」と言い張るんです。しかも「なぜそのような書き方をするんだ？ 一行に書けばいいではないか」と言われる始末。かなり意味不明な対応をされました。

Starter ではこのバグは修正しています（当然です）。

提案を却下しておきながら次のバージョンで釈明なく取り入れる

Re:VIEW 2.x では、LaTeX スタイルファイルが「sty/reviewmacro.sty」しかなく、カスタマイズするにはこのファイルを編集するのが一般的でした。しかしこの方法には問題があります。

- 「sty/reviewmacro.sty」を直接編集すると、Re:VIEW のバージョンアップをするときに困る。
- 自分でスタイルファイルを追加できるが、そのためには設定ファイルの項目を変更する必要があり、初心者には敷居が高い（どの項目をどう変更すればいいか知らないため）。

そこで、あらかじめ空のスタイルファイルを用意し、ユーザのカスタマイズはそのファイルの中で行うことを提案しました。Starter でいうと「sty/mystyle.sty」のことですね。実装は簡単だし、問題の解決策としてとても妥当な方法です。

しかしこの提案は、あーだこーだと理由をつけられて却下されました（これに限らず、初心者への敷居を下げるための提案は却下されることが多い印象です）。仕方ないので、Starter では初期の頃から独自に空のスタイルファイルを提供していました。

ところが、なんと Re:VIEW 3.0 でこの機能が取り込まれていたのです！「提案が通ったならそれでいいじゃないか」と心ないことを言う人もいるでしょうけど、いろいろ理由をつけて提案を却下しておきながら、何の釈明もなくしれーっと取り込むのは、却下されたほうとしてはたまったものではありません。不満が残るのは当たり前です。

仲のいい人とそうでない人とで露骨に態度を変える

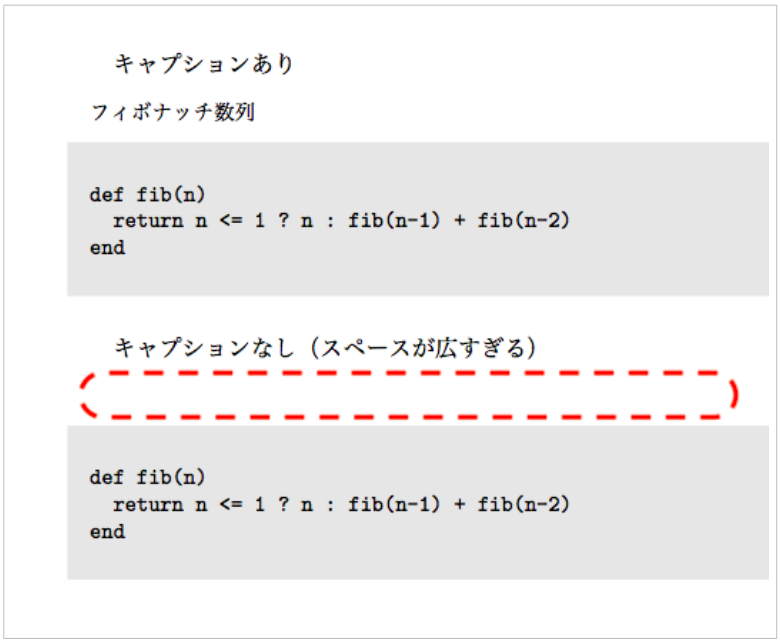
昔の Re:VIEW では、「`//list`」においてキャプション（説明文）を指定しなくても言語指定をすると、本文とプログラムリストの間が大きく空いてしまうというバグがありました。たとえばこのように書くと：

▼サンプル

```
//list[][][ruby]{
def fib(n)
  return n <= 1 ? n : fib(n-1) : fib(n-2)
end
```

```
//}
```

図 C.2 の下のように表示されていました。



▲図 C.2: 本文とプログラムリストの間が大きく空いてしまう（下）

この現象は、空文字列がキャプションとして使われるため、その分の空行が空いてしまうことが原因です。そこで、キャプションが空文字列のときは表示しないようにする修正を報告しました。

しかしこのバグ報告も、後方互換性のために却下されました。「出力結果が変わってしまうような変更は、たとえバグ修正だとしても、メジャーバージョンアップ以外では受けつけられない」というのが理由でした。こんな明白なバグの互換性なんかいらないはずだと思ったので、粘って交渉したのですが、互換性を理由に頑なに拒まれました。

ところが、開発者と仲のいい別の人が「やはりバグではないか？」とコメントした途端、メジャーバージョンアップでもないのに修正されました。見事な手のひら返しです。

「出力結果が変わってしまう変更は、たとえバグ修正でも（メジャーバージョンアップ以外では）受けつけられない。それが組版ソフトだ」という理由でさんざん拒絶しておいて、いざ別の人が言及するとすぐに修正する。出力結果が変わってしまう修正だというのに！

大事な点なので強調しますが、**出力結果が変わる変更はダメと言って拒絶しておきながら、別の人が「やはりバグでは？」と言うと出力結果が変わる変更でもあっさり行うという、露骨な態度の違い**。よく平気でこんなことできるなと逆に感心しました。

こんなことが積み重なったので、もうバグ報告も機能提案もしないことにしました。Re:VIEW のコードは品質が良くないので、ソースコードを読んでいると細かいバグがちょこちょこ見つかり

ますが、もうシラネ。

また Re:VIEW のリリースノートを見ると、Starter の機能が Re:VIEW の新機能として取り込まれているのを見かけます。つまり、**Re:VIEW に新機能提案しても通らないけど、Starter に機能を実装すると Re:VIEW に取り込まれる可能性が高い**ということです。こちらはバグ報告や提案が変な理由で却下されてストレスが溜まることから解放されるし、Re:VIEW 側は Starter が先行実装した機能を選んで取り込めばいいし、Win-Win で、これでいいのだ。

今後の開発

- Starter は、将来的に Re:VIEW のソースコードをすべて上書きするでしょう。**大事なのはユーザが書いた原稿であって、Re:VIEW でも Starter でもありません**。たとえ Re:VIEW のソースコードをすべて捨てたとしても、ユーザの原稿は使えるようになるはずです。
- 設定は、今は設定ファイルを手作業で変更していますが、これは GUI による設定に置き換わるでしょう。そのため設定ファイルの互換性はなくなる可能性が高いです。
- PDF 生成のための組版ツールとして、LaTeX 以外に Vivliostyle をサポートしたいと考えています。ただし Vivliostyle プロジェクトが独自のドキュメント作成ツールを開発中なので、Vivliostyle を必要とする人はそちらを使うだろうから、Starter での優先順位は高くなくてもいいかなと思っています。
- 同人イベントがオンラインに移行することから、PDF より ePub の需要が高くなるでしょう。今の Starter は ePub 対応が弱いので、強化する必要があります。
- 印刷物での配布が減るので、カラー化がいっそう進むでしょう。Starter はコードハイライトができないので、対応を急ぐ必要があります。
- Visual Studio Code での執筆を支援するプラグインが必要とされるでしょう。そのための本を買ったけど、積ん読のままです。
- Markdown ファイルの対応は、パーサを書き換えたあとで検討します。
- 諸般の事情により Git リポジトリを公開していないので、公開できるよう準備を進めます。

あとがき / おわりに

いかがだったでしょうか。感想や質問は随時受けつけています。

著者紹介

カウプラン機関極東支部 (@_kauplan)

“賞味期限が 1 年にも満たないようなツールやフレームワークに振り回されるのを許しておけるほど、我々の人生は長くはない。”

- `https://kauplan.org/`
- 『パンプキン・シザーズ』推し
- 『ワールド・トリガー』推し
- 『プリンセス・プリンシパル』推し

既刊一覧

- 『SQL 高速化 in PostgreSQL』（技術書典 2）
- 『オブジェクト指向言語解体新書』（技術書典 3）
- 『jQuery だって複雑なアプリ作れるもん！ 』（技術書典 4）
- 『Shell スクリプトでサーバ設定を自動化する本』（技術書典 5）
- 『Ruby のエラーメッセージが読み解けるようになる本』（技術書典 6）
- 『わかりみ SQL』（技術書典 7）
- 『Python の黒魔術』（技術書典 8）
- 『カウプランレポート vol.01』（技術書典 10）

課長のせいで今日も死んでます。楽して覚える情報技術

2025 年 5 月 30 日　ver 1.0 （技術書典 18）

著　者　株式会社おもしろテクノロジー
印刷所　日光企画

(powered by Re:VIEW Starter)