報道各位

2020 年 5 月 7 日
株式会社インプレスR&D
https://nextpublishing.jp/

Google が開発した API 規格で、スキーマ駆動開発に入門しよう！

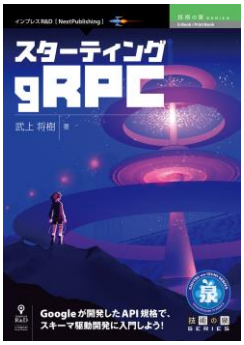
『スターティング gRPC』発行

技術の泉シリーズ、5 月の新刊

インプレスグループで電子出版事業を手がける株式会社インプレスR&D は、『スターティング gRPC』(著者:武上 将樹)を発行いたします。

最新の知見を発信する『技術の泉シリーズ』は、「技術書典」や「技術書同人誌博覧会」をはじめとした各種即売会や、勉強会・LT 会などで頒布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。

『スターティングgRPC』
https://nextpublishing.jp/isbn/9784844378457



著者:武上 将樹
小売希望価格:電子書籍版 1600 円(税別)／印刷書籍版 2000 円(税別)
電子書籍版フォーマット:EPUB3／Kindle Format8
印刷書籍版仕様:B5 判／カラー／本文 166 ページ
ISBN:978-4-8443-7845-7
発行:インプレス R&D

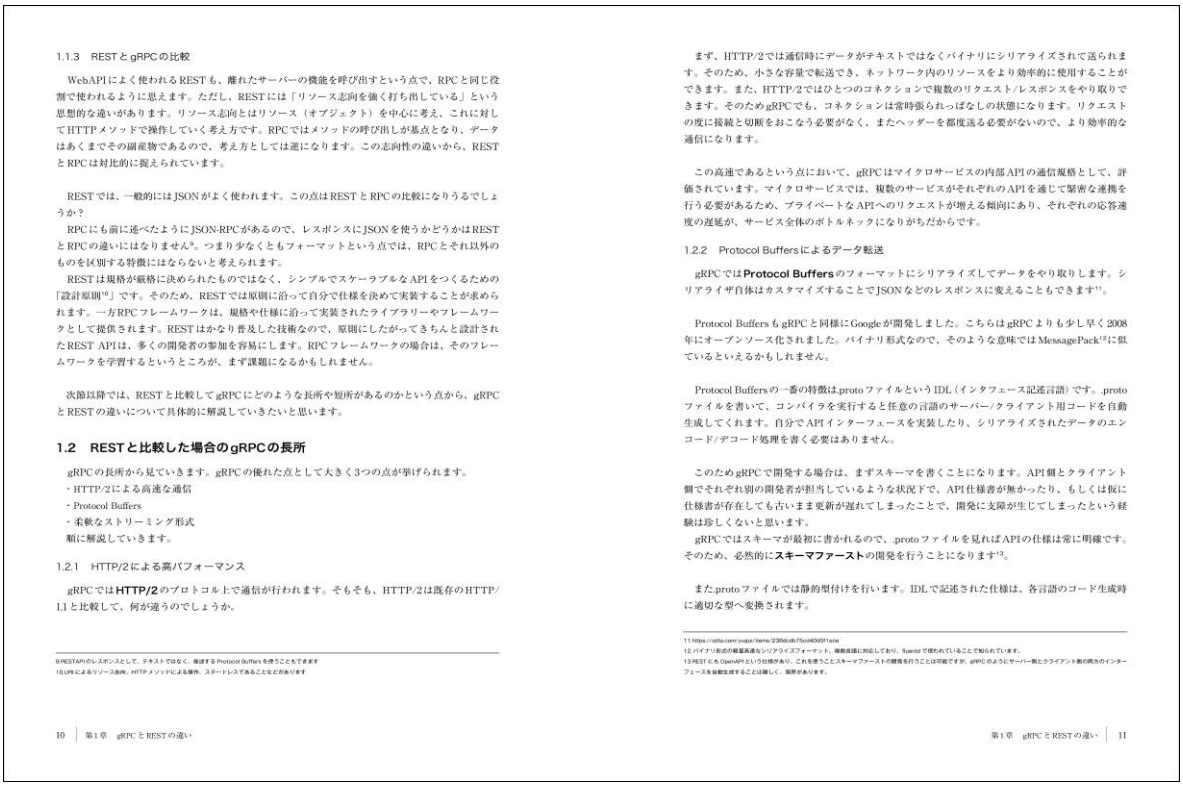
＜＜発行主旨・内容紹介＞＞

gRPC は Google が開発した高速な API 通信とスキーマ駆動開発を実現する RPC フレームワークであり、マイクロサービス間の内部通信を実現する有力な選択肢として活用されはじめています。

本書ではサーバー側(Go)/ クライアント側(Ruby)と異なる言語を用いて、いくつかのサンプルアプリケーションを実装しながら、gRPC と REST の違い、Protocol Buffers におけるスキーマの文法、単純な RPC から双方向ストリーミング RPC までの gRPC における基本的な実装方法などを平易に説明します。

（本書は、次世代出版メソッド「NextPublishing」を使用し、出版されています。）

gRPC と REST の違いを解説



1.1.3　REST と gRPC の比較

WebAPI によく使われる REST も、離れたサーバーの機能を呼び出すという点で、RPC と同じ役割で使われるように思えます。ただし、REST には「リソース志向を強く打ち出している」という思想的な違いがあります。リソース志向とはリソース（オブジェクト）を中心に考え、これに対して HTTP メソッドで操作していく考え方です。RPC ではメソッドの呼び出しが基点となり、データはあくまでその副産物であるので、考え方としては逆になります。この志向性の違いから、REST と RPC は対比的に捉えられています。

REST では、一般的には JSON がよく使われます。この点は REST と RPC の比較になりうるでしょうか？
RPC にも前に述べたように JSON-RPC があるので、レスポンスに JSON を使うかどうかは REST と RPC の違いにはなりません[9]。つまり少なくともフォーマットという点では、RPC とそれ以外のものを区別する特徴にはならないと考えられます。
REST は規格が厳格に決められたものではなく、シンプルでスケーラブルな API をつくるための「設計原則[10]」です。そのため、REST では原則に沿って自分で仕様を決めて実装することが求められます。一方 RPC フレームワークは、規格や仕様に沿って実装されたライブラリーやフレームワークとして提供されます。REST はかなり普及した技術なので、原則にしたがってきちんと設計された REST API は、多くの開発者の参加を容易にします。RPC フレームワークの場合は、そのフレームワークを学習するというところが、まず課題になるかもしれません。

次節以降では、REST と比較して gRPC にどのような長所や短所があるのかという点から、gRPC と REST の違いについて具体的に解説していきたいと思います。

1.2　RESTと比較した場合のgRPCの長所

gRPC の長所から見ていきます。gRPC の優れた点として大きく3つの点が挙げられます。
・HTTP/2による高速な通信
・Protocol Buffers
・柔軟なストリーミング形式
順に解説していきます。

1.2.1　HTTP/2による高パフォーマンス

gRPC では**HTTP/2** のプロトコル上で通信が行われます。そもそも、HTTP/2は既存の HTTP/1.1 と比較して、何が違うのでしょうか。

9 RESTAPIのレスポンスとして、テキストではなく、後述する Protocol Buffers を使うこともできます
10 URI による リソース操作、HTTP メソッドによる操作、ステートレスである ことなどがあります

まず、HTTP/2では通信時にデータがテキストではなくバイナリにシリアライズされて送られます。そのため、小さな容量で転送でき、ネットワーク内のリソースをより効率的に使用することができます。また、HTTP/2ではひとつのコネクションで複数のリクエスト/レスポンスをやり取りできます。そのためgRPCでも、コネクションは常時張られっぱなしの状態になります。リクエストの度に接続と切断をおこなう必要がなく、またヘッダーを都度送る必要がないので、より効率的な通信になります。

この高速であるという点において、gRPC はマイクロサービスの内部 API の通信規格として、評価されています。マイクロサービスでは、複数のサービスがそれぞれの API を通じて緊密な連携を行う必要があるため、プライベートな API へのリクエストが増える傾向にあり、それぞれの応答速度の遅延が、サービス全体のボトルネックになりがちだからです。

1.2.2　Protocol Buffersによるデータ転送

gRPC では**Protocol Buffers** のフォーマットにシリアライズしてデータをやり取りします。シリアライザ自体はカスタマイズすることでJSON などのレスポンスに変えることもできます[11]。

Protocol Buffers も gRPC と同様に Google が開発しました。こちらは gRPC よりも少し早く 2008 年にオープンソース化されました。バイナリ形式なので、そのような意味では MessagePack[12]に似ているといえるかもしれません。

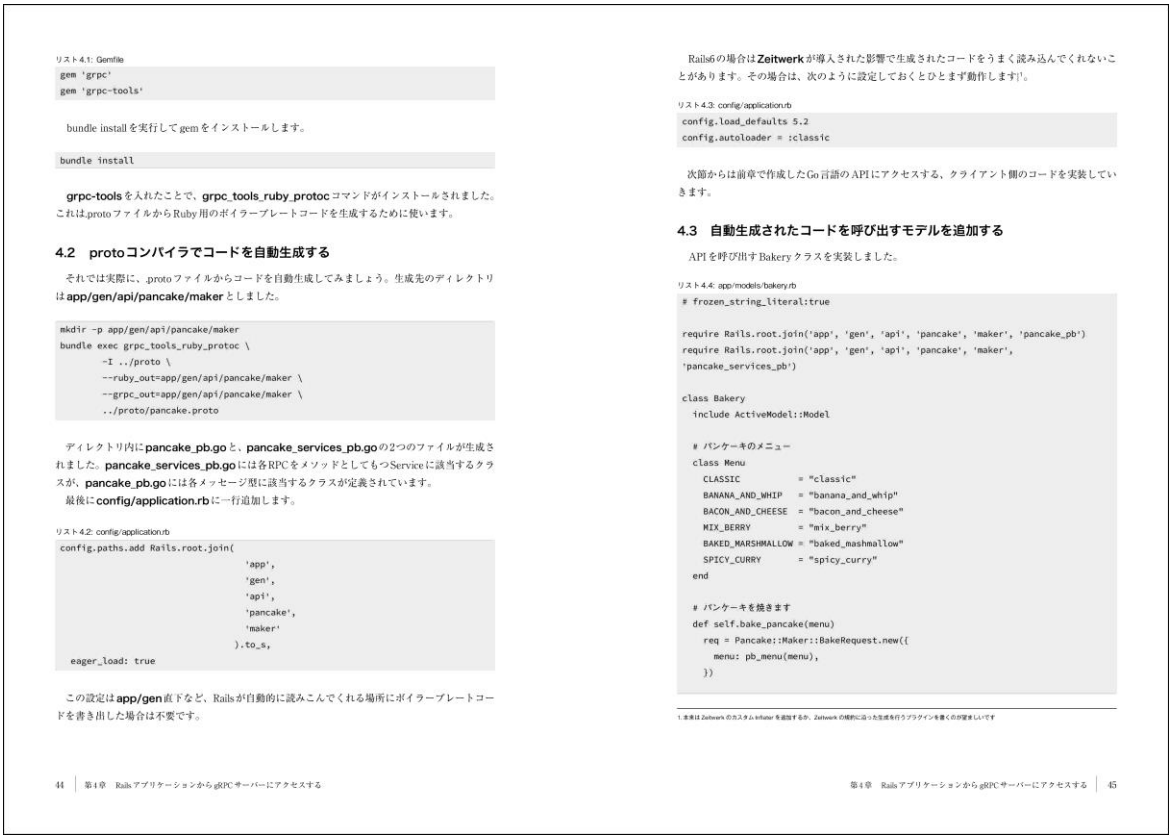
Protocol Buffers の一番の特徴は.proto ファイルという IDL（インタフェース記述言語）です。.proto ファイルを書いて、コンパイラを実行すると任意の言語のサーバー/クライアント用コードを自動生成してくれます。自分で API インターフェースを実装したり、シリアライズされたデータのエンコード/デコード処理を書く必要はありません。

このため gRPC で開発する場合は、まずスキーマを書くことになります。API 側とクライアント側でそれぞれ別の開発者が担当しているような状況下で、API 仕様書が無かったり、もしくは仮に仕様書が存在しても古いまま更新が遅れてしまったことで、開発に支障が生じてしまったという経験は珍しくないと思います。
gRPC ではスキーマが最初に書かれるので、.proto ファイルを見れば API の仕様は常に明確です。そのため、必然的に**スキーマファースト**の開発を行うことになります[13]。

また.proto ファイルでは静的型付けを行います。IDL で記述された仕様は、各言語のコード生成時に適切な型へ変換されます。

11 https://qiita.com/yuqui/items/236dcdb75cd40d0f11ece
12 バイナリ形式の軽量高速なシリアライズフォーマット。複数言語に対応しており、fluentd で使われていることで知られています。
13 REST にも OpenAPI という仕様があり、これを使うことでスキーマファーストの開発を行うことは可能ですが、gRPC のようにサーバー側とクライアント側の両方のインターフェースを自動生成することは難しく、限界があります。

Rails アプリから gRPC サーバーにアクセスする方法を紹介



リスト 4.1: Gemfile
```
gem 'grpc'
gem 'grpc-tools'
```

bundle install を実行して gem をインストールします。

```
bundle install
```

grpc-tools を入れたことで、**grpc_tools_ruby_protoc** コマンドがインストールされました。これは.proto ファイルから Ruby 用のボイラープレートコードを生成するために使います。

4.2　protoコンパイラでコードを自動生成する

それでは実際に、.proto ファイルからコードを自動生成してみましょう。生成先のディレクトリは**app/gen/api/pancake/maker** としました。

```
mkdir -p app/gen/api/pancake/maker
bundle exec grpc_tools_ruby_protoc \
        -I ../proto \
        --ruby_out=app/gen/api/pancake/maker \
        --grpc_out=app/gen/api/pancake/maker \
        ../proto/pancake.proto
```

ディレクトリ内に**pancake_pb.go** と、**pancake_services_pb.go** の2つのファイルが生成されました。**pancake_services_pb.go** には各RPCをメソッドとしてもつServiceに該当するクラスが、**pancake_pb.go** には各メッセージ型に該当するクラスが定義されています。
最後に**config/application.rb** に一行追加します。

リスト 4.2: config/application.rb
```
config.paths.add Rails.root.join(
                                 'app',
                                 'gen',
                                 'api',
                                 'pancake',
                                 'maker'
                                ).to_s,
  eager_load: true
```

この設定は**app/gen** 直下など、Railsが自動的に読みこんでくれる場所にボイラープレートコードを書き出した場合は不要です。

Rails6の場合は**Zeitwerk** が導入された影響で生成されたコードをうまく読み込んでくれないことがあります。その場合は、次のように設定しておくとひとまず動作します[1]。

リスト 4.3: config/application.rb
```
config.load_defaults 5.2
config.autoloader = :classic
```

次節からは前章で作成した Go 言語の API にアクセスする、クライアント側のコードを実装していきます。

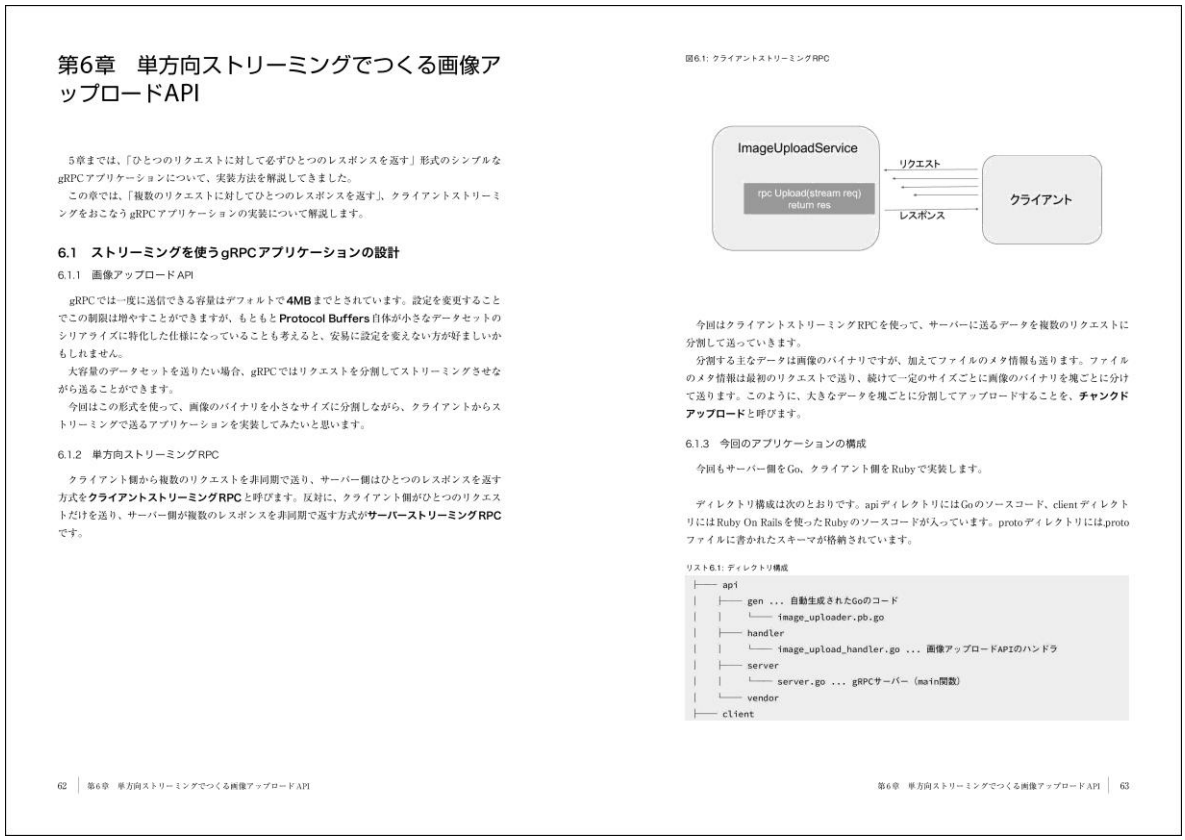
4.3　自動生成されたコードを呼び出すモデルを追加する

API を呼び出す Bakery クラスを実装しました。

リスト 4.4: app/models/bakery.rb
```
# frozen_string_literal:true

require Rails.root.join('app', 'gen', 'api', 'pancake', 'maker', 'pancake_pb')
require Rails.root.join('app', 'gen', 'api', 'pancake', 'maker',
'pancake_services_pb')

class Bakery
  include ActiveModel::Model

  # パンケーキのメニュー
  class Menu
    CLASSIC           = "classic"
    BANANA_AND_WHIP   = "banana_and_whip"
    BACON_AND_CHEESE  = "bacon_and_cheese"
    MIX_BERRY         = "mix_berry"
    BAKED_MARSHMALLOW = "baked_mashmallow"
    SPICY_CURRY       = "spicy_curry"
  end

  # パンケーキを焼きます
  def self.bake_pancake(menu)
    req = Pancake::Maker::BakeRequest.new({
      menu: pb_menu(menu),
    })
```

1 本来は Zeitwerk のカスタム Inflator を追加するか、Zeitwerk の規約に沿った生成を行うプラグインを書くのが望ましいです

画像アップロード API やリアルタイムリバーシの制作を通じて学ぶ



第6章　単方向ストリーミングでつくる画像アップロードAPI

5章までは、「ひとつのリクエストに対して必ずひとつのレスポンスを返す」形式のシンプルなgRPCアプリケーションについて、実装方法を解説してきました。
この章では、「複数のリクエストに対してひとつのレスポンスを返す」、クライアントストリーミングをおこなうgRPCアプリケーションの実装について解説します。

6.1　ストリーミングを使うgRPCアプリケーションの設計

6.1.1　画像アップロードAPI

gRPCでは一度に送信できる容量はデフォルトで**4MB**までとされています。設定を変更することでこの制限は増やすことができますが、もともと**Protocol Buffers**自体が小さなデータセットのシリアライズに特化した仕様になっていることも考えると、安易に設定を変えない方が好ましいかもしれません。
大容量のデータセットを送りたい場合、gRPCではリクエストを分割してストリーミングさせながら送ることができます。
今回はこの形式を使って、画像のバイナリを小さなサイズに分割しながら、クライアントからストリーミングで送るアプリケーションを実装してみたいと思います。

6.1.2　単方向ストリーミングRPC

クライアント側から複数のリクエストを非同期で送り、サーバー側はひとつのレスポンスを返す方式を**クライアントストリーミングRPC**と呼びます。反対に、クライアント側がひとつのリクエストだけを送り、サーバー側が複数のレスポンスを非同期で返す方式が**サーバーストリーミングRPC**です。

62　第6章　単方向ストリーミングでつくる画像アップロードAPI

図6.1: クライアントストリーミングRPC

今回はクライアントストリーミングRPCを使って、サーバーに送るデータを複数のリクエストに分割して送っていきます。
分割する主なデータは画像のバイナリですが、加えてファイルのメタ情報も送ります。ファイルのメタ情報は最初のリクエストで送り、続けて一定のサイズごとに画像のバイナリを塊ごとに分けて送ります。このように、大きなデータを塊ごとに分割してアップロードすることを、**チャンクドアップロード**と呼びます。

6.1.3　今回のアプリケーションの構成

今回もサーバー側をGo、クライアント側をRubyで実装します。

ディレクトリ構成は次のとおりです。apiディレクトリにはGoのソースコード、clientディレクトリにはRuby On Railsを使ったRubyのソースコードが入っています。protoディレクトリには.protoファイルに書かれたスキーマが格納されています。

リスト6.1: ディレクトリ構成

```
├── api
|   ├── gen ... 自動生成されたGoのコード
|   |   └── image_uploader.pb.go
|   ├── handler
|   |   └── image_upload_handler.go ... 画像アップロードAPIのハンドラ
|   ├── server
|   |   └── server.go ... gRPCサーバー（main関数）
|   └── vendor
├── client
```

第6章　単方向ストリーミングでつくる画像アップロードAPI　63

＜＜目次＞＞

＜＜著者紹介＞＞

武上 将樹
株式会社MERY開発部所属サーバーサイドエンジニア。Go/Rubyで開発していますが、一番好きな言語はTypeScirptです。

＜＜販売ストア＞＞

電子書籍:
Amazon Kindle ストア、楽天 kobo イーブックストア、Apple Books、紀伊國屋書店 Kinoppy、Google Play Store、honto 電子書籍ストア、Sony Reader Store、BookLive!、BOOK☆WALKER
印刷書籍:

Amazon.co.jp、三省堂書店オンデマンド、honto ネットストア、楽天ブックス

※　各ストアでの販売は準備が整いしだい開始されます。

※　全国の一般書店からもご注文いただけます。

【インプレス R&D】 https://nextpublishing.jp/

株式会社インプレス R&D（本社：東京都千代田区、代表取締役社長：井芹昌信）は、デジタルファーストの次世代型電子出版プラットフォーム「NextPublishing」を運営する企業です。また自らも、NextPublishingを使った「インターネット白書」の出版など IT 関連メディア事業を展開しています。

※NextPublishing は、インプレス R&D が開発した電子出版プラットフォーム（またはメソッド）の名称です。電子書籍と印刷書籍の同時制作、プリント・オンデマンド（POD）による品切れ解消などの伝統的出版の課題を解決しています。これにより、伝統的出版では経済的に困難な多品種少部数の出版を可能にし、優秀な個人や組織が持つ多様な知の流通を目指しています。

【インプレスグループ】 https://www.impressholdings.com/

株式会社インプレスホールディングス（本社：東京都千代田区、代表取締役：唐島夏生、証券コード：東証 1 部 9479）を持株会社とするメディアグループ。「IT」「音楽」「デザイン」「山岳・自然」「旅・鉄道」「学術・理工学」を主要テーマに専門性の高いメデイア&サービスおよびソリューション事業を展開しています。さらに、コンテンツビジネスのプラットフォーム開発・運営も手がけています。

【お問い合わせ先】

株式会社インプレス R&D NextPublishing センター
TEL 03-6837-4820
電子メール: np-info@impress.co.jp