いろんなちほーがまざったほん
開発がとくいなフレンズになる方法
- テーマは十人十色! -

たかしファンクラブ　著

2017-04-29 版　　発行

はじめに

本書は有志で集まってテーマを一つに絞らず、それぞれが好きなことを書くという趣向のもと執筆しました。その結果、思ったよりバラバラなテーマになりましたが（笑）普段それぞれ好きでやってることが違うので当然だな、あとで気付きました。いや、逆に 1 つの本で色々なことを知ることができていいんじゃないか？ と開き直り初の書籍制作となりましたが、それでも各章で面白い内容になりました。ぜひ興味を持って楽しんでいただければと思います。

（発行代表 @roana0229）

本書の内容

本書は下記の内容によって構成されています。

- GoogleAppsScript のライブラリ開発について
- SQL で書くいろいろなグラフ
- 機械学習によるフレンズ語翻訳機
- PHP の OR マッパーについて
- iOS アプリ開発の色々

免責事項

本書に記載された内容は、情報の提供のみを目的としています。したがって、本書を用いた開発、製作、運用は、必ずご自身の責任と判断によって行ってください。これらの情報による開発、製作、運用の結果について、著者はいかなる責任も負いません。

目次

第 1 章

GoogleAppsScript ライブラリ開発

みなさん GoogleAppsScript[*1]（以下、GAS）は使ったことがありますか？ GAS は簡単に言うと、Google のサーバ上で JavaScript を実行できるサービスです。外部に API として公開し POST,GET のリクエストを受けたり、Google Spreadsheets[*2]（以下、Spreadsheets）や Google FusionTables[*3]（以下、FusionTables）でデータを保持することもできます。個人のアプリでバックエンドに GAS を利用していて、DB に Spreadsheets を利用し、ライブラリを開発した際に得られた知見と成果の内容です。

1.1 きっかけと GoogleAppsScript から利用する DB の選択

結論から言うと、FusionTables より Spreadsheets を利用することをオススメします。FusionTables のメリットは GoogleMap との連携やチャート化などデータをビジュアル化しやすいというところです。良さそうなのになぜ使わないのか、その理由としては、FusionTables が SELECT 句以外で WHERE 句が利用できないことが大きいです。正確に言うと、ROW@<b>{ID というオートインクリメントされるカラムしか指定できません。削除したい対象を SELECT で抽出してから、ROW}ID を指定して DELETE することは可能です。しかし、書き込みの上限が 30 件/分かつ 5000 件/日と厳しく、リクエストを送るクライアントが多いと現実的ではありません。

Spreadsheets メリットとしては非エンジニアでも触れるだと思います。例えば、

[*1] https://developers.google.com/apps-script
[*2] https://docs.google.com/spreadsheets
[*3] https://developers.google.com/fusiontables

- ボタンやプッシュ通知の A/B パターン文言
- 動的なテンプレートメッセージ

などを文言やパターンを変更する際に、エンジニアを介さず変更を行うことができるます。そんな気軽に変更されてはたまらん！ というデータはシートを分けておき、編集可能者を制限しておくと良いでしょう。それぞれに容易に変更できる環境でありながら実行権限を付与できることもメリットです。

良いところしかないのかと言うとそうではありません。GAS で Spreadsheets を利用する際に使う Spreadsheet Service[*4]が提供しているメソッドでは、SQL を利用してデータを抽出することができません。そこで SpreadSheetsSQL[*5] という Spreadsheets から SQL ライクにデータを抽出できる GAS のライブラリを公開しています。ライブラリを使うメリットとしては **Spreadsheets** のセルを意識する必要がなくなるというところにあります。もちろん、このライブラリなしでも DB として利用することは可能です。常に固定のセルを取得するだけなら、直に Spreadsheet Service を利用した方が良いと思います。

1.2 GAS ライブラリの開発

GAS ライブラリは GAS で作ることができます。SpreadsheetsSQL のソースコードを例に説明します。まず初めにこのライブラリを使う時は以下のように使います。

▼

```
var result = SpreadSheetsSQL.open(SHEET_ID, SHEET_NAME).select(['name', 'cv']).result();

// result の内容
[{
  name: ' サーバルちゃん',
  cv: ' 尾崎由香'
},{
  name: ' かばんちゃん',
  cv: ' 内田彩'
}]
```

SpreadsheetsSQL を利用する際には、`SpreadSheetsSQL.open()`を呼ぶ必要があります。これは**ライブラリのプロジェクト名.メソッド**という形になっていて

▼

[*4] https://developers.google.com/apps-script/reference/spreadsheet/
[*5] https://github.com/roana0229/spreadsheets-sql

```
/**
 * This method use to create SpreadSheetsSQL instance.
 * @param {String} id SpreadSheet id
 * @param {String} name SpreadSheet sheet name
 * @return {SpreadSheetsSQL} SpreadSheetsSQL instance
 */
function open(id, name) {
  return new SpreadSheetsSQL_(id, name);
}
```

コードではこのようになっています。コメントは JSDoc[*6]で記述することができます。`new SpreadSheetsSQL_(id, name)`で `SpreadSheetsSQL_`というクラスをインスタンス化していることがわかると思います。また、クラス名の末尾にアンダースコアがついていることにお気付きかと思います。これはスコープ制御をするためでありライブラリ内でのみ参照できるプライベートな状態になります。`SpreadSheetsSQL.open()`した際の戻り値は `SpreadSheetsSQL_`型ですが、JSDoc にはアンダースコアなしで記述されています。実際にはプライベートクラスのインスタンスが返っているが、JSDoc を見るとプロジェクト名のクラスが返るということです。

次に取得するカラムの指定に `SpreadSheetsSQL.open().select()`を呼ぶ必要があります。

▼

```
/**
 * This method use to get columns.
 * <pre><code>Example: SpreadSheetsSQL.open(id, name).select(['name', 'age', 'married', 'company']).r
 * </pre></code>
 * @param {String[]} selects column array
 * @return {SpreadSheetsSQL} SpreadSheetsSQL instance
 */
function select(selects) {
  throw new Error("it's a mock method for content assist");
}
```

コードではこのようになっています。`result()`でも同様のコードが記述されています。呼び出されたら例外を投げるメソッドに見えます。しかし、ここで宣言されているメソッドはあくまでライブラリ外から参照するためのメソッドでしかありません。実態は

▼

```
class SpreadSheetsSQL_ {
  ...
  select(selects) {...}
  ...
}
```

[*6] http://usejsdoc.org

のように SpreadSheetsSQL_クラス内に記述されています。通常ではプライベートなクラスのメソッドは外からは見えず呼び出すことができません。メソッドが見えないということは呼び出すことができないということになっていまします。そのため、ライブラリのライブラリのルート階層にインターフェースとして記述し、プライベートなメソッドでも GAS のオンラインエディタでの補完を有効にしています。

まとめるとこのようになります。

- プロジェクト名. メソッドという形で呼び出すことができる。
- 末尾にアンダースコアでライブラリ外からは見えないようにする。
- 補完のためのインターフェースをルート階層に列挙し、JSDoc の@return にはライブラリ名を指定する。

1.3 GAS ライブラリの公開

開発もそこまで気をつけることはありませんでしたが、公開はもっと簡単です。

1. GAS のライブラリ化したいプロジェクトを開く。
2.「ファイル」->「版を管理」に移動して、新しいバージョンを保存する。
3.「ファイル」->「プロジェクトのプロパティ」で表示される、スクリプト ID を共有する。

たったこれだけで、他人がでライブラリとして GAS プロジェクトを扱うことができます。利用できるとは言ってもドキュメントが必要ですよね？ 実は JSDoc を書いていれば、https://script.google.com/macros/library/d/スクリプト ID/バージョン で見ることができます。公式がきちんとドキュメントまでサポート[*7]しているのはとても嬉しいことですね。

1.4 おまけ: ES6 + ローカルで開発する

GAS は個人でサーバを用意せず、気軽に開発できて便利です。ただ、オンラインエディタ上でしか書けないため、普段使いしている慣れた環境で開発することができません。そこで Babel + gulp + node-google-apps-script を使うことで、ES6 かつローカルで開発しています。

ローカルのファイルをアップロードするための選択肢としていくつか方法があります

[*7] 執筆時点 2017/04/16 時点では CSS が読み込めない状態になっています。

が、今回は GoogleDevelopersJapan で過去に紹介された node-google-apps-script[*8] を利用します。セットアップは URL 先に譲ります。

また、GAS は ES6 に対応していないため、ES6 で記述した JavaScript をそのまま実行することはできません。そのため https://babeljs.io を利用して、実行可能な JavaScript のコードに変換します。単純に Babel を利用して変換しても良いですが、コードを変更したら自動で変換 + 同期できるように、gulp を利用します。

1. node-google-apps-script のセットアップ
2. gulp + babel のインストール `npm install -g gulp && npm install --save-dev gulp gulp-babel babel-preset-es2015`
3. es6 というディレクトリを作って、`コード.js`を作成
4. gulpfile.js を作成

これで準備完了です。`gulp babel`と実行すると、src/コード.js に変換された JavaScript が出力され、`gapps upload`することでコード.js が反映されます。`gulp`と実行すると、変更監視状態になり es6 のコードを変更したら、Babel で変換し、アップロードまでを自動的に行います。

▼

```
.
├── gapps.config.json
├── gulpfile.js
├── node_modules
├── es6
│   └── コード  .js
└── src
    └── コード.js
```

▼

```
var gulp = require('gulp');
var babel = require('gulp-babel');
var spawn = require('child_process').spawn;

gulp.task('default', ['watch', 'upload']);

gulp.task('watch', () => {
  gulp.watch('es6/*.js', ['babel']);
});

gulp.task('babel', () => {
  return gulp.src('es6/*.js')
    .pipe(babel({
```

[*8] https://github.com/danthareja/node-google-apps-script

```
            presets: ['es2015']
        }))
    .pipe(gulp.dest('src'));
});

gulp.task('upload', () => {
  gulp.watch('src/*.js', function() {
    spawn('gapps', ['upload']);
    console.log('finished `gapps upload`');
  });
});
```

▼

```
class Bot {
  constructor(message) {
    this.message = message;
  }

  say() {
    Logger.log(this.message);
  }
}

function main() {
  var bot = new Bot("ようこそ技術書典へ");
  bot.say();
}
```

これでローカルかつ ES6 で開発しながら、GAS に実行可能な状態で同期することができました。

1.5 さいごに

どうでしたか？ GoogleAppsScript が大きく目立つことがない中、更にそのライブラリを作るというニッチなところまとめてみました。GoogleAppsScript は API+DB としてだけではなく、cron のように定期的に処理を実行することも可能です。また、個人ではなかなかサーバ側に手が出せないという人や、ちょっとした便利ツールを作る時にとても便利だと思っています。

第 2 章

PHP の OR マッパーを調べてみた 2017

2.1　ぴーえっちぴーちほーで元気に暮らす老害おじちゃん！

　こんにちは。 @boscoworks です。普段は目黒あたりで PHP エンジニアをやっています。
最近は Ruby on Rails での Web 開発が主流になってきましたね。仕事柄、社外のエンジニアの方ともよくお会いするのですが、Rails をやっているエンジニアはとても多い印象です。
私の周りでもたまに「**PHP** はオワコン」という言葉が飛び交っていて「そろそろエンジニアから足を洗うときが近づいているかな」などと冗談めかして言ったりしています。来年あたり本当にやめているかもしれない。
さて、とはいえ私自身も私の周辺もまだまだ PHP は活況、PHP+MySQL は未だ Web 開発の王道を行っていると断言して差し支えないでしょう。
ただ PHP での Web 開発をやっていて（個人的に）いつも悩むのが OR マッパーの選定です。ぐるぐる悩んだ結果「ええいもう PDO で良くないかコレ！」と叫んだこともしばしば。
結局フレームワークに標準で入ってる OR マッパーを使うのが定石なんですけどね、なんかこうモヤモヤしませんか？ しますよね？
ということで、最近の PHP の OR マッパーをあれこれ調べることにしてみましたよ。

2.2　調査対象

　独断と偏見で以下をピックアップしました。

1. Propel

- 一時代前の Symfony が標準 ORM として導入していたやつ。最近はあんまり聞かない。
  - http://propelorm.org/
  - https://github.com/propelorm/Propel2
  - https://packagist.org/packages/propel/propel

2. Doctrine

- 最近の Symfony が標準 ORM として導入しているやつ。ザ・PHP の ORM 感ある。
  - http://www.doctrine-project.org/
  - https://github.com/doctrine/doctrine2
  - https://packagist.org/packages/doctrine/orm

3. Eloquent

- Laravel の ORM。こいつが有名というよりは、Lavavel が急成長した結果、相対的に使われるようになった的な。
  - https://laravel.com/docs/5.4/eloquent
  - https://github.com/illuminate/database
  - https://github.com/laravel/framework
  - https://packagist.org/packages/illuminate/database
  - https://packagist.org/packages/laravel/framework

4. Idiorm & Paris

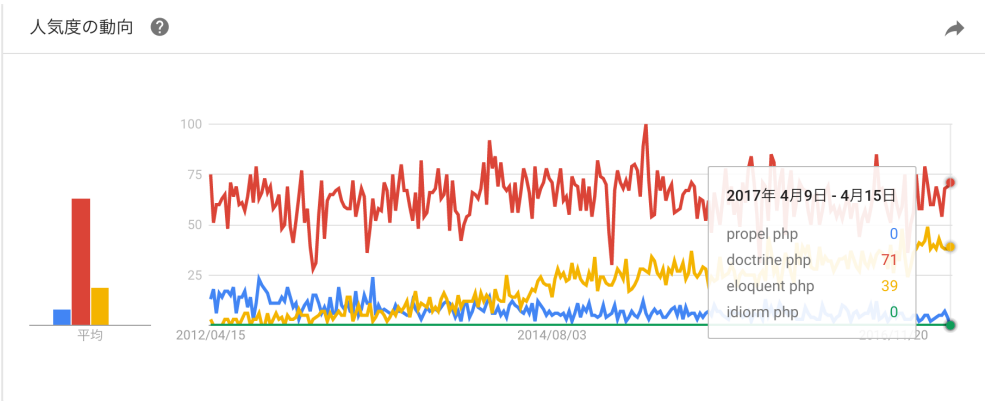
- 「PHP OR マッパー」でググると上の方に出てくるやつ。
  - http://j4mie.github.io/idiormandparis/
  - https://github.com/j4mie/idiormandparis

2.3　調査方法

3 つの視点から「どの OR マッパーが人気なのか」を探ります。

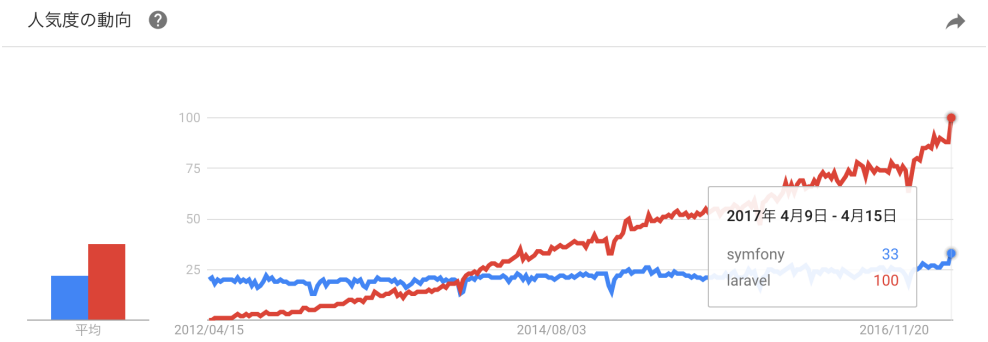
- 「Google トレンド (https://trends.google.co.jp/trends/)」による人気度の動向調査
- GitHub の Star 数、Fork 数
- Packagist 経由での composer install 数

Google トレンドによる人気度の動向調査



▲図 2.1　ORM 動向

Doctrine はさすが王者の風格、1 位を譲らないですね。
注目は Eloquent、2013 年末あたりで Propel を追い越してからも順調に右肩上がり。いまや Doctrine に並ぶ勢いですね。
Idiorm...(＾ ω ＾)
これ結局 Symfony と Laravel のシェア争いとイコールになるんじゃないかと思い調べてみると。



▲図 2.2　フレームワーク動向

Symfony はシェアを落とすことなく、ただ Laravel が怒涛の右肩上がり。Laravel 人気すごいですね。

GitHub の Star 数、Fork 数

Star 数、Fork 数が多いということは、すなわち開発が活発に行われているということ。不具合があれば直りやすいし、周辺技術への追従も比較的頻繁に行われるかもしれません。

ORM	Star 数	Fork 数
Propel	940	282
Doctrine	3,505	1,686
Eloquent(illuminate/database)	1,008	281
Eloquent(laravel/framework)	6,941	4,134
Idiorm	8	0

(※ 3/28 調べ)

Eloquent は illuminate/database というパッケージが大本になっていたのが、 laravel/framework に replace されたのですね。なので Star 数 Fork 数は分けて表示。なんか比較できない感じになってしまった。

Doctrine と Eloquent はいいとして、Propel の Star 数、Fork 数がそこそこあるのが意外です。

いまやレガシーな ORM と思いきや、しっかりメンテナーもついているんですね。

Packagist 経由での composer install 数

ORM	Install 数
Propel	295,535
Doctrine	21,048,164
Eloquent(illuminate/database)	2,262,532
Eloquent(laravel/framework)	20,991,608
Idiorm	194,306

(※ 3/28 調べ)

こうやってみると Doctrine 健闘しているんですね。 Laravel 本体といい勝負。
Idiorm は composer install はされてるのに GitHub では下火なのが印象的。興味本位でインストールして、そのままバイバイなケースが多いのかな。

2.4 閑話休題

FuelPHP の OR マッパーについて一言言わせてほしいんだ

最近、完全に私の好みで FuelPHP でプロダクトコードを書いているのですが、OR マッパーが微妙だなーと思うのは私だけでしょうか・・・。
チェーンメソッドでクエリを構築できるのは直感的に書けて良いのですが、

- SELECT クエリの結果をデフォルトでメモリにキャッシュしちゃう
- モデルクラスが oil コマンドで強制上書きなので、DDL に弱い (ただし工夫すれば回避できる)
- アンダーバー区切りでディレクトリを切る仕組みなので、N 対 N なテーブルの中間テーブルはいびつなモデル名になる
- Model_Crud とクエリビルダが共存することができる。どっちかに寄せないとすぐキメラになる

みたいな感じで、ちょっと癖のある子になっています。

FuelPHP のモデルキャッシュを削除する方法

Model**Crud** のキャッシュをクリアするには、**\Orm\Model** の **$**cached_objects を初期化してあげる必要があります。

▼

```
class Model_Article extends \Orm\Model
{
    public static cc()
    {
        static::$_cached_objects[get_called_class()] = [];
    }
}
```

クエリビルダのほうは `delete()` なるメソッドがあるのでそれを使います。

▼

```
$query = DB::query("SELECT * FROM articles")
            ->cached($cacheTtl, "articles.all", false)->execute();
Cache::delete("articles.all");
```

「DDL 発行が頻繁に起きる DB に対してのバッチ用途」で FuelPHP のモデル作成は結構運用コストがかかっています。
裏を返せば、「そんなに DDL 発行が発生しない Web アプリケーションもしくは REST API 用途」であれば、FuelPHP の OR マッパーは結構優秀なのかな。

2.5 まとめ

最後は FuelPHP の OR マッパーについての (超個人的な) 愚痴でしたが、PHP の OR マッパーについて時代の潮流を垣間見ることができたように思います。数字としてはっきり出た感じはしますね。時代は Laravel 。
Ruby 界隈で Rails が覇道を行くように、いまは PHP on Laravel 。
Laravel を使うなら OR マッパーは Eloquent を自然に使うべきだし、 Laravel を選べないなら Doctrine 。
10 年くらい前の PHP 界隈では Propel VS Doctrine だった気もするけど。たぶん 10 年後くらいには「昔 Laravel っていうフレームワークがあってだな」って言っているんでしょうね。
さて、冒頭でも少し言いましたが、最近周りが Rails エンジニアばかりなので、「いやいや時代は PHP でしょ」みたいな気骨のある PHP エンジニアな友達が欲しいです。そろそろコミュニティに参加したり、何某かのコミッターにでもならないといけないかなぁ。

第 3 章

近くのフレンズを探すアプリを作ろう！

こんにちは、masao です。普段は Rails で社内システムを作ったり、
iOS と Android のアプリ開発をしています。
今回は出会い系アプリのような近くの人の写真のサムネイルが表示される、
フレンズを探せるアプリを作っていきたいと思います。

3.1 API の準備

テーブルの用意

まずはアプリから叩く API にフレンズの位置情報を保存する DB を作成しておきます
今回はフレンズの座標 (緯度経度) を Mysql の geometry 型で保存します

▼

```
CREATE TABLE ‘members‘ (
  ‘id‘ int(10) unsigned NOT NULL AUTO_INCREMENT,
  ‘geometry‘ geometry NOT NULL,
  PRIMARY KEY (‘id‘),
  SPATIAL KEY ‘geometry‘ (‘geometry‘)
) ENGINE=MyISAM DEFAULT CHARSET=utf8;
```

geometry 型を取得する際にはバイナリが取得されるので
ASTEXT() を使って文字列として取得します

▼

```
mysql> SELECT id, ASTEXT(geometry) FROM members;
+----+---------------------------------------------+
| id | ASTEXT(geometry)                            |
+----+---------------------------------------------+
|  1 | POINT(132.132521 32.703595)  |
|  2 | POINT(113.151213 32.103456)  |
+----+---------------------------------------------+
```

クエリの準備

続いては、自分の座標から自分以外の 1km 以内のフレンズを探すクエリを作ります
使う関数は以下です

X = 経度
Y = 緯度
LINESTRING = POINT と POINT を繋いだ線のことです
GEOMFROMTEXT = geometry 型に変換できます
GLENGTH = LINESTRING の長さを取得できます

以上を使ってこんな感じのクエリを作成しましょう

▼

```
Glength(
  GeomFromText(
    'LineString(132.132521 32.703595, 113.151213 32.103456)'
  )
)
```

ただし、緯度経度はデータから取ってくるので CONCAT を使います

▼

```
Glength(
  GeomFromText(
    CONCAT('LINESTRING(',
      X( self.geometry ),' ', Y( self.geometry ), ",",
      X( other.geometry ), ' ', Y( other.geometry ), ')'
    )
  )
```

これで自分とフレンズの距離が取れました！
次はこの距離を使って 1km 以内の距離に絞り込みましょう

ここで問題なのが、Glength は度の値なので、
1km を度に変えなければいけません。

まず、 1 度が何 km なのかを求めます。
地球の外周が およそ 40,075km なので、これを 360 度で割ります。

40075 / 360 ≒ 111.319444

つまり、1 度は 111.319444 km です。111.319444km のうち 1km は何度かを求めます。

1 / 111.319444 = 0.00898316

1km は Glength にすると約 0.009 なので、これをもとに 1km 以内のフレンズを探しましょう！

▼

```
SELECT
  friends.id,
  GLENGTH(GEOMFROMTEXT(
    CONCAT('LINESTRING(',
      X( self.geometry ),' ',Y( self.geometry ), ",",
      X( friends.geometry ),' ',Y( friends.geometry ), ')'
    )
  )) AS distance
FROM
  members AS self
    INNER JOIN
  members AS friends
WHERE
  self.id = 1 AND self.id != friends.id
ORDER BY
  distance ASC
```

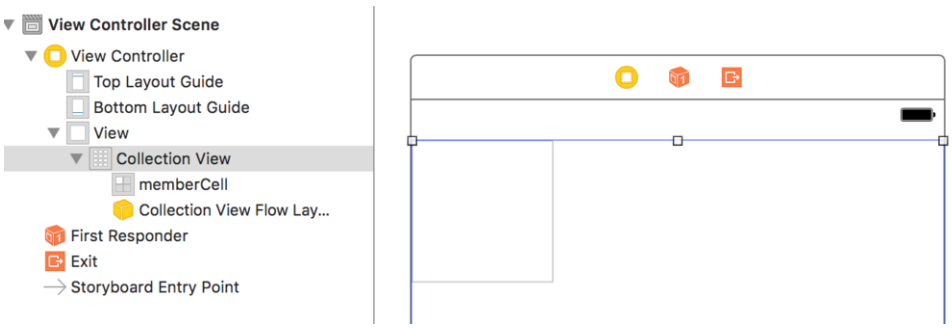
これで 1km 以内のフレンズが探せるようになったので API で叩いて取れるようにしておきましょう！ (ry

3.2 アプリの準備

今回は Swift でアプリを作っていきます。UICollectionView を使って周りにいる人たちの画像を表示しましょう！

StoryBoard

ViewController に CollectionView を設置していきます。
CollectionViewCell に identifier を設定していきます。今回は memberCell にしました。



▲図 3.1　storyboard

ViewController

ViewController で先ほど作った API を叩いてフレンズを取得します。取得したフレンズ を CollectionView に設置していきます。

▼

```
import UIKit

class ViewController: UIViewController, UICollectionViewDataSource {

    private var nearlyFriends = Array()

    override func viewDidLoad() {
        super.viewDidLoad()

        // ここに API からフレンズを取得する処理 (今回は省略)
        getNearlyFriends()
    }

    //フレンズの個数を返すメソッド
    func collectionView(_ collectionView: UICollectionView,
                        numberOfItemsInSection section: Int) -> Int
    {
        // API から取得してきたフレンズの個数を返す
        return nearlyFriends.count
    }


    //フレンズを返すメソッド
    func collectionView(_ collectionView: UICollectionView,
                        cellForItemAt indexPath: IndexPath) -> UICollectionViewCell
    {
        //CollectionView からセルを取得する。
        let cell = collectionView.dequeueReusableCell(
                    withReuseIdentifier: "memberCell", for: indexPath) as UICollectionViewCell

        // API から取得したフレンズから IndexPath 番目のデータを取得
        let friend = nearlyFriends[indexPath]

        // cell に取得したフレンズをもとに画像などを設定
        cell.backgroundView = friend.getImage()
    }
}
```

collectionView で画像を設定したり、距離を表示したりしてカスタマイズしていきましょう。

第 4 章

フレンズ語翻訳機を実装する

4.1　フレンズ語とは

フレンズが喋ってる独特の言い回しの言葉である。辛い気持ちのときでも優しく包んでくれて癒やされるのである。例を挙げると以下のようになる。おもしろーい。

フレンズ語の例

- わーい
- たーしー
- まんぞく
- えっへん
- やったぁ
- すごいよー、これ、しんたいけん
- あなたは○○が得意なフレンズなんだね
- すっごーい
- はやく、はやくー
- わたしのせいじゃないよぉ
- そんな恐ろしいことしないよ
- でもまぁ騒ぐほどのことでもないか

4.2　手法について考える

手始めにやりやすそうなところから考える。フレンズ語の文法は先を度挙げた例を見ればわかるように文法が比較的かんたんである。例えば「あなたは○○が得意なフレンズな

んだね」などある程度テンプレート化すれば作れそうなものも存在する。というわけでまずは簡単なテンプレートを作ってフレンズ語を生成してみようではないか。

フレンズのテンプレートの生成手法

ここで使用するものは今話題の深層学習などではない。まずは昔ながらの技術を使うことにしよう。日本語の文法を解析しようと思えばやはり mecab と cabocha であろう。

cabocha と mecab を使用してフレンズのテンプレートを生成する

例えば次のような文章を翻訳するとしよう。

```
「私は歌うのが好きだ」
```

テンプレートは以下のようなものを使うとする

```
「すごーい、君は○○フレンズなんだね」
```

つまり翻訳後は

```
「すごーい、君は歌うのが好きなフレンズなんだね」
```

となるべきである。

ぱっと見た感じ主語をぶった切って「○○」のところに入れるとそれっぽくなる感じがする。とりあえず cabocha にかけてみようではないか。

```
$ echo "私は歌うのが好きだ" | cabocha
    私は---D
  歌うのが-D
      好きだ
```

一番一のチャンクを無視すれば主語をなくせそうである。

4.3 実際に実装する

Go 言語で書くとこのようになる。ちなみに自然言語処理を Go で書くのは珍しいかもしれないが私は Gopher なので Go 言語を使う。皆さんは好きな言語を使って実装すれば

良いと思う。

```
import (
    cabocha "github.com/ledyba/go-cabocha"
)

func TrimSubject(s string) (string, error) {
    var ret string
    c := cabocha.MakeCabocha()
    sentence, err := c.Parse(s)
    if err != nil {
        return "", err
    }
    for i, chunk := range sentence.Chunks {
        // 最初のチャンクをスキップする。また主語は名詞であることが多いので名詞で始まると
きも条件に加える
        if i == 0 && chunk.Tokens[0].Features[0] == "名詞" {
            continue
        }

        }
        for _, token := range chunk.Tokens {
            ret += token.Body
        }
    }

    return ret, nil
}
```

「私は歌うのが好きだ」において「歌うのが好きだ」が抽出できる。しかし、このままでは「君は歌うのが好きだフレンズなんだね」となり文章的に違和感のあるものになってしまう。mecab で文書構造を見てみよう

```
$ echo 私は歌うのが好きだ | mecab
私 名詞,代名詞,一般,*,*,*,私,ワタシ,ワタシ
は 助詞,係助詞,*,*,*,*,は,ハ,ワ
歌う 動詞,自立,*,*,五段・ワ行促音便,基本形,歌う,ウタウ,ウタウ
の 名詞,非自立,一般,*,*,*,の,ノ,ノ
が 助詞,格助詞,一般,*,*,*,が,ガ,ガ
好き 名詞,形容動詞語幹,*,*,*,*,好き,スキ,スキ
だ 助動詞,*,*,*,特殊・ダ,基本形,だ,ダ,ダ
```

まず邪魔なセルリアン的な文字である「だ」は助動詞であるようだ。一番最後の助動詞をぶった切ろう。すると「歌うのが好き」という言葉が抽出できるようになる。そして「すごーい、君は歌うのが好きフレンズなんだね」という言葉が生成出来るようになるが。日本語が苦手なフレンズのような言葉になってしまうので一番最後に 「な」 を入れてあげる。すると晴れて「すごーい、君は歌うのが好きなフレンズなんだね」という文書を生成できる。

ちなみに今回は「好き」が形容詞語幹なので「な」を挿入したが一般のときは「の」を入れてやるとそれっぽくなる。コード化すると以下のようになる。

```
//第一引数活用させたもの
//第二引数未活用のもの
func ExtractCharacteristicWords(s string) (string, string, error) {
    nodes, err := mecab.Parse(s)
    if err != nil {
        return "", "", err
    }
    var ret = []string{}
    if err != nil {
        return "", "", err
    }
    for i := len(nodes) - 1; i >= 0; i-- {
        if nodes[i].Pos != "名詞" && nodes[i].Pos != "形容詞" {
            nodes = nodes[:i]
        } else {
            for _, node := range nodes {
                ret = append(ret, node.Surface)
            }

            if nodes[i].Pos1 == "形容動詞語幹" {
                ret = append(ret, "な")
            } else if nodes[i].Pos1 == "一般" {
                ret = append(ret, "の")
            } else if nodes[i].Pos == "形容詞" {
                ret[len(ret)-1] = nodes[i].Base
            }
            break
        }
    }

    if len(ret) == 0 {
        return "", "", errors.New("can not convert")
    }

    return strings.Join(ret, ""), strings.Join(ret[:len(ret)-1], ""), nil
}
```

以下は簡単な文書で実行した例である

```
$ go run main.go -i 私は歌うのが好きだ
すごーい。君は歌うのが好きなフレンズなんだね

$ go run main.go -i 君は泳げるんだね
すごーい。君は泳げるんフレンズなんだね。

$ go run main.go -i 涼宮ハルヒの憂鬱
すごーい。君は憂鬱のフレンズなんだね。

$ go run main.go -i 僕は友達が少ない
すごーい。君は友達が少ないフレンズなんだね。
```

なんかそれっぽい文書が生成された。しかし、まだ不十分である。入力された文書がポジティブな発言の場合は問題にならないが、ネガティブな言葉を入力されると煽られている感じになる。サーバルちゃんは煽ってこないのだ。

4.4 煽られないようにする

端的にいうとポジティブな発言のときに「すごーい」と言わせてそれ意外のときは「だいじょーぶ、フレンズには得意不得意があるから。」と言ってもらえば良い。

手法について

東北大学の乾・岡崎研究室が公開している日本語評価極性辞書 と言うものを利用してネガポジ判定を行う。この辞書は用語の感情極性をネガティブ、ポジティブの 2 値で、名詞の感情極性をネガティブ、ポジティブ、ニュートラルで表している。この辞書を利用してネガポジ判定を行う。

実装する

形態素解析解析によって分割された単語から名詞の基本形と用言を取得する。

```
$ echo "私は歌うのが好き" | mecab
私 名詞,代名詞,一般,*,*,*,私,ワタシ,ワタシ
は 助詞,係助詞,*,*,*,*,は,ハ,ワ
歌う 動詞,自立,*,*,五段・ワ行促音便,基本形,歌う,ウタウ,ウタウ
の 名詞,非自立,一般,*,*,*,の,ノ,ノ
が 助詞,格助詞,一般,*,*,*,が,ガ,ガ
好き 名詞,形容動詞語幹,*,*,*,*,好き,スキ,スキ
```

すると「私」、「歌う」、「の」、「好き」が取り出せる。辞書には「好き」がポジティブなのでポジティブ側のパラメータを $+1$ する。これによって文書がポジティブであることを判定できる。

実装

```
type NPElem struct {
    p int
    s string
}
var index = map[string][]NPElem{}
```

```
func readNPIndex(path string) error {
    var err error
    f, err := os.Open(path)
    if err != nil {
        return err
    }
    reader := bufio.NewReader(f)
    for line := ""; err == nil; line, err = reader.ReadString('\n') {
        line = strings.Replace(line, "\n", "", -1)
        i := strings.Split(line, "\t")
        if len(i) < 2 {
            continue
        }
        np := parseNP(i[0])
        words := strings.Split(i[1], " ")

        e := NPElem{np, i[1]}
        key := words[0]
        index[key] = append(index[key], e)
    }
    return nil
}

func CalcNP(s string) (int, error) {
    var result int
    nodes, err := mecab.Parse(s)
    if err != nil {
        return 0, err
    }
    for _, node := range nodes {
        elems, ok := index[node.Base]
        if !ok {
            continue
        }

        result += elems[0].p
    }
    return result, nil
}
```

実行結果

```
$ go run main.go -i 私は歌が好き
すごーい。君は歌が好きなフレンズなんだね

go run main.go -i 僕は運動が苦手だ
だいじょーぶ。フレンズには得意不得意があるからー。

$ go run main.go -i スホーイは空を飛ぶ
すごーい。君は空のフレンズなんだね。
```

この手法の問題

辞書に載っていない単語がやってきた時対応しづらい。
また、ネガティブな単語とポジティブな単語が同じバランスで入っている時きれいに判定ができない。

4.5 さらなる精度を求めるための改良

ベイズ分類器を利用してネガポジを振り分ける。ベイズ分類器について詳細に書くとページ数の関係で辛いことになりそうなのでこのあたりの記事を見ると良さそうです。

ナイーブベイズ分類器を頑張って丁寧に解説してみる - Qiita

なにに使われている技術かと言うと迷惑メールのフィルタとかに使われている技術です。迷惑メールの場合は迷惑メールである、迷惑メールでないを判定しますがここではポジティブな文書である、ネガティブな文書である、どちらでもないを判定します。Go 言語で実現するためには `github.com/jbrukh/bayesian` というパッケージを利用するとよいです。

実装

まず Class と分類器を定義します。

```
// class の定義
const (
    P bayesian.Class = "Posi"
    N bayesian.Class = "Nega"
    E bayesian.Class = "Neither"
)

// 分類器を定義した構造体
type classifier struct {
    PN *bayesian.Classifier
    NE *bayesian.Classifier
    EP *bayesian.Classifier
}

// 分類器を初期化する
func newClassifier() *classifier {
    return &classifier{
        bayesian.NewClassifier(P, N),
        bayesian.NewClassifier(N, E),
        bayesian.NewClassifier(E, P),
    }
}
```

判定関数を実装するベイズ分類器は分かち書きを作って食わせて上げる必要があるので mecab を使って分かち書きをつくる。PN, NE, EP 判定機に分かち書きにした文書を入力すると score が取得できるので P, N, E をそれぞれ集計する。一番 0 に近いものを結果として出力する。

```
func (c *classifier) DeliberationNP(s string) bayesian.Class {
    var doc = []string{}
    var (
        PScore = float64(1.0)
        NScore = float64(1.0)
        EScore = float64(1.0)
    )
    nodes, err := mecab.Parse(s)
    if err != nil {
        return ""
    }

    for _, node := range nodes {
        doc = append(doc, node.Base)
    }

    pnScores, _, pnb := c.PN.LogScores(doc)
    if pnb {
        PScore = PScore * (-1 * pnScores[0])
        NScore = NScore * (-1 * pnScores[1])
    }
    niScores, _, nib := c.NE.LogScores(doc)
    if nib {
        NScore = NScore * (-1 * niScores[0])
        EScore = EScore * (-1 * niScores[1])
    }
    ipScores, _, ipb := c.EP.LogScores(doc)
    if ipb {
        EScore = EScore * (-1 * ipScores[0])
        PScore = PScore * (-1 * ipScores[1])
    }

    if PScore < NScore {
        if PScore < EScore {
            return P
        }
    } else {
        if NScore < EScore {
            return N
        }
    }
    return E
}
```

学習関数は以下のようにする

```
func learn(s string, which bayesian.Class, c *classifier) *classifier {
    var doc = []string{}
    nodes, err := mecab.Parse(s)
    if err != nil {
        return c
    }

    for _, node := range nodes {
        doc = append(doc, node.Base)
    }
    c.Learn(doc, which)
    return c
}

func (c *classifier) Learn(document []string, which bayesian.Class) {
    if which == P {
        c.PN.Learn(document, which)
        c.EP.Learn(document, which)
    } else if which == N {
        c.NE.Learn(document, which)
        c.PN.Learn(document, which)
    } else if which == E {
        c.EP.Learn(document, which)
        c.NE.Learn(document, which)
    }
}
```

分類器を 3 つに分割することで全ての学習データが揃わなくても分類することができる。

実行する

```
$ go run nb/main.go -i 今日は凍え死にそうだ
Nega

$ go run nb/main.go -i あなたは何もできないフレンズなのね
Nega

$ go run nb/main.go -i 残業辛い
Nega

$ go run nb/main.go -i かばんちゃんはすっごいんだよ！
Posi

$ go run nb/main.go -i 当たり前じゃない！ かばんちゃんを助けて、また色んなところ行くんだから！
Posi

$ go run nb/main.go -i スホーイは空をとぶのが好きなフレンズなんだね。
Posi

$ go run nb/main.go -i 食べないよ！
```

```
Nega

$ go run nb/main.go -i いきるのがつらい
Nega
```

4.6 さいごに

今回はテンプレートに単語を埋め込んでフレンズ語を生成しました。文書のタグ付け技術とか深層学習をつかってフレンズ語を生成できると面白いなと思っています。もし第二弾がかけるようであれば作ってみたいなと思っています。

4.7 参考文献

- ナイーブベイズ分類器を頑張って丁寧に解説してみる - Qiita
- ネガポジ判定を行う Gem 作ってみた - Qiita
- 日本語評価極性辞書 - 乾・岡崎研究室 - Tohoku University

4.8 使用ライブラリ

- github.com/ledyba/go-cabocha
- github.com/yukihir0/mecab-go
- github.com/jbrukh/bayesian

4.9 つくったもの

- https://github.com/ieee0824/friends-translator

第 5 章

MySQL でいろいろなグラフを書こう！

5.1 はじめに

MySQL の出力結果をグラフにして表示するには手間がかかります。GUI ツールを使うこともあれば、結果を CSV に出力してエクセルなどで操作することもあるでしょう。

しかし、我々には標準出力という強い味方がいます。

SSH で接続した時、エクセルを開くのが面倒なとき、様々な場合であっても標準出力であれば気軽に結果を確認することができます。

今回は、MySQL の標準出力のみでリッチなグラフを描画する方法について考えていきたいと思います。MySQL でグラフをかければ、面倒なエクセル操作とおさらばできます！ やったー！

環境

あまり難しいことを考えず、Homebrew でインストールした MySQL を利用します。version については以下の通りです。

```
$ brew info mysql
mysql: stable 5.7.17 (bottled)
```

5.2 1. 連番を生成する

レコードの内容に対して、連番を生成するところから始めます。グラフとか関係なく、日付順にデータを並べたりする際などに便利です。

MySQL に制御構文はありませんが、SELECT 句によるレコード一件を 1 回のループとしてみれば、変数をインクリメントさせることができます。

説明用に以下のようなテーブルを用意しました。@<tt>{ mysql> SELECT * FROM random_nums; +-------+ | num | +-------+ | 122 | | 13123 | | 189 | | 1213 | | 3 | | 79879 | | 3169 | | 791 | | 9810 | | 114 | +-------+ 10 rows in set (0.00 sec) } 連番を生成しつつ、ランダムな整数を小さい順に並べたいと思います。

```
mysql> SET @i = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT
    ->   @i := @i + 1 AS increment,
    ->   num
    -> FROM
    ->   random_nums
    -> ORDER BY
    ->   num
    -> ;
+-----------+-------+
| increment | num   |
+-----------+-------+
|         1 |     3 |
|         2 |   114 |
|         3 |   122 |
|         4 |   189 |
|         5 |   791 |
|         6 |  1213 |
|         7 |  3169 |
|         8 |  9810 |
|         9 | 13123 |
|        10 | 79879 |
+-----------+-------+
10 rows in set (0.00 sec)
```

連番を生成することができました。ここで注意しなくてはならないのは、インクリメントの回数の上限はテーブルのレコード数に依存するということです。より大きな数でインクリメントを生成した場合は、それ相応の大きさをのテーブルを用意する必要があります。

5.3 2. 座標平面

グラフを描画するにはまず画面がなくてはいけません。先ほど作った連番を利用して、座標平面を作ってみます。21 のインクリメントが必要なので、適当に 21 のレコードを持つテーブルを生成します。

```
mysql> CREATE TABLE increments (id INT, num INT);
Query OK, 0 rows affected (0.03 sec)

mysql> SET @i = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO increments (id, num) SELECT @i := @i + 1 AS id, FLOOR(RAND() * 100) AS num FROM ho
Query OK, 21 rows affected (0.00 sec)
Records: 21  Duplicates: 0  Warnings: 0
```

※ hoge は 21 行ある適当なテーブルです。ではこの increments テーブルを利用して座標平面を描いていきます。

```
mysql> SELECT
    ->   (i.id - 11) / 10 AS x_axis,
    ->   (j.id - 11) / 10 AS y_axis
    -> FROM
    ->   increments AS i,
    ->   increments AS j
    -> ORDER BY
    ->   x_axis,
    ->   y_axis
    -> ;

+---------+---------+
| x_axis  | y_axis  |
+---------+---------+
| -1.0000 | -1.0000 |
| -1.0000 | -0.9000 |
| -1.0000 | -0.8000 |
| -1.0000 | -0.7000 |
.....................
|  1.0000 |  0.9000 |
|  1.0000 |  1.0000 |
+---------+---------+
441 rows in set (0.00 sec)
```

こんな感じで、(-1.0, -1.0) から (1, 1) を 0.1 刻みで分割した (x, y) の組み合わせを生成することができました。あとは GROUP_CONCAT 関数を使って結合してあげれば完成です。実際にやってみます。

```
mysql> SELECT
    ->   GROUP_CONCAT(
    ->     value
    ->     ORDER BY x_axis, y_axis DESC
    ->     SEPARATOR ''
    ->   ) AS 'Coordinate plane'
    -> FROM (
    ->   SELECT
    ->     grid.x_axis,
    ->     grid.y_axis,
    ->     CASE
    ->       WHEN grid.x_axis = 0    AND y_axis = 0
    ->         THEN '-+-'
    ->       WHEN grid.x_axis = -0.1 AND y_axis = -0.1
    ->         THEN ' 0 '
    ->       WHEN grid.x_axis = 0.1  AND y_axis = -1
    ->         THEN ' y '
    ->       WHEN grid.x_axis = 1    AND y_axis = -0.1
    ->         THEN ' x '
    ->       WHEN grid.x_axis = 0
    ->         THEN ' | '
    ->       WHEN grid.y_axis = 0
    ->         THEN '---'
    ->       ELSE '   '
    ->     END AS value
    ->   FROM (
    ->     SELECT
    ->       (i.id - 11) / 10 AS x_axis,
    ->       (j.id - 11) / 10 AS y_axis
    ->     FROM
    ->       increments AS i,
    ->       increments AS j
    ->     ORDER BY
    ->       x_axis,
    ->       y_axis
    ->   ) AS grid
    ->   ORDER BY
    ->     grid.x_axis,
    ->     grid.y_axis
    -> ) AS vlues,
    -> (
    ->   SELECT @cel_size := 10
    -> ) AS cel_size
    -> GROUP BY
    ->   y_axis
    -> ;
+------------------------------------------------------------------+
| Coordinate plane                                                 |
+------------------------------------------------------------------+
|                               |  y                               |
|                               |                                  |
|                               |                                  |
|                               |                                  |
|                               |                                  |
|                               |                                  |
|                               |                                  |
|                               |                                  |
|                               |                                  |
|                            0  |                              x   |
```

```
| ------------------------------+------------------------------ |
|                               |                               |
|                               |                               |
|                               |                               |
|                               |                               |
|                               |                               |
|                               |                               |
|                               |                               |
|                               |                               |
|                               |                               |
|                               |                               |
+-------------------------------------------------------------------+
21 rows in set (0.00 sec)
```

5.4 3. 棒グラフを作ろう

では、先ほど作成した座標平面を利用して、棒グラフを作ってみます。
基本的な作成方法としては、先ほどの座標平面の各 (x, y) において、棒グラフが描画されるかどうかを判断すれば OK です。

例として、都道府県の人口トップ 4 をピックアップしました (ex. wikipedia)

```
mysql> SELECT
    ->   *
    -> FROM
    ->   prefecture_population
    -> LIMIT
    ->   4
    -> ;
+----+--------------+------------+
| id | name         | population |
+----+--------------+------------+
|  1 | 東京都       |   13515271 |
|  2 | 神奈川県     |    9126214 |
|  3 | 大阪府       |    8839469 |
|  4 | 愛知県       |    7483128 |
+----+--------------+------------+
4 rows in set (0.00 sec)
```

```
mysql> SELECT
    ->   GROUP_CONCAT(
    ->     value
    ->     ORDER BY x_axis
    ->     SEPARATOR ''
    ->   ) AS '主要 4 都府県の人口'
    -> FROM (
    ->   SELECT
    ->     x_axis,
```

```
    ->     y_axis * -1 AS yy_axis,
    ->     MAX(x_range),
    ->     name,
    ->     rate,
    ->     CASE
    ->       WHEN x_range = x_axis
    ->         THEN '  '
    ->       WHEN rate > y_axis
    ->         THEN '□□'
    ->       ELSE '  '
    ->     END AS value
    ->   FROM (
    ->     SELECT
    ->       x_axis,
    ->       y_axis,
    ->       (id - 2) / 2 AS x_range,
    ->       name,
    ->       population,
    ->       @max := (SELECT MAX(population) FROM prefecture_population) AS max,
    ->       ROUND(population / @max, 1) AS rate
    ->     FROM (
    ->       SELECT
    ->         (i.id - 11) / 10 AS x_axis,
    ->         (j.id - 11) / 10 AS y_axis
    ->       FROM
    ->         increments AS i,
    ->         increments AS j
    ->       ORDER BY
    ->         x_axis,
    ->         y_axis
    ->     ) AS grid, (
    ->       SELECT
    ->         id,
    ->         name,
    ->         population
    ->       FROM
    ->         prefecture_population
    ->       ORDER BY
    ->         population DESC
    ->       LIMIT 4
    ->     ) AS populations
    ->   ) AS tmp_values
    ->   WHERE
    ->     x_axis <= x_range
    ->   GROUP BY
    ->     x_axis,
    ->     yy_axis
    -> ) AS graph_value,
    -> (
    ->   SELECT @cel_size := 10
    -> ) AS cel_size
    -> GROUP BY yy_axis
    -> ;
+-------------------------------------------------------------------------------------------------------------
| 主要４都府県の人口
+-------------------------------------------------------------------------------------------------------------
|
| □□□□□□□□□□□
```

```
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠
| ⊠ ⊠ ⊠ ⊠ ⊠ ⊠ ⊠ ⊠ ⊠ ⊠  ⊠ ⊠ ⊠ ⊠ ⊠ ⊠ ⊠ ⊠  ⊠ ⊠ ⊠ ⊠ ⊠ ⊠ ⊠
⊠                                                                      |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
| ⊠⊠⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠⊠  ⊠⊠⊠⊠⊠⊠⊠
⊠                                               |
+------------------------------------------------------------------------------------------------------
21 rows in set (0.01 sec)
```

第 6 章

きゃりあこんさるてぃんぐちほー

6.1　これはなに？

エンジニアの職務経歴書の書き方だよ

6.2　なんでこんなもんを書いたのか？

こんにちは。
唐突ですが、筆者はひょんなことからエンジニアの職務経歴書を見るお仕事に関わってしまいました。
この仕事を 1 年ほど続けてわかったこととして、いくつかありますが、

- エンジニアの実力と職務経歴書を各実力は相関しない
- 職務経歴書の良し悪しで行ける会社も給与も変わる

中でも上はホントそうだなと思いました。
「この人実力はあるのかもしれないけど、多分採用担当者には伝わらないだろうな」とか思う日々が続き悶々としていたので、今日は職務経歴書に関するあるあるや、うまい書き方なんかをご紹介したいと思います。

6.3　あらすじ

ブラック企業で 3 年間修行した たかし は、この度転職を決意。
キャリアコンサルタントのに相談しに行くのであった

6.4 たかし、転職を決意する！

たかし、以下た「こんにちはたかしです！ 今日はよろしくお願いします！」
キャリアコンサルタント、以下コ「こんにちはたかしさん。本日はご足労頂きありがとうございます。さっそくですが職務経歴書を見せて頂けますか？」
た「はい！（ドヤ顔で）」

▼

```
2014 年 4 月 1 日 - 2017 年 4 月 1 日
株式会社 xx にて web サービスの開発担当
【人数】5 名
【担当】設計、開発、テスト、保守
```

6.5 何をしていたのか、具体的に

コ「最低ですね」
た「そんな……」
コ「ここからは web サービスを開発していたことしかわからないです。採用担当者としてはどんなサービスに関わっていたのかを知りたいはずです」
た「なぜでしょうか？」
コ「たとえば、私が EC サイトを開発している会社の採用担当者だとしましょう。ほしいのは SNS 開発経験者？ それとも EC サイト開...」
た「EC サイト開発経験者」
コ「そうですね。前職で同じような業界にいた人は、採用担当者としても気になります」
—— 書き直すたかし

▼

```
- 株式会社 xx にて web サービスの開発担当
+ 株式会社 xx にて転職サイトの開発担当
```

コ「たかしさんは転職サイト開発をしていたのですね。web サービスというとフロントエンドやサーバサイド、インフラなどいろいろ役割があると思いますが、どこを担当していましたか？」
た「サーバサイドでした」
コ「採用ポジションもどこが空いているかで細かく別れるので、そういった情報も書いておきましょうね」

―― 書き直すたかし

▼

```
- 【実績】サブリーダーとして納期を守りました
+ 【実績】サーバサイドエンジニアのサブリーダーとして納期を守りました
```

6.6 スキルを見せろ！

コ「サーバーサイドエンジニアだったことが分かるようになりましたね。では具体的にどんな開発をしましたか？」
た「そうですね。スカウトメールの配信システムや、企業と求職者がメッセージの遣り取りをする機能、あとは運営用のツールなんかを作りましたね」
コ「結構色々作ってますね。ではそれらのシステムやツールはどんな言語やフレームワーク、ロジック、外部サービスなどを用いて作られていますか？」
た「スカウトメールは php ですね。協調フィルタリングというロジックを用いて求職者の好みそうな求人をマッチする仕組みです。メッセージ機能と運営用のツールは Rails で作りました」
コ「なるほど。今の説明で、貴方が どんな技術を使えるエンジニアなのか 分かるようになりました。一気に貴方の魅力があがってきましたね。」
た「おおお！」
―― 書き直すたかし
た「これでどうでしょうか？」

▼

```
+ 実際に開発したものとして以下などがあります
+ ・スカウトメール配信システム（PHP, 協調フィルタリングを用いて求職者の + 嗜好とマッチング）
+ ・メッセージ機能（Rails）
+ ・運営用ツール（Rails）
```

6.7 背景にある問題と、解決までのルートをみせよう！

コ「大分良くなってきましたね」
た「よかった」
コ「ちなみに、スカウトメールはどんな理由で開発が始まったのですか？」
た「これはですね、当時サービスが伸びて登録している求人が増えて、自分にあった求人を見つけるのが難しくなってきたんです。これを解決するため、運営が求職者の嗜好に

マッチする求人を見つけてあげればよいのでは？ という事になったのですが、流石に大変すぎました。そこで、求職者の嗜好にマッチした求人を自動で見つけられたらどうか？ ということで作られました。これ、僕が考えたんですよ（ドヤ顔で）」
コ「なるほど！ 素晴らしいですね。たかしさんは自分で問題を見つけ、エンジニアリングで解決する方法を提案し、解決したのですね。採用担当者は求職者が どんな問題があった時に、どうやって解決できるか？ を見ています。自発的に行動し、周りの問題点を解決できる人なのだなと伝えましょう！」
た「おおお！」
—— 書き直すたかし
た「これでどうでしょうか？」
▼

```
+ 【実績例 1】
スカウトメール配信システム（PHP，協調フィルタリングを用いて求職者の嗜好とマッチング）
+ 【開発経緯】
+ 当時サービスが伸びて登録している求人が増えた。
+ これにより、求職者は自分にあった求人を見つけるのが難しくなった。
+ これを解決するため、運営が求職者の嗜好にマッチする求人を見つける企画がはじまったのだが、求職者の人数が多くコストがかかりました。
+ そこで、求職者の嗜好にマッチした求人を自動で見つければコストを削減できるのでは？
+ となり、開発しました。企画者は自分です。
+ 【実績例 2】
・メッセージ機能（Rails）
+ 【実績例 3】
・運営用ツール（Rails）
```

6.8 出した成果は正義！

コ「かなり素晴らしいですが、まだまだよくできます」
た「どうすれば！！！？」
コ「そうですね。ではたかしさんがつくったものによってどんな効果がありましたか？」
た「スカウトメールでは手動でマッチしてた時間が全部なくなりました。その上人為的なミスもなくなったので、クレームも減りましたね。メッセージ機能は、これまでメールで行われていた求職者と企業のやり取りの中身が分かるようになったので、悪さをしてる企業や、面接をすっぽかす求職者がわかったりしてサービスのクオリティを上げるための情報源になりました。運営用のツールも手動で行われていたオペレーションのミスを減らしたり、便利にしたりでよろこばれました」
コ「良いですね。具体的に、スカウトメール配信システムを手動でやってた頃は何人が何時間くらいかけていたとかわかりますか？」
た「うーんと、多分 5 人が毎日 3 時間位かけていたと思います。それが全部なくなりまし

た」
コ「スゴイですね。1 日 15 時間分。20 日で 300 時間分の仕事を自動化できたのですね。定性的な成果ももちろん大事ですが、定量的な成果は採用担当者にも「ウチに入ってくれたらこのくらい成果を出してくれそうだな」と、大きく響きく重要な事項です」
　た「おおお！」
　—— 書き直すたかし
　た「これでどうでしょうか？」
▼

```
+   【成果】
+   1 日 3 人 5 時間分＝ 300 時間/月の業務コスト改善
+   サービス改善のための情報源となった
+   オペレーションミスの低減。使いやすくなった。
```

6.9　リーダーってなにしたの？

　コ「かなり素晴らしいですが、せっかくなのでもっと良くしましょう」
た「はい！ 喜んで！」
コ「では質問です。たかしさんは『サブリーダーとして納期を守った』そうですが、具体的にサブリーダーという役職として何をしたことで納期をなぜまもれたのでしょうか？」
た「うーん。根性ですかね？」
コ「採用担当者から見たら「根性でどうにかしたんだな」しか伝わらないし、リーダーとしての能力が無さそうに見えるかもしれませんね」
た「むーん。」
コ「では逆に聞きましょう。納期が遅れそうになることありましたか？」
た「ありました。欠員が一人出て、その時は大変でした」
コ「その時は根性で？」
た「根性ではどうにもなりそうになかったので、仕様の内最初のリリースで必要ないものについて後回しにできないか相談しました」
コ「良いですね。納期を守った話やリーダーだった話はよく出ますが、(リーダーとして or 納期を守るために) 実際に何をしたのか？ が大事です。そういう所をアピールしていきましょう」
　—— 書き直すたかし
　た「これでどうでしょうか？」
▼

+ メンバーの離職で欠員が出て納期が厳しくなった時には、仕様の内最初のリリースで必要ないものについて後回しにできないかなど相談して納期を守ることができました。

6.10 まだまだ伸ばせる！ あなたの職務経歴書！

コ「もう十分ですが、更に高みを目指しましょう」
た「yes!」
コ「では 1 つずつ聞きますね。このプロジェクトって結果どうなりましたか？」
た「まだ続いてますが、現時点では入った頃よりもずっと大きなプロジェクトになり、入った頃の 10 倍以上の売り上げを上げています」
コ「成果と同様結果も大事です。どれほどのプロジェクトでどのくらい貢献して来たのか？ 採用担当者が想像しやすくなります」

コ「続いて、あなたが担当した開発等で工夫したことや気をつけたことってありますか？？ 根性でどうにかしたやつ以外で」
た「そうだなー...。全般的にコードきれいに書くことには気をつけてました。」
コ「例えばどんな風に？」
た「リーダブルコードやコードコンプリート等のコードをキレイにするための有名な本はだいたい読みましたし、コードレビューの時にチームに展開もしました。工夫したって程じゃないですが、rubocop や lint を導入して静的解析もしてます」
コ「良いですね！ 本を読んでること、チームに知識や行動を展開できること、ツールなどを導入して改善できること。どれも素晴らしいです。採用担当者は学習意欲の高さ やチームへ好影響を出す巻き込み力があるか 現場改善のための行動 等をよく見ています！」
コ「では失敗したことなんかも教えて頂けますか？」
た「詳細はかけないですけど、かなり大きなバグを出して大きな損害を出したことがあります」
コ「それは大きな失敗ですね。その経験っていま活きてますか？」
た「当時テストが全然整備されてなかったんですけど、それ以降はテストしっかり書くようになりましたね。こういう所はミスしたらサービスが大ダメージ受けるからテストをシッカリしようとかいつも考えてます」
コ「失敗したことがちゃんと活かせる人かどうか？ はとても大事なことです。転んでもただでは起きない強い人はどこでも好まれます」
コ「他に技術的にアピールしたいこととかありますか？ こういうツール使ったとか、こういう技術的なアプローチをしたとか」

た「フロントエンドに AngularJs を導入したこととかかな。当時はまだ jQuery 全盛だった頃で、まだまだ新しかった AngularJs を入れて修正しやすい設計似できたことは良かったと思います」
コ「エンジニアにとって 技術的なチャレンジをしていること、新しいものに興味が有ること は大事なことですし、大きなアピールになりますよ！ 採用担当者だって 自社の未来を作る人がほしい のです」

```
2014 年 4 月 1 日 - 2017 年 4 月 1 日
株式会社 xx にて転職サイトの開発担当

【人数】5 名
【担当】設計、開発、テスト、保守

【実績】サーバサイドエンジニアでした。
メンバーの離職で欠員が出て納期が厳しくなった時には、仕様の内最初のリリースで必要ないものについて後回しにできないかなど相談して納期を守ることができました。
実際に開発したものとして以下などがあります

【実績例 1】
スカウトメール配信システム（PHP, 協調フィルタリングを用いて求職者の嗜好とマッチング）
  【開発経緯】
    当時サービスが伸びて登録している求人が増えた。
    これにより、求職者は自分にあった求人を見つけるのが難しくなった。
    これを解決するため、運営が求職者の嗜好にマッチする求人を見つける企画がはじまったのだが、求職者の人数が多くコストがかかりました。
    そこで、求職者の嗜好にマッチした求人を自動で見つければコストを削減できるのでは？
    となり、開発しました。企画者は自分です。
    【成果】
    1 日 3 人 5 時間分＝ 300 時間/月の業務コスト改善
【実績例 2】
・メッセージ機能（Rails）
  【成果】
  サービス改善のための情報源となった
【実績例 3】
・運営用ツール（Rails）
  【成果】
  オペレーションミスの低減。使いやすくなった。
【その他】
コードを綺麗に書くことには気をつけています。
リーダブルコードやコードコンプリート等のコードをキレイにするための有名な本はだいたい読みましたし、コードレビューの時にチームに展開もしました。
rubocop や lint を導入して静的解析もしてます。
出たばかりの頃の AngularJs を用いて修正しやすい設計の js コーディングにも気をつけました。

また、テストもしっかり書くようにしています。これは以前大きなバグを出した時から気をつけていて、特にサービスが止まるようなクリティカルな箇所かどうかを常に気をつけています。
【結果】
このサービスは自分が入ってから 3 年で売り上げを 10 倍以上に伸ばしました。
```

コ「大変良くできました！ 素晴らしい転職ができますように！」

6.11 最後に

これはフィクションですが、職務経歴書において大事なことは、採用担当者にあなたの魅力がちゃんと伝わることです。
客観的に見るのは結構しんどいので、誰かにレビューしてもらうか、もしくは以下のチェックシートなどを参考にしてよりよい職務経歴書を、よりより職場を目指して下さい！
それでは！！

6.12 おまけのチェックシート

- あなたが何をしていたか具体的に分かる
- あなたの業界が分かる
- あなたのプロジェクトの規模やフェーズが分かる
- あなたの技術力が分かる
- あなたがどんなことが出来るのか分かる
- あなたの学習意欲が分かる
- あなたが最近のトレンドについて追っているかどうか分かる
- あなたが問題をどう解決するのか具体的な例がある
- あなたが周りを巻き込んで物事を改善できるか分かる
- あなたの問題意識の高さが分かる
- あなたが失敗を反省して活かせる人が分かる
- あなたが転んでもただでは起きないマインドか分かる

- プロジェクトの結果が分かる
- プロジェクトの成果が定性的、定量的に分かる
- あなたが自分自身の魅力を人に語れる

第 7 章

TableView を好きになっても ScrollView は嫌いにならないでください

7.1 はじめに

みなさんこんにちは。iOS アプリでスクロールする画面を作るとき、皆さんはどうやって作成しているでしょうか？ TableViewController や PageViewController を使ってやられる方結構いると思います。TableViewController であれば複雑なレイアウトもセルごとに分割できるのでわかりやすくできますよね。私はこれまで ScrollView を使った実装もしてきました。ただ、ScrollView の特性を良く理解しないまま使っていたので結構詰まることもあり、結局 TabaleView を使うなどしてきました。しかし、スクロールさせたいなら ScrollView でもいいじゃないですか！ 私が ScrollView について詳しくないから使いこなせていないだけだ、そうにちがいない、もっと ScrollView を好きになろうという決意でこの記事を書いています。笑 ScrollView が苦手な人も基本をおさらいしてみましょう。

7.2 ScrollView の基本

ScrollView は二つの size を持っています。たとえながら説明すると、一つは窓枠の役割を担う frame、もう一つは景色の役割を担う contentSize です。frame は他のオブジェクトも持つプロパティで、オブジェクトの見た目の大きさです。contentSize は表示コンテンツの大きさを示すもので、ScrollView の上に乗っている View の大きさになります。frame という窓から見える景色の大きさが contentSize と覚えるとわかりやすいですね。

大きは横・縦どちらの方向にも contentSize として設定できるので横・縦どちらのスクロールも設定次第でできます。

```
--[[path =  (not exist)]]--
```

frame と contentSize

7.3 使ってみてわかるもどかしさ

以下のコードでは二つの画像に挟まれた textView を用意しています。

```
--[[path =  (not exist)]]--
```

ねこ

▼

```
@IBOutlet weak var text1: UILabel!

override func viewDidLoad() {
  super.viewDidLoad()
    text1.translatesAutoresizingMaskIntoConstraints = true
    text1.text = "ねこ"
    text1.sizeToFit()
}
```

この状態で text1.text をこのようにしてみます

▼

```
text1.text = "ねこねこねこねこねこねこねこねこねこねこね..."
```

こうするとどうなるかというと

```
--[[path =  (not exist)]]--
```

ねこ２

こうなるんですが、このままでは スクロールができません。なぜかというと ScrollView に乗っている ContentView の高さが変動していないからです。ContentView の高さを変動させ ContentSize を変更させるために、以下の処理を追加します

```
contentViewConstHeight.constant += 調整したい高さ;
```

これで ContentView の大きさは調整されるのでスクロールすることができます。ちなみに、contentSize を調整するには ScrollView がもつ contentSize プロパティからもできます。

```
ScrollView.contentSize += 調整したい高さ;
```

こんな感じで簡単な仕組みですが、複雑なデザインを調整すると AutoLayout でやる場合は使いにくかったりします。

7.4 忘れがちなエラー

私自身やってしまう失敗があります。

▼

```
Scroll View
Has ambiguous scrollable content width

Scroll View
Has ambiguous scrollable content height

Ambiguous Layout
Scrollable content size is ambiguous for "Scroll View"
```

だいたいこれらは AutoLayout で ScrollView を配置して上に view を乗せたときに発生します。要は、width・height が不明、つまり ContentView の大きさがわからないというエラーですね。手順項目としては以下のとおり

- ScrollView を配置
- ScrollView の width・height を設定
- ContentView を配置

--[[path = (not exist)]]--

scrollView_AutLayout の設定

```
--[[path =  (not exist)]]--
```

contentView_AutLayout の設定

ScronnView の width・height が決まらないと ContentView をおいてもエラーが出てしまうので気を付けましょう。

7.5 ScrollView で無限スクロール

7.6 ページ単位のスクロール

ちなみに ScrollView に画面ごとの View を配置して複数画面をスクロールできるように設定することもできますが、ページ単位のスクロールであれば UIPageViewController を使うと良いかもしれません。ページ単位の View をまとめて管理してくれます。

▼

```
https://developer.apple.com/reference/uikit/uipageviewcontroller
```

また無限スクロールできるライブラリもあるようです。使ってみてはいかがでしょう。

▼

```
EndouMari/TabPageViewController
```

7.7 おわりに

いかがでしたでしょうか、ScrollView 好きになりましたか？私は... やっぱり TableView の方が使いやすいかなと思ってしまいました。笑複雑なデザインを構成するとき、追加で表示させるパーツや高さ調整がややこしくなったりしてしまいますね。こういうときは TableView を使ってセル単位でデザインを使用した方が扱いやすいですね。そういった点から ScrollView はシンプルな画面のスクロールで使うぶんには良いかなと思います。常に View をシンプルなものにできれば良いですが...。ただ、今回は ScrollViewww をおさらいしたのでもう基本は大丈夫ですね。みなさん ScrollView を嫌わずに使ってみましょう！

第 8 章

Charles を使ってプロキシサーバをローカルに構築する

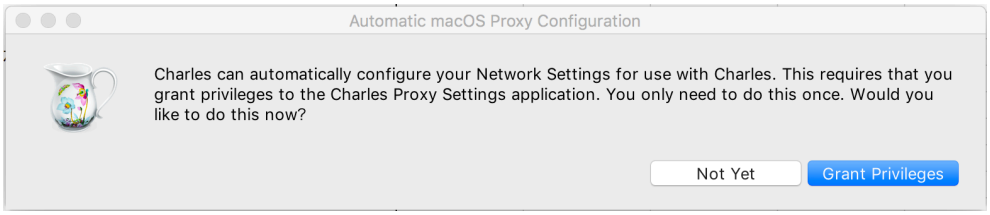
8.1　iOS 開発での通信デバッグ

通信処理を行うアプリの開発をしていると、当然ながら通信処理に関するデバッグが必要になってくる。Android 開発では steto を使えば簡単に通信ログが確認できるが、iOS 開発ではこれがなかなか面倒くさい。最近では Charles というツールが便利なようなので触ってみた。

8.2　Charles のセットアップ

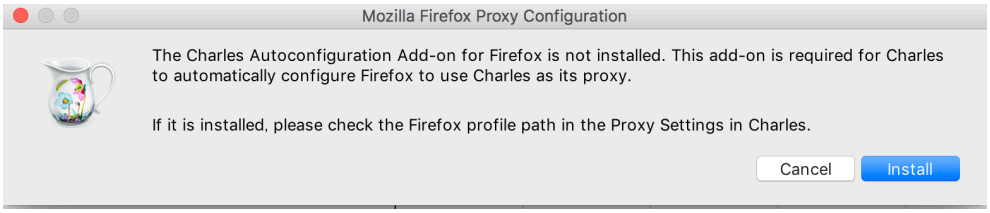
Charles は下記の公式 URL からダウンロードできる。`https://www.charlesproxy.com/download/`

Charles の初回起動時、Charles を起動している間はプロキシの設定を Charles のものに切り替えるかどうか聞かれるので、「Grant Privileges」を選択して設定を行う。



▲図 8.1　起動時設定

そのあと、Firefox のアドオンもインストールするか聞かれるが、これは利用したい場合のみ「Install」を選択すれば良い。



▲図 8.2　Firefox アドオンの選択

Charles をプロキシサーバとして利用するため、Mac 本体と iPhone は同じ Wifi に接続しておくこと。

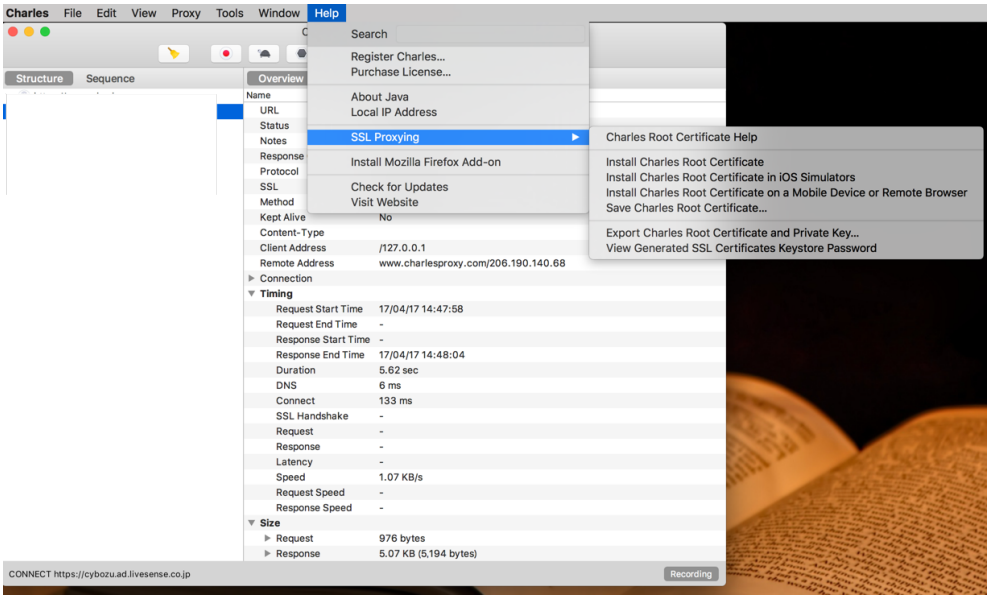
8.3　Mac の設定

Charles のメニューから「Proxy -> SSL Proxying Settings」を選択し、開かれた画面で Add ボタンを押下して「Host」に * を入力して OK ボタンで完了。

```
--[[path =  (not exist)]]--
```

ワイルドカード設定

同じく Charles のメニューから「Help -> SSL Proxying -> Install Charles Root Certificate」を選択して証明書を Mac にインストールする。



▲図 8.3　証明書インストール

キーチェーンツールを起動すると「Charles Proxy Custom Root Certificate」がインストールされていることがわかる。この時、「このルート証明書は信頼されていません」と表示される場合があるが、「Charles Proxy Custom Root Certificate」をダブルクリック -> 信頼タブを開く -> 常に信頼するを選択、で証明書を信頼できるように設定することができる。

8.4　iPhone の設定

Mac の IP アドレスは Charles のメニュー「Help -> Local IP Address」で確認できる。ポートはデフォルト 8888 で設定されているので、この IP アドレスとポートを iPhone 本体のの Wifi 設定から HTTP プロキシ設定に入力する。

▲図 8.4　Wifi 設定

この状態で iPhone で通信を発生させると Charles 側で確認ダイアログが表示されるので「Allow」を選択する。

最後に下記の URL に iPhone でアクセスし、本体にプロファイルをインストールする。

`http://www.charlesproxy.com/getssl`

8.5　感想と後悔

Chales には無料版があり、設定も簡単なので、気軽に試すことができるが、しっかりと使いたい場合には有料版を使うほうが良い。iOS では ATS の設定を変更する必要があったりと、この辺は Apple のセキュリティガイドラインに沿うしかない。それでも、BreakPoint の設定ができたり、キャプチャが撮れたり、デバッガーとしては一通りの機能が備わっていて良いんじゃないかと思った。同じような HTTP プロキシツールとして、Apple が示している他のツールもあるので、そのうち試してみたい。

ちなみに、今回の執筆にあたって、当初考えていたテーマは SpriteKit についてだった。サンプルのアプリは先に作り終わっていて、いざ原稿を書こうと思った段階で……かなり簡単に説明しようと思っても 20p くらい必要なことに気がついた。なので別の機会に、ぜひ SpriteKit についての記事を書きたいな〜と思った次第。

いろんなちほーがまざったほん　開発がとくいなフレンズになる方法 - テーマは十人十色! -

2017 年 4 月 29 日　発行
著　者　　たかしファンクラブ
イラスト　@onunu
編　集　　@roana0229、@onunu
印刷所　　日興企画