# 上級試験合格のための学習方法の解説と中級者になるためのフレームワーク紹介

by 古庄道明(がる/gallu)

# 自己紹介

- 古庄道明と申します
- 「がる(gallu)」というハンドルでふらついております
- 本職は技術者です。現役プログラマーやってます
  - バックエンド系なので、インフラとかDBとか運用とかも一通り
  - 最近はPM業も多いですねぇ
  - 「ヒアリングして技術提案」なんてこともさせていただいてます
- 教育も色々と携わっております(専門学校講師とか)
- 「PHP8技術者認定上級試験」「PHP8技術者認定初級試験」の作成を担当させていただいてます
- コラムとかblogとかeラーニングの教材とかで技術系の文章を書くこともちらほら

# 今日のお題

- 本日は大体二本立てでお送りします
- 一つは「上級試験用の学習方法」。
  「試験専用」の学習もあまりうまみがないので、実務での実力Upも視野に入れながらお話をします
- もう一つがWebアプリケーションフレームワーク。
  実務では高確率で出てくるので、「まだあんまり触ったことがない」人から「普段なんとなく使っている」人くらいまでをメインターゲットにいくつかお話をします

# 上級試験用の学習方法と教え方

- 上級試験は、「まんべんなく」問題を作成しています
- そのため「幅広い知識」が必要になりますが、同時に「幅広い〝前提〟知識」も必要になります
- なので、今回は「前提」知識のあたりを含めて、「使える」知識を身につけるための四方山をお話できれば、と思います
- 教え方ですが……このレベルになると「教えてくれる人がいる」とも限らないので
  - 独学する人は「ここが学ぶポイント」だと思ってください
  - 教える人は「ここが教えるポイント」だと思ってください

# PHP「以外」をある程度学ぶ

- PHPで多いのはやはり「Webアプリケーション」です
- ただ、Webを作成する場合、それなりにあちこち、前提知識が必要になります
  - 「全部の専門家」にはならなくてもよいですが、「専門家と会話できる」くらいの知識は必要です
- 特に「インフラ」と「DB(RDBとSQL)」「プロトコル」周りは、基礎知識くらいは身につけておくとよいでしょう

# 例えば……

- header() 関数が「どんな風に動いている」のか
- header() 関数等より前にechoすると怒られる理由
- Cookieとセッションの違い

- このあたりは、HTTP(Hyper Text Transfer Protocol)と通信の流れを学習すると理由と仕組みが理解できます
- HTTPのRFCを、最低でも1度は読んでおくとよいでしょう
  - 拡張バッカス・ナウア記法(EBNF)は事前に学んでおきましょう
- 同様に、メール送信まわりは、SMTP(Simple Mail Transfer Protocol)のRFCを読んでおきましょう

- PHPのデフォルト挙動である「セッションはlocalなファイルに保存する」が、そのままだと「Webサーバをロードバランサーで2台以上」にした時に、問題が起き得る
  - L7ロードバランサを使えば回避できますが、対障害性は……
- そのため、session_set_save_handler() 関数を使って「ユーザー定義のセッション保存関数を設定」して、DB等に保存する

- このあたりは「インフラ」「運用」を学習すると理由と仕組みが理解できます
- 「ある程度の規模」と「運用」を見越した設計をしてみましょう

- JavaScriptとPHPとで、変数の値をやりとりする方法
- formのvalueに value=″true″ と書いているのに、PHPで bool(true) で受け取れない理由

- このあたりは、HTTPの「通信の流れ」等を学習すると理由と仕組みが理解できます
  - 端的に、通信は最終的に「文字列」です
- 「通信の流れ」のレイヤーで考えられるようになると色々と不明瞭なところが減ります
  - 紙やホワイトボードなどでの「お絵かき」をお勧めします
  - シーケンス図も大変によいですね(実務で使えますし)

- 「なんとなくこうなる」だと、丸暗記に近くなるのでどうしても限界が来ますし応用もしにくいと思います
- できるだけ「基礎と原理」を学んで、「なぜ?」をクリアに解決していくと、丸暗記ではない「本当の技術力」になると思います
- そのために、必要な「基礎」「土台」を身につけておくと、後々の学習効率が上がります

# 「実装して完成させる」までが学習です

- 簡単でシンプルで小さいものでよいので「完成させる」事が大事です
  - この辺、小説家と漫画家と音楽家の友人からそれぞれ別口で同じ話を聞くあたり、どこも一緒なんだなぁ、と(笑
- 一方で……
  - 「パーツ」を組めないと「全体」が組めない
  - 「全体」を組まないと「パーツ」の重要性がわからない
  - 「なんとなくパーツを作る」と、使われない
  - 「なんとなく全体をくみ上げた」だと「よくわからないおまじないパーツ」が増える、と、応用がきかない
- なので、ある程度「バランスよく」学習して実装しましょう
  - まずは小規模で簡単なものから

# 「要求を仕様に落とす」「仕様をコード設計に落とす」スキルは別途必要

- お仕事で特に「ある程度以上」で必要になるのが……
  - 相手の要求を理解する
  - 要求を仕様に落とす
  - 仕様をコードにする
- 極論、PHP試験で鍛えられるのは「コードの書き方」まで
- でも「コードの書き方」がわかってないと仕様をコードにできないし要求を仕様に落とせない
  - PHP上級試験でしっかり勉強を!! (ダイレクトマーケティング
- 要求のヒアリングと仕様の設計も意識しておこう
  - 「数」と「量」は有効なので、まずはたくさんの経験を!

# コードはまず「読む」もの

- お仕事において「純粋に新規に書くだけ」は少ない
  - 既存コードの理解
  - 既存コードの改修
  - コードレビュー
  - サンプルになるコードの確認
  - その他
- だから、コードは「書く」以上に「読む」事が多いし、なので「(ある程度の速度で)読める」必要がある
  - 単純に「動作が理解できる」、次に「その実装をした意図(やりたいこと)」が読める、くらいまでをまず目標に

- コードを読めるようにするためには
  - 書く側）「読みやすい」コードを書く
  - 読む側）「コードが読める」基礎力を高める

- 「読みやすい」コードの書き方は、例えば「リーダブルコード（O'Reilly）」とかを読んでみましょう
- 「基本的なPHPの構文」がしっかり使えると、（読み手のスキルへの期待もありますが）読みやすくなります

# 例えば、型指定と適切な変数名

- 「適切な**型指定**」は、バグを減らしますがコードを書く時のヒントにもなります
- PHP8で「**名前付き引数**」が使えるようになったので、関数宣言時の変数名もより丁寧にしていきましょう
- $nameって変数名に「住所」が入っていたりすると、いろいろとつらいものがあります(だいたい実話

# 例えば、 break n

- 多重ループの内側で break だけを書くと、直近の1つのループだけ抜けます
- しかし時々「全部のループを一気に脱出したい」ようなロジックがあります
- この時に、例えば三重のネストなら「**break 3**」で一気に抜けられますが、知らずにフラグ制御とかで書くと、冗長で読みにくいコードになります

# 例えば、 ??

- 配列のkeyが「あったりなかったり」するケースや、変数が宣言されていたりいなかったりするケースで、Null合体演算子はとても便利に使えます
  - ?? 自体はPHP7からです
    - $name = $arr['name'] ?? '名無し';
  - Null合体代入演算子は、PHP7.4からです
    - $arr['key] ??= 'default';
- isset() や三項演算子で数行かけて書かれると、やはり少し読みにくいものです

- ほかにも「バージョンが上がって読みやすくなった」文法も多いです
  - 逆に言うと「今まで少し読みにくかった記法でしか書けなかった」という事でもあります
- 一方で「あまり新しい書き方を導入されまくると、学習の浅い人が読めなくなる」というお話も伺います
- 低いレベルに合わせるよりも、せっかくなら(程度問題ではあろうにしても)より改善された書き方を覚えて広めたほうが、自分も他のエンジニアも幸せになれます
  - そのためにも、積極的に「聞く」「教える」文化を
  - PHP上級試験も大変に便利です!(ダイレクトマ略

# 「現状の把握でコードを読む」ことも多い

- ドキュメント……
  - ありますか?
  - 情報足りてますか
  - メンテナンスされてますか?
- 知りたい情報がすべてドキュメントにあればよいですが、なければ「コードから読み解いていく」必要も出てきます
  - 「調査」というやつですね
- そうすると、例えば「普段書くコード」だとあまり使わない知識や関数でも、「解析のためには便利」で知っておいたほうがよいものもいろいろあります

- 調査までいかないにしても、既存コードを「読む」「理解する」「(必要なら)改善する」力もまた、当然に必要になってきます
  - 特に「上級者」になってくると、そういった依頼は増えてきます
- 「読める」ためにも「コードの書き方」を知っている必要があります
- 特に「読む」場合は「ちょっと癖のあるコード」「古い書き方」なども理解しておくとよいでしょう
  - だから、上級試験、PHP7の話も入っています: ダイレクトマ(略
    - PHP5は、問題のボリューム上、入れられなかった orz
      - なお「2〜3年前」に「PHP4」のコードのお話があって以下略

# 小まとめ

- まずは「自分の意図するコードが書ける」ようになる、は一つの目標ですが
- その次には「読みやすいコード」であったり「他人のコードの解読や解析や改善」であったり、といった「質の向上」が不可欠になってきます
- 質量転換(量質転化)と言うように、たくさんのコードを書いたり読んだりするのは、ある程度まで有効な学習方法です
- ただ「闇雲に量を重ねる」だけだと限界が来るので、「質の向上」「新しい知識の導入」を平行させるとよいでしょう

# フレームワーク

- 実務の話を絡めると、Webアプリケーションフレームワーク(以下フレームワーク)は「ほぼ必須」になります
- そんなフレームワークについて学んでいきましょう

# フレームワークの誤解と勘違い

- フレームワークを使うと、誰でも大体同じようなコードがかける
  - 「かけません」。同じフレームワークでも、びっくりするくらい人や現場によって変わります
- フレームワークを使うと品質が安定/向上する
  - 「しません」。書く人次第です
- フレームワークを使うと保守性が上がる
  - 「上がりません」。書く人次第です
- フレームワークを使うとセキュアなコードになる
  - 書き方次第。「セキュアになるようなサポート」はあるけど、いくらでもアンセキュアに書けてしまう

- フレームワークを使うと実装が楽になる
  - 「楽」の定義次第。「自分にとって楽な」「案件の性質に合っている」ものをうまく見つけられれば楽になる
- 開発スピードが向上、開発コストが削減する
  - ケースバイケース。「そのフレームワークに慣れている人」がいるとスピードは向上しやすい

- フレームワークは「便利な道具」ではありますがつまりは「道具」です。
  - 詳しくは後述
- 使う人と使い方次第で、「びっくりするほど便利」から「びっくりするほど駄目になる」までグラデーションがあるので、あまり過信しないようにしましょう
- とはいえ「ちゃんと使う」と「ないよりは圧倒的に便利」である事も事実です
  - 「包丁」とか「ドリル」とか「筆」とか色々な例えができそうです

# フルスタックとマイクロの差異

- 大まかな軸として「フルスタック」と「マイクロ」があります

- フルスタックは「大体、全部入り」です。欲しいもの必要なものは「大体入ってます」
  - いろいろ悩まずにすみますが、「違うもの」が欲しい時のハードルが一気に上がる事が多いです
- マイクロは「大体、なにも入っていない」です。欲しいもの必要なものは「自力で見つけて追加インストール」しましょう
  - そもそも「追加インストール」が前提なので、スキルその他が必要ですが、「好きなものを入れる」事ができます

# 「便利になるためのお道具」

- いうて「便利になるためのお道具」なので、「お道具に振り回されて工数増加、お道具の制限でやりたいことができない」はあんまり意味がない
- フレームワークは「楽をするための便利な道具」ですが、「なにをもって楽とするか」はそれぞれ異なります
  - 便利/楽にも方向性があって。「塩、味噌、醤油、酢、砂糖等の基本調味料が各種そろっている」と「混合調味料が1種類ある」のどちらが便利ですか? という問いがあります。
- 使うフレームワークが「本来提供しようとしている〝楽〟」を見極めましょう
  - その辺を見極めず「Tipsと小細工で無理やり実装実現」すると、後でとても苦労します(実装も、バージョンアップ時も)

- レールがしっかり引かれているフレームワークを使う時は「フレームワークが想定しているレール」と「許容範囲の逸脱と、許容範囲外の逸脱」の境目とかをある程度意識しておきましょう
  - 「あんまり逸脱する」時は、別のフレームワークを検討したほうがよい、か、仕様を変更したほうがよいケースがあります
  - フルスタックは割とこっちが多いです
- レールがあまりないフレームワークを使う時は「メンバーのスキルを確認」。「フレームワークがなくても組めるスキル」がないと、大体失敗します
  - マイクロは割とこっちが多いです

- フレームワークのルール「だけ」だと野放図な傍若無人が大惨事になります。ちゃんと「追加のルール」を決めましょう
  - 個人的には「基本になるサンプルコード一式」をお勧めします
    - 簡単なCRUDの機能一式、あたりを作っておくだけでも違います
- フレームワークを使っていても、根っこは「PHP」です。ちゃんと「PHPの文法」などは理解しながら使いましょう
  - 稀に「Laravelはわかるけど PHPはわからない」というお人が以下略
- 改めて。フレームワークは「お道具」である事をしっかり意識しておきましょう

# MVCとDDD(クリーン/オニオンアーキテクチャ)

- 久しくMVC(Model View Controller)が賑わいを見せておりました
  - 「Modelとは?」という議論も色々ありますが棚の上に
- しかしここ数年、DDD(Domain-Driven Design/ドメイン駆動設計)が流行り始めました
  - アーキテクチャとしては、「クリーンアーキテクチャ」または「オニオンアーキテクチャ」が採用される事が多いようです
- フレームワークによっては、クリーン/オニオンアーキテクチャなディレクトリの振り分けで「少し思案する」ケースもあるので、そのあたりも意識しておくとよいでしょう

# フレームワークのコードを読む

- いずれのフレームワークも、(大体は)PHPでコードが書かれているので。深く理解する時は「そのフレームワークの実装」を見たほうが早い事があります
  - Phalconとか例外もありますが
- フレームワークのソースコードは「PHPの技術力がある程度しっかりしていないと」読めないので、ちょうどよい試金石でもあります
- 「普段、実務だとあまり使われない(けど知っていると便利な)テクニックや文法や書き方」も多いので、"フレームワークのソースコードを読む"という事も視野に入れておくとよいでしょう

# 運用について考える

- PHP自体もそうですが、フレームワークや「その中で使われているライブラリ」(や独自に入れたライブラリ)も、ちゃんとバージョンアップをしていかないといけません
  - 別言語ですが、フレームワークのバージョンアップ周りの事故で割と顕著だったのはたとえば「都税サイト 不正アクセス 」
    - Apache Struts 2の問題でしたねぇ
    - 「都税サイト」が顕著ですが、ほかにも結構なサイトがやられてました
  - ライブラリ単体ですと、Log4j2辺りが記憶に新しく……
- バージョンアップの頻度、バージョンアップにかかる労力、現在のメンテナンス頻度などを確認しておきましょう
  - 所謂「メジャーどころ」はこの辺が安定している可能性が高いので、選択しやすいんですよね

# 紹介

- 最後に、いくつか「今、比較的耳にする事の多い」フレームワークを紹介していきます
  - 偏りがあるのは「古庄の思考と思想とバックボーン」起因だと思っていただければと思います

# Laravel

- 現在「ぶっちぎりトップ」を走っているフレームワークです
- フルスタック系
- よく使われていますが、同じくらい「割と現場によって使い方が異なる」ので、いささかの注意が必要です
  - バージョンもいろいろと違っていて、悩ましい事も多いですね
- 大体なんでもできますが、時々「重い」事もあるので、ある程度以上になるとその辺のチューニングとかも考慮したほうがいい事があります
- 軽量版のlumenというのがあった、のですが、2024/04にGitHubリポジトリがアーカイブ化されてますね……
- 「普通のものを作る」のにはとても便利です

# Symfony

- 以前はもう少しよく耳にした気もするのですが、最近、案件で耳にする事も減ってきたように思われます
- Symfonyは「Webアプリケーションフレームワーク」ですが、同時に「Webアプリケーションフレームワークを作るためのフレームワーク」でもあります
  - Laravelも、Symfonyベースです
- なので、Laravelのコードを深く調べると、定期的に「あ、これSymfonyか」となります
- なので、実は「結構みなさんお世話になっている」縁の下の力持ちさんです

# CakePHP

- 「海外では……」というお話も時々伺いますが、日本だと「そこそこのシェアがある」フレームワークです
- 歴史のあるフレームワークなので(ほかも割とそうですが)メジャーバージョンが変わると結構大きく変わってくるので注意が必要です
  - 現在は 5 が最新です
- フレームワーク自体が明示的に「MVC」を推しています
  - 基本「便利」なのですが、最近はDDDが流行っている事もあって……
- (Laravelもですが)Ruby on Railsインスパイアである、と言われています

# Slim

- 現在「個人的に最も推している」フレームワークです
  - 今まで紹介した中では唯一の「マイクロフレームワーク」
- びっくりするくらい「何もない」ので、必要なものは大体「自分で実装する」必要があります
  - なんで個人的にも「よく使うお道具」を開発公開しています
- 「レールがない」ので初めは苦労しますが、「どこに行っても適度に茨道」なので、「一風変わった案件」とは相性がよい、と思っています
  - 「レールがしっかりした」フレームワークだと、あんまり外れた案件は実装がめちゃくちゃ大変です……
- 「フレームワークのコード」も軽いので読みやすい

- このあたりまでが(1つを除いて)わりと有名だったりポピュラーだったり案件がそれなりにあったりします。
- 以降、それ以外で「比較的目にする」ものを、ざっくりと

- ZendFramework
  - PHP謹製。……いけるかと思ったら沈みました(2020年にGitHubアーカイブ行き)
- FuelPHP
  - 一時期は大人気だったのですがその後急降下。メンテナンスが年単位で止まっているので……
- CodeIgniter
  - FuelPHPとご縁が深い。ギリギリメンテナンスされているけど活発とは言いがたい感じなので、どうなるか……

- Yii
  - 日本だとあまり耳にしないように思います。10年くらい前に、一度日本でも「まれに耳にする」感じだったのですが、それ以降は……
- Phalcon
  - 「PHPエクステンション(C言語)で書かれた高速のフレームワーク」で一世を風靡……するほど伸びなかったですね。メンテナンスはまだコンスタントに続いているようです
- その他
  - ほかにも色々ありますので、よかったら調べてみてください

# フレームワークの学び方

- まずはどのフレームワークにしても「簡単なWebアプリケーション」を作成してみるとよいでしょう
  - 「DBへの読み書き」で片付く程度のもの(BBSとか)、次に「認証が絡む」もの(ToDoリストとか)、あたりが無難かと思います
- 上述を一度作ったあと、改めてフレームワークを調べて「そのフレームワークらしい書き方」を再度学習しましょう
  - そのフレームワーク「らしさ」はとても大切です
- 多少理解できてきたら次は……
  - より複雑なものを作ってみる
  - 別のフレームワークを触ってみる
  - フレームワークのコードを読んでみる

# まとめ

- PHPの上級試験自体は「PHPの言語を学ぶ」事で全ての範囲をリーチできますが、そのためには「割と広めの"PHP以外の知識"」が重要になります
- ただ一方で「PHPの知識」はやはりそれらの土台の1つでもあります
- せっかくなら「ただ学んだ」ではなく「実務で有用な知識を得る」ために、色々な知識を学んでいただければと思います

- そんな学習の一助になれば、望外の喜びです!!