やさしくつつみこむヨドバシ API

2024年12月05日

ヨドバシリテイルデザイン 戸田

概要

いつもご利用ありがとうございます。

お客様を第一に考えるヨドバシカメラでは、これからも
- 安心できる商品を
- 確実・迅速にお客様にお届けいたします。

この実践には
- 信頼できる取引先様との円滑な連携
- ヨドバシエクストリーム配送メンバーや配送業者との円滑な連携

が必要です。

ヨドバシでは、こういった業務ドメインごとの Public API を準備しております。

ヨドバシが提供する API は「優しさ」を重視し、
誰もが活用しやすい REST API をご用意いたします。

また、ヨドバシ内部の API 開発においては「惑わない」を方針としています。
開発者が、課題解決に集中できるように、多角的にサポートされる環境を用意しています。

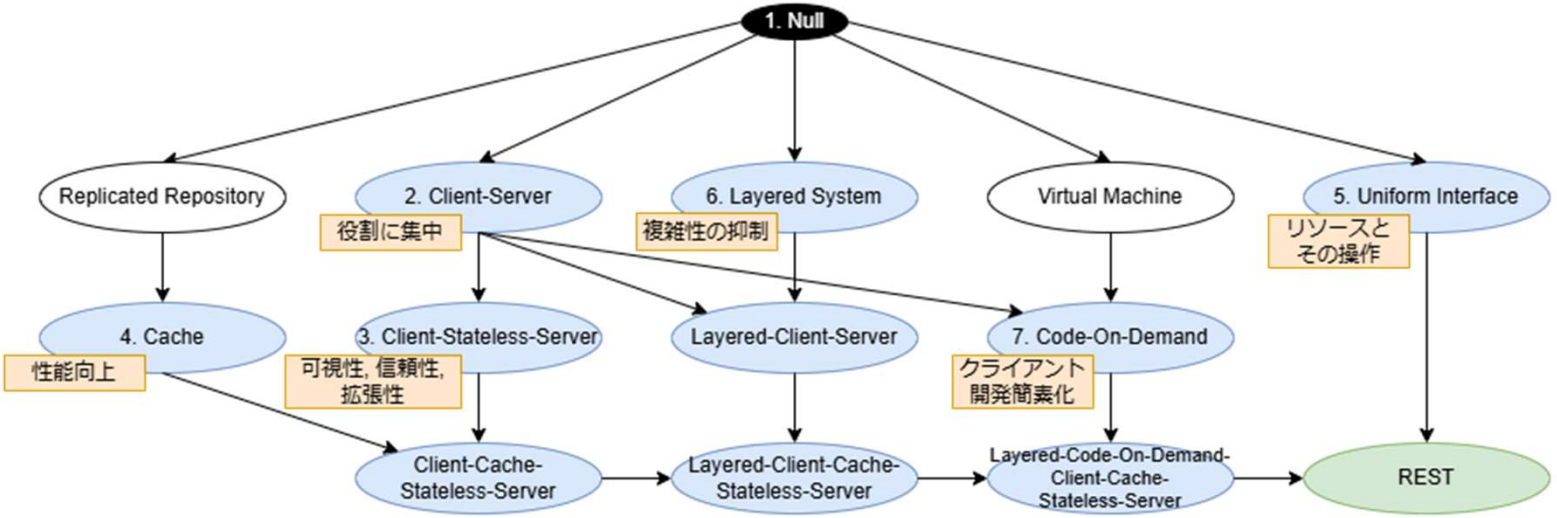
本講演では、過去の開発を振り返りながら、これらの取り組みについてご紹介いたします。

やさしい API　1/7

「優しさ」とは相手を思いやり、実践することです。

API 利用者にとって、すぐに理解でき、都合にあわせて、すぐに使える API が理想的です。

そこで我々は REST API を採用しました。



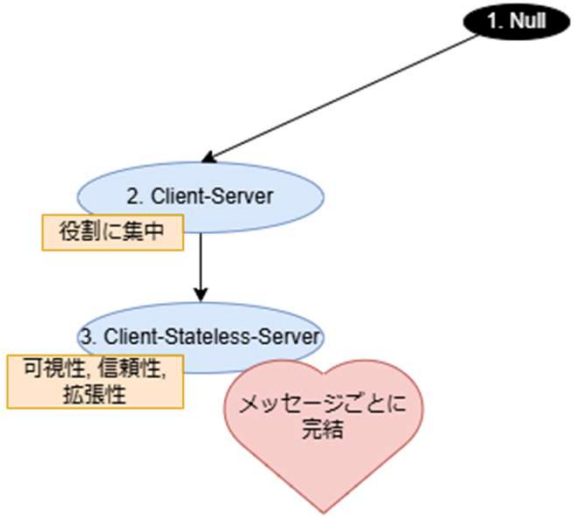
ここからは古典中の古典 Roy T. Fielding さんの「Architectural Styles and the Design of Network-based Software Architectures」をなぞりながら「優しさ」とヨドバシの取り組みをご紹介いたします。

やさしい API　2/7

まずは 1. Null Style から 3. Client-Stateless-Server 。

Stateless ＝ 文脈をもたない ＝ メッセージごとに完結

　複雑な手続きが不要で、メッセージをみれば内容がわかり、やり直しができる。・・・やさしい



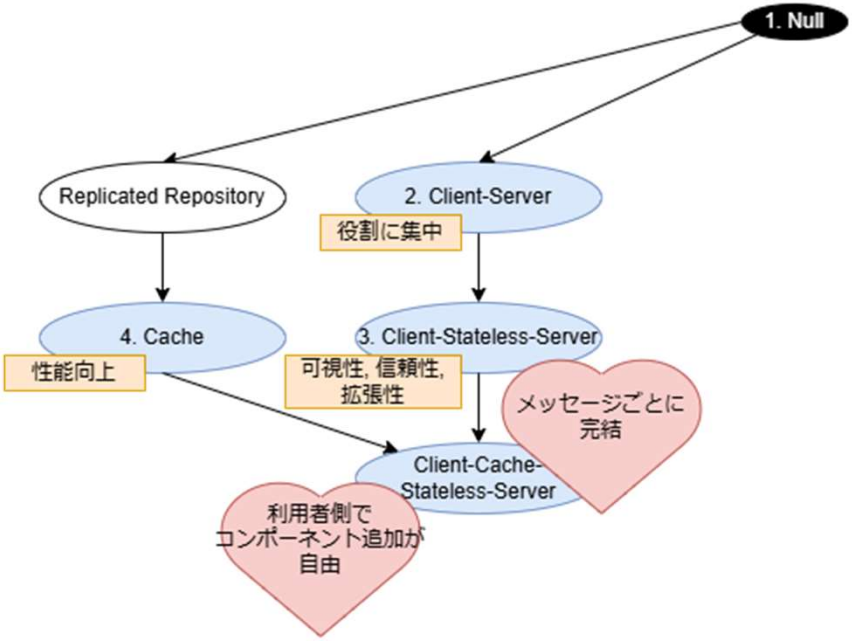
Roy T. Fielding
Architectural Styles and the Design of Network-based Software Architectures より

やさしい API　3/7

つぎに 4. Cache から Client-Cache-Stateless-Server 。

Stateless ＋ Cache ＝利用者側で自由にコンポーネント追加が可能

利用者側で Cache や Proxy Server を追加でき性能・安全性などの非機能要件を考えられる・・・やさしい



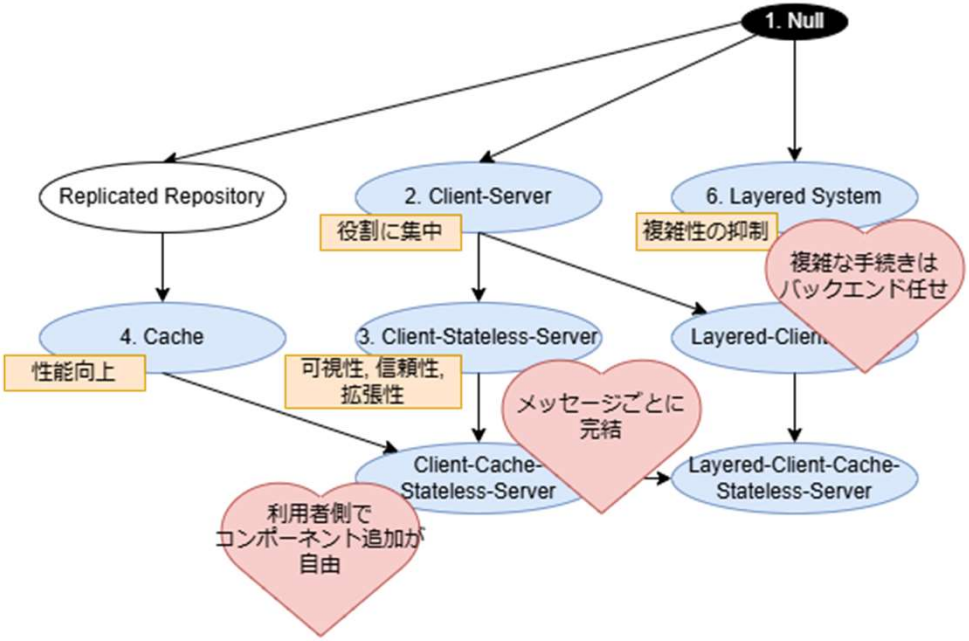
Roy T. Fielding
Architectural Styles and the Design of Network-based Software Architectures より

やさしい API　4/7

5 を飛ばして 6. Layered System から Layered-Client-Cache-Stateless-Server 。

Layered + Stateless + Cache = メッセージ単位で隠ぺいされた複雑な業務ロジックを利用

複雑な業務を理解することなく、メッセージのみで依頼や結果を得られる。・・・やさしい



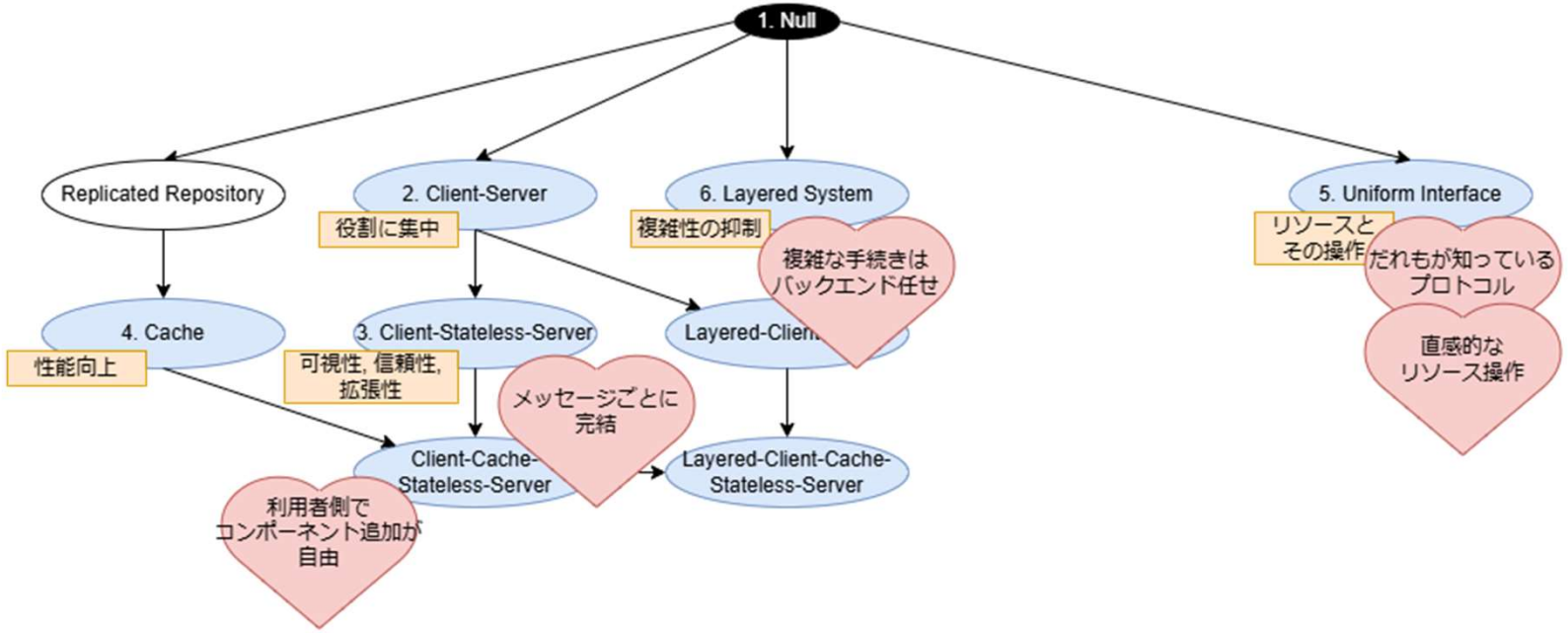
Roy T. Fielding
Architectural Styles and the Design of Network-based Software Architectures より

やさしい API　5/7

もっとも大切な優しさ 5. Uniform Interface。

Uniform Interface ＝ HTTP、URI(Uniform Resource Identifier)

だれもが知っているプロトコルで、対象リソースが明確、操作も明確・・・やさしい



Roy T. Fielding
Architectural Styles and the Design of Network-based Software Architectures より

惑わない API 開発　1/9

ここからは外面の「優しさ」をいったん忘れて、我々ヨドバシの API 開発のプロセスについてご紹介します。

5. Uniform Interface を掘り下げます。

HTTP とは決めたものの、まだまだ自由度がありすぎて、開発者は迷子になりそうです。

その中でももっとも悩ましいのは

URI(Uniform Resource Identifier) で指し示す リソース を適切に定めること。

これが、直感的に使いやすい API を提供できるか否かの決め手になります。

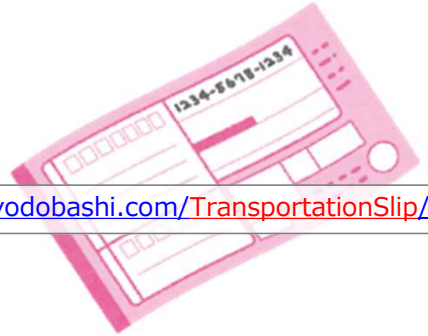
惑わない API 開発　2/9

ここで問題です。

配送ドメインで「荷物」は明らかにリソースです。

では「配送伝票（送り状）」はリソースでしょうか？それとも「荷物」の属性でしょうか？

GET http://yodobashi.com/Laggage/1234



GET http://yodobashi.com/TransportationSlip/z0091

GET http://yodobashi.com/Laggage/1234

```
{
   “LaggageId” : 1234,
   “SendTo” : {
      “Place” : “新宿５丁目”,
      “DateTime” : “2024-12-05 14:25”
   }
}
```

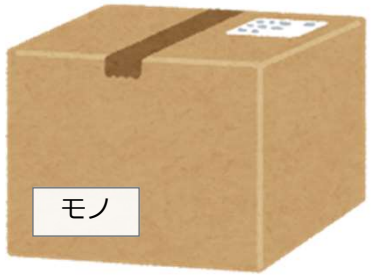
惑わない API 開発　3/9

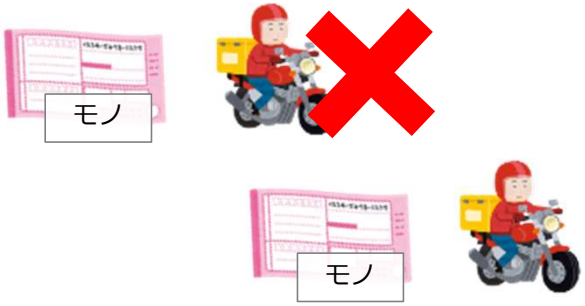
正解は「どちらでもいい」です。

回答１．物理的に存在する荷物はリソース、配送伝票は情報なので属性


```
GET http://yodobashi.com/Laggage/1234
{
   “LaggageId”：1234,
   “SendTo”：{
      “Place”：“新宿５丁目”,
      “DateTime”：“2024-12-05 14:25”
   }
}
```

回答２．配送伝票は配達ごと・配達業者ごとで再発行されます。リソースだと管理しやすそうです。





GET http://yodobashi.com/Laggage/1234

GET http://yodobashi.com/TransportationSlip/z0091

GET http://yodobashi.com/TransportationSlip/q0207

不都合があり配送業者を変更すると配送伝票はあらためて作成されます。

マイクロサービス設計の肝 Bounded Context を定義するのに伝票はとても有用です。

伝票はサービスの要件定義書と捉えることができ・・・という楽しい話は別の機会に（ざんねん）

惑わない API 開発　4/9

ただ「どちらでもいい」のままでは API サービスは提供できません。

判断が必要ですが、この領域の判断はエンジニアには難しいかもしれません。
となると、早めにステークホルダーに相談し方針を合意したほうががいいですね。

合意が遅れると「顧客が本当に必要だった」タイヤのブランコはおあずけになります。
方向を間違えたまま時間とコストとをかけてしまうことになります。

作品名
リキテンスタインが本当に必要だったもの　<作者 Gemini>

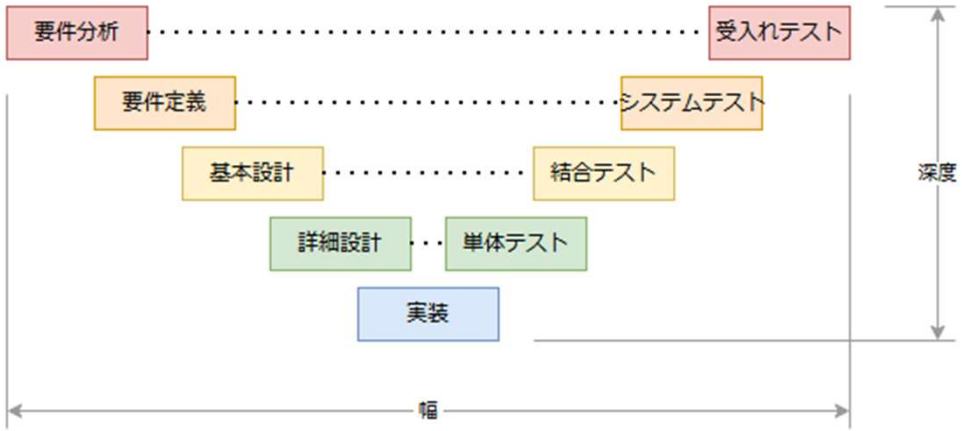
そういえば・・・気がつけばもう５３歳ですよ・・・

惑わない API 開発　5/9

「顧客が本当に必要だったもの」の原典はミルト・ブライスが 1971 年に発表した
　PRIDE（PRofitable Information by DEsign - through phased planning and control）
だそうです。
私と PRIDE とは同級生かぁ・・・

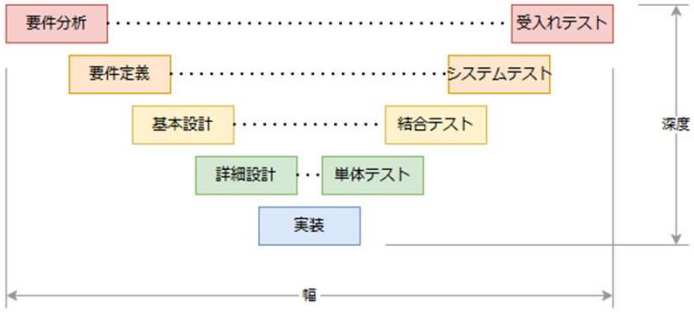
振り返ると、ソフトウェア開発史は「本当に必要だったもの」ギャップとの闘いだったように思います。

後知恵で考えなおすと、V-Model の谷を浅く狭くすればギャップは小さくなると仮説を立てたと言えそうです。



惑わない API 開発　6/9

V-Model の谷を浅く狭くすればギャップは小さくのは確かですが・・・



聴き取った要件をオブジェクトとして実装すれば、V-Model の谷は浅くできそうです。
タイヤ クラス、ロープ クラス、木の枝 クラスを実装すればいい・・・オブジェクト指向言語は救世主ですね。

聴き取った要件をチームメンバーが自発的に造ってすぐにレビューできれば、谷は狭くできそうです。
タイヤ担当、ロープ担当、木の枝担当が細かいことは後回しでデプロイ・・・アジャイルの環境は天国ですね。

あとはこれをワークするだけですから簡単ですね、要件を聴きとれる能力があって頭の中で基本設計・詳細設計する能力があって実装できる能力とデプロイできる能力さえあれば誰でもできますからね。

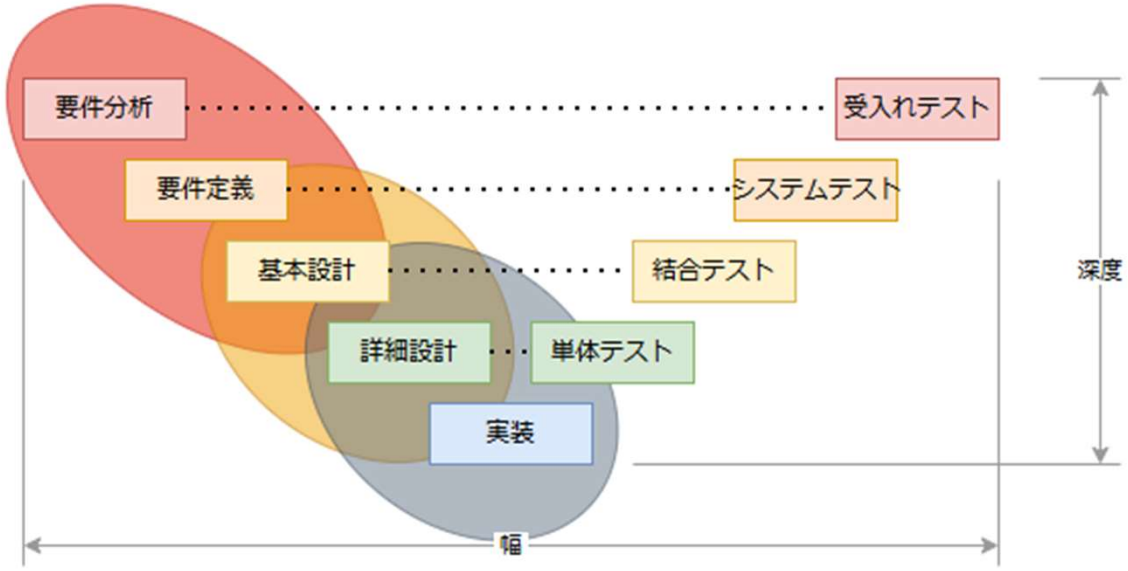
これは「デキる人が考えたデキる人が実践できる」ものなのかも知れませんね。それは残酷かも。
(優秀なスクラムマスタの元では初級者ばかりのチームでもクリアできますが、自分の経験では成功例は３例くらいでした)

惑わない API 開発　7/9

誰だって最初から全フェーズをこなせる能力はありません。
ではどうするか？どうやってギャップを解消するのか？どう効率を上げるのか？

我々はこうします。
- 基本設計までは標準規約と設計ツールを利用し進めます。
- 「基本設計」以上「詳細設計」未満のリソース定義はステークホルダーに付き合ってもらいます。
- 詳細設計以降はヨドバシアーキテクチャを考慮したコードジェネレータを使います。

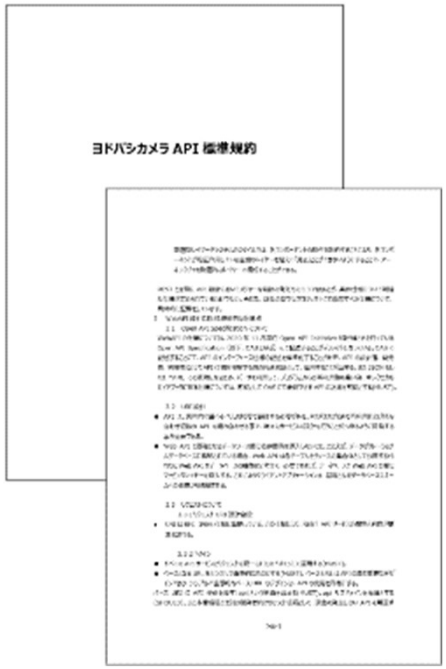
開発者はこの一本道を惑うことなく突き進みます。



惑わない API 開発　8/9

基本設計までは標準規約と設計ツールを利用する

本来の取り組むべき課題に設計者が集中できるように API 標準規約を策定しました。
この策定にあたっては極めて一般的な標準規約となるように各種 RFC を取り込むかたちで
ブリスコラ様と協議を重ねて作り上げました。
また、設計ツール BAMs Design にもこの API 標準規約を実装し設計をガイドしてくれるようにもなっています。
さらに、設計者ごとの同義語・類似語ユレを解消するため辞書ツールも併せて活用しています。



惑わない API 開発　9/9

「基本設計」以上「詳細設計」未満のリソース定義はステークホルダーに付き合ってもらう
詳細設計以降はヨドバシアーキテクチャを考慮したコードジェネレータを使う

設計ツール BAMs Design から生成された OpenAPI Specification は、
API 設計書公開サーバーである BAMs Catalog で共有されます。

ステークホルダーは BAMs Catalog を閲覧することで、リソースの切り口やモデルについて確認ができます。

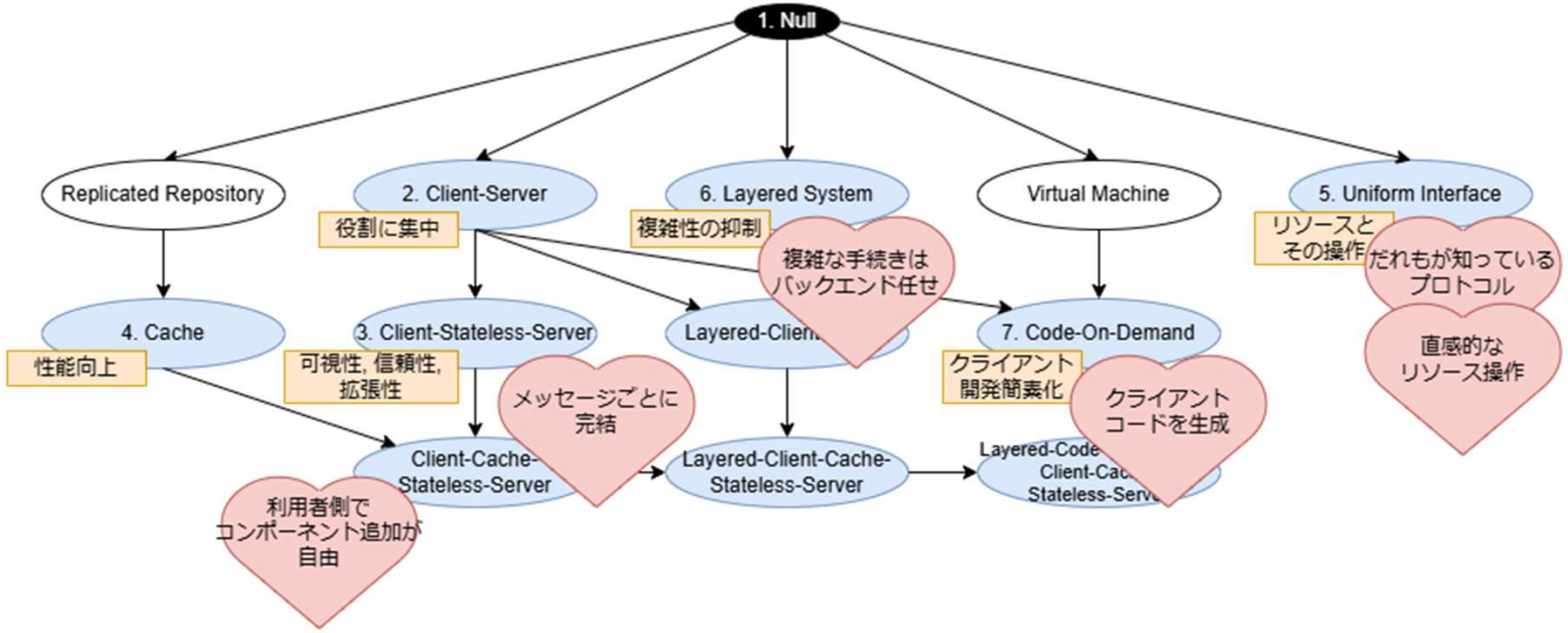
また、BAMs Design には OpenAPI Specification ファイル（openapi.yaml）からヨドバシアーキテクチャに最適化されたコードジェネレータも内蔵されており、開発生産性や品質向上に寄与しています。

やさしい API　6/7

最後の優しさは 7. Code-On-Demand。

Code-On-Demand = サーバーからクライアントコードを提供

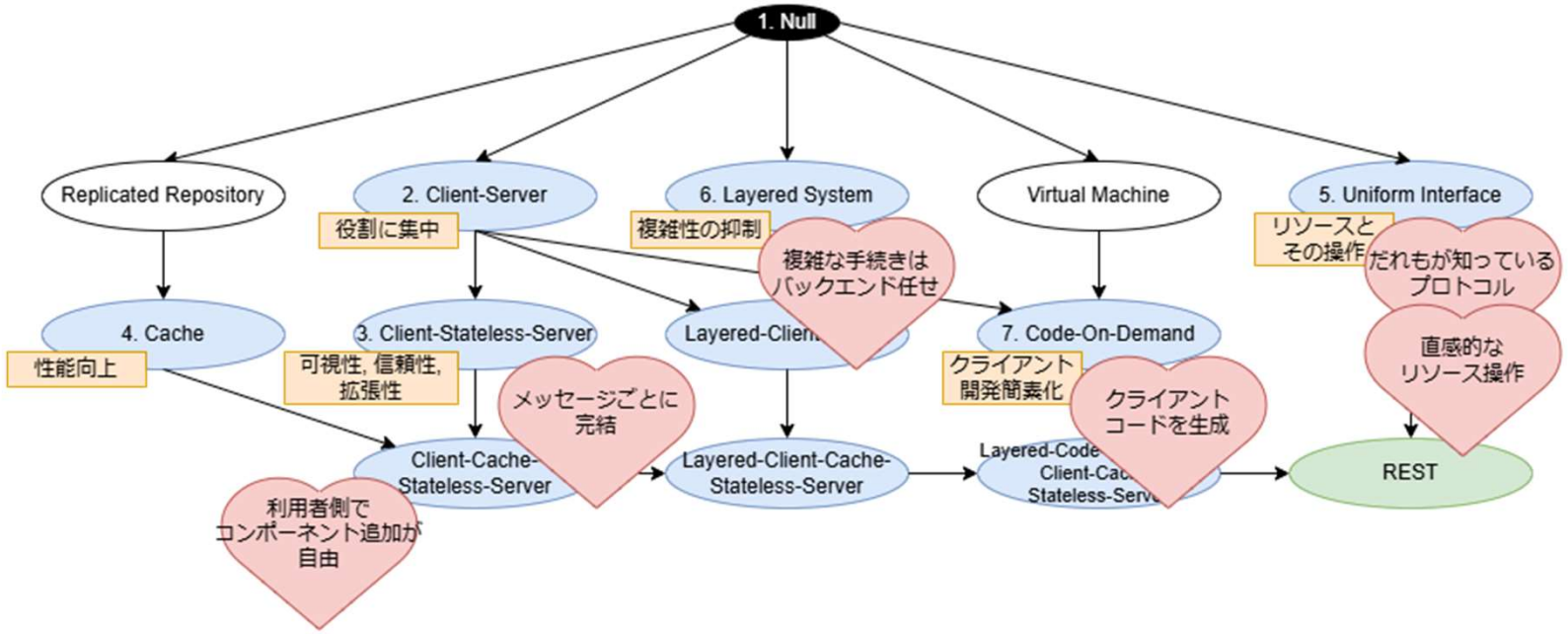
ヨドバシは Code Generation 可能な IF 仕様書 OpenAPI Specification を提供します・・・やさしい
（元論文の文脈とはちがいますが）



Roy T. Fielding
Architectural Styles and the Design of Network-based Software Architectures より

やさしい API　7/7

こんなに多くの「優しさ」でつつむヨドバシの Public API。



Roy T. Fielding
Architectural Styles and the Design of Network-based Software Architectures より

最後に

ヨドバシではおもしろい経験ができます。

自分たちの手で新しいサービスを世に送り出せます。

公開した Public API はさらに新しいビジネスを創ります。

システム開発をするだけではありません。

システム開発のための土台も自分たちで産み出せます。

Public API を利用したことはあっても、自分の API が見知らぬ誰かに使われたことは？

我々はこの興奮を共有できます。

ヨドバシのサービスを一緒にビルドしてくれる仲間を募集しております。

https://yodobashi.jobs

ほんとに最後に

ありがとうございました。

www.yodobashi.com