# 解説動画作成ツールをScalaで作成している話 / Creating an explainer video maker in Scala

@windymelt at ScalaMatsuri (2023-04-15, 2023-04-16)

# You can get this slide via…

- https://www.3qe.us/2023/04-scalamatsuri-zmm-movie-generator (my site)
  - Scan QR Code
- Subject to be changed.

上記のURLからこのスライドをダウンロードできます

# About me

- Windymelt
  - www.3qe.us
  - @windymelt@plrm.capslock.dev
  - @windymelt (at almost everywhere in the world)
- Scala, at work and private
- Web engineer at https://hatena.co.jp
  - Mainly developing backend server
- Developing Hatena Bookmark (social bookmark service)
- We're hiring!!

株式会社はてなで働いています 採用強化中

# About me (2)

- Currently interested in…
  - Scala (of course!)
  - Fediverse (Mastodon, Misskey, ActivityPub…)
  - Cloudflare Worker
  - ffmpeg
  - APEX Legends

Fediverseやffmpegなどに興味があります

<AD>

# [AD] Hatena Bookmark

- Social Bookmark Service
- You can comment, share, discover bookmarks
- https://b.hatena.ne.jp


はてなブックマークというサービスを展開しています

# DISCLAIMER!

- Product introduced in this talk has no business with my employer
- That’s my own hobby products

今回の話は勤務先とはぜんぜん関係ない趣味の話です

</AD>

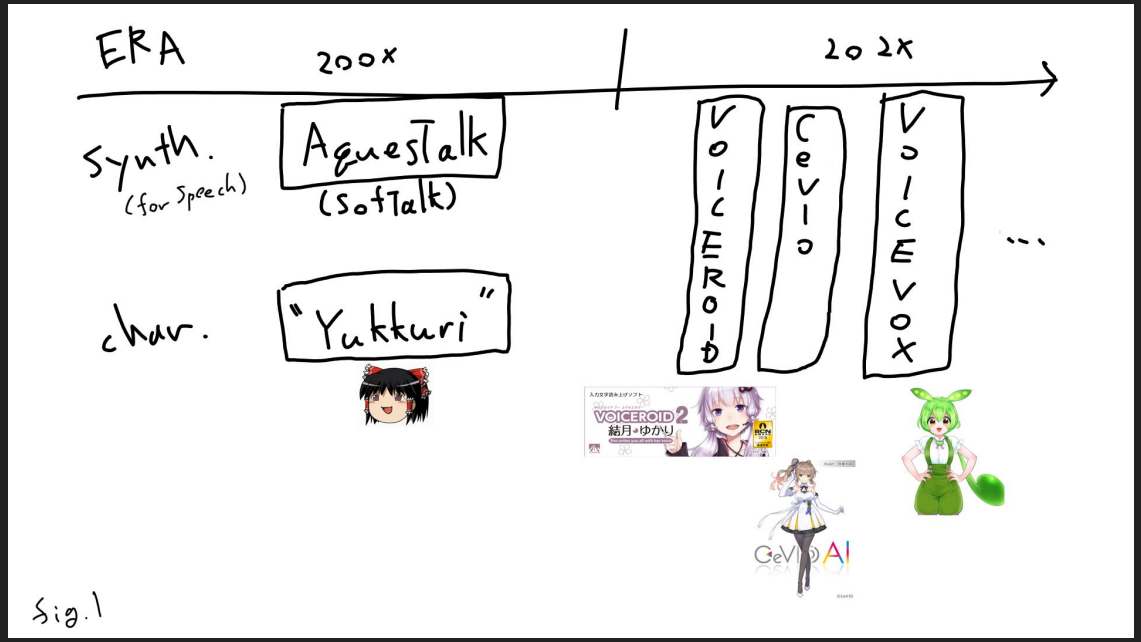
# Prerequisites – Explainer video

- Do you know *Explainer video (解説動画)* ?
- Popular videos genre in Japan
- ```
  type CTM = (
    Synthetic speech,
    Character,
    Video explaining something,
  )
  ```

解説動画は合成音声とキャラクター、解説内容から成る
  ```


活発な解説動画文化が日本に存在する

# Synthetic Speech Technology (in ja_JP)




最近は合成技術がキャラクターとセットで提供されがち

# What is talked in their videos

- Everything
  - Cooking recipe
  - History
  - Technology review
  - Gaming (RTA, etc.)
  - Politics, Economy, etc.
  - Just a talk
- Each Synthesize engine / Character have their guideline
  - e.g. “Cannot be used for political activity.”

内容は様々だが、キャラクターにはガイドラインがある

# Two problems

動画文化が抱えている2つの問題

# Knowledge and Video editing skill is not equal

- Knowledge
  - Skill / Experience with specific domain
  - Have Ph.D
  - Talk skill
  - etc.

≠

- Video editing
  - Video editor skill
  - Encoding skill
    - ffmpeg hell
  - Creating screen format
  - etc.

優れた知識があっても動画作成が上手とは限らない

# Disadvantage for video

- Insufficient searchability
  - You can never search YouTube contents using its spoken words (now)
- Influencer gains more power
  - If you cannot search it, You have to rely on influencer
  - Influencer dominancy
- We have to watch all through a video to confirm whether we want it

検索しにくくインフルエンサーが力を持つ状況

# So I made ZMM

https://github.com/windymelt/zmm

問題解決のためにZMMを作った

# ZMM

- XML => MP4 CLI application
- Docker image available
- Stands for “Zunda movie maker”
  - came from a character name
  - respecting existing (but complex) movie maker
- Demo?
  - Caveat: VOICEVOX does speak only Japanese
  - https://www.nicovideo.jp/watch/sm41810130
  - https://github.com/windymelt/zmm/blob/main/docs/sm41810130.xml

原稿XMLからMP4を生成するアプリケーションです

# Example input

```xml
<?xml version="1.0" encoding="utf-8" ?>
<content version="0.0">
  <meta>
    <voiceconfig id="metan" backend="voicevox">
      <voicevoxconfig id="2" />
    </voiceconfig>
    <voiceconfig id="zundamon" backend="voicevox">
      <voicevoxconfig id="3" />
    </voiceconfig>
    <characterconfig name="metan" voice-id="metan" serif-color="#E14D2A" tachie-url="../../assets/metan
    <characterconfig name="zundamon" voice-id="zundamon" serif-color="#379237" tachie-url="../../assets/
  </meta>
  <dialogue backgroundImage="../../assets/background.jpeg" bgm="assets/bgm.mp3">
  <say by="zundamon">こんにちはなのだ</say>
  <say by="metan">ずんだもん、こんにちは</say>
  <say by="zundamon">難しい操作は不要なのだ</say>
  </dialogue>
</content>
```

# 原稿XMLの例

# ZMM – Features

- Automatic subtitle and voice from XML
- Background image
- Code highlighting
- Mathjax support
- Modularized synthesis engine
  - Currently only VOICEVOX supported
- Who need ZMM?
  - Who have academic background
  - Who doesn’t have video editing skills
- Aiming Plain and Easy

字幕生成やコードハイライト、数式表示などを搭載

# Why XML?

- Scenario contains tree structure
  - e.g. Different character talks, but same background image and BGM
- XML is eXtensible Markup Language
  - Other tool can insert own tags
  - ZMM just ignores unsupported tags
- XML can insert code snippet
  - CDATA section is useful for code snippet

```
<code id="voice" lang="sh">
  <![CDATA[
$ java -jar zmm-0.2.0.2.jar show voicevox
  ]]>
</code>
```

XMLは動画原稿に適切なフォーマット

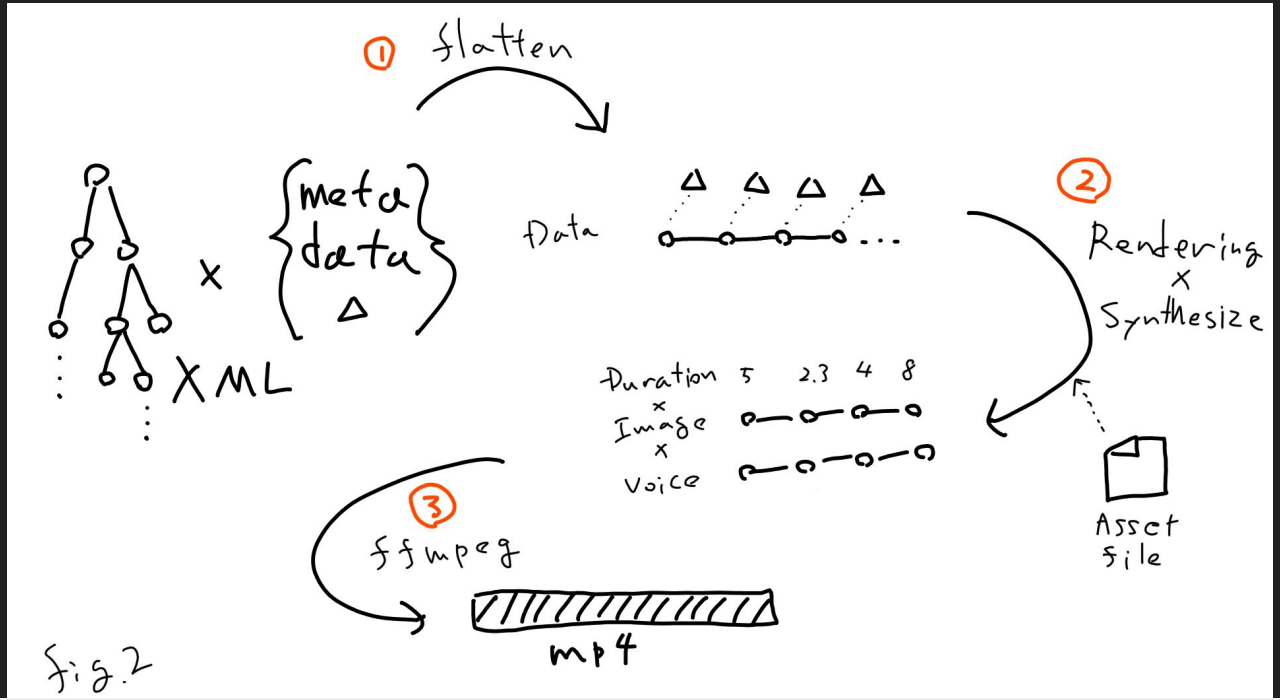
# ZMM – Architecture

ZMMのアーキテクチャ

ZMM: XML ⇒ MP4

ZMMはXMLをMP4に変換するアプリケーション

# ZMM – Architecture




全体像で見るとこんな感じ

# Three Part of ZMM (Simplified)

- Flattener
  - XML => Seq[(Sentence, Context)]
  - *In ZMM, we call Metadata as Context*
- Materializer
  - Seq[(Sentence, Context)] => Seq[(Image, Wav)]
- Concatenator
  - Seq[Image, Wav] => MP4

ZMMの3要素: 平らにする・変換する・結合する

# Three Part of ZMM (Much More Simplified)

- Flatten
- Map
- Combine

Scala風に言うとflatten, map, combine

“OK I love them”

Scalaが得意とする分野ですね

# 1. Flatten

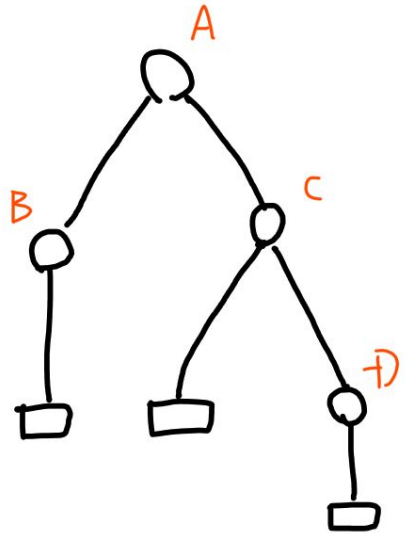
まず平らにする所から見ていきましょう

# Context inside tree

```
<dialogue backgroundImage=”A”>
  <scene backgroundImage=”B”>
    <say by=”char1”>こんにちは</say>
  </scene>
  <scene backgroundImage=”C”>
    <say by=”char1”>こんにちは</say>
    <scene backgroundImage=”D”>
      <say by=”char1”>こんにちは</say>
    </scene>
  </scene>
</dialogue>
```

原稿XMLの各場所にはコンテキストがある
コンテキストは画像や速度などの設定のこと

m = background Image

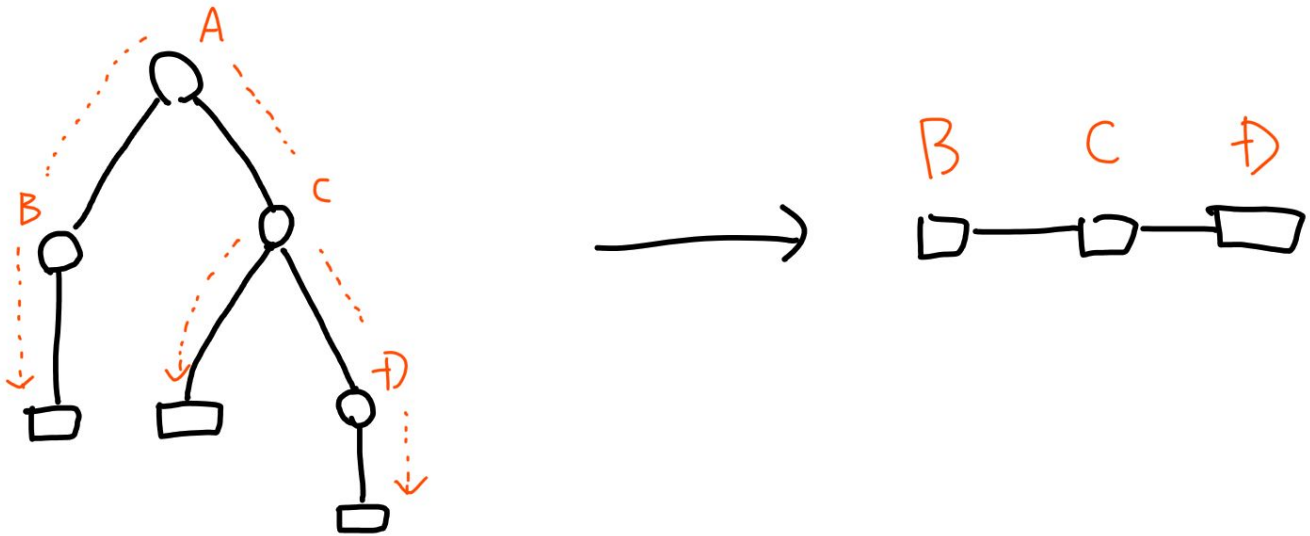



図にすると、背景情報がノードに乗っている感じ

# Flatten it

これを平らにする

m = background Image




平らにすることで、動画にする準備ができる

# How XML should be flatten?

- Principle: Child context overrides parental context
- Complecation: context is not only backgroundImage
  - Actually context is a big `case class` →
  - Talking speed, character, BGM, etc.
- We have to merge two contexts safely
- Merged context is also a context
  - 🤔 Merge…? Binary operation…?

```
final case class Context(
    voiceConfigMap: Map[String, VoiceBackendConfig] = Map.empty,
    characterConfigMap: Map[String, CharacterConfig] = Map.empty,
    backgroundImageUrl: Option[String] = None,
    spokenByCharacterId: Option[String] = None,
    speed: Option[String] = Some("1.0"),
    serifColor: Option[String] = None, // どう使うかはテンプレート依存
    tachieUrl: Option[String] = None,
    dict: Seq[(String, String, Int)] = Seq.empty,
    additionalTemplateVariables: Map[String, String] = Map.empty,
    bgm: Option[String] = None,
    codes: Map[String, (String, Option[String])] = Map.empty, // id →
    maths: Map[String, String] = Map.empty, // id → LaTeX string
    sic: Option[String] = None, // 代替読みを設定できる(数式などで使う)
    // TODO: BGM, fontColor, etc.
) {
```

CSSのように、子のコンテキストが親の属性を上書きする
しかも、安全に行う必要がある

Context is a `Monoid` !!!

コンテキストはモノイドになることに注目

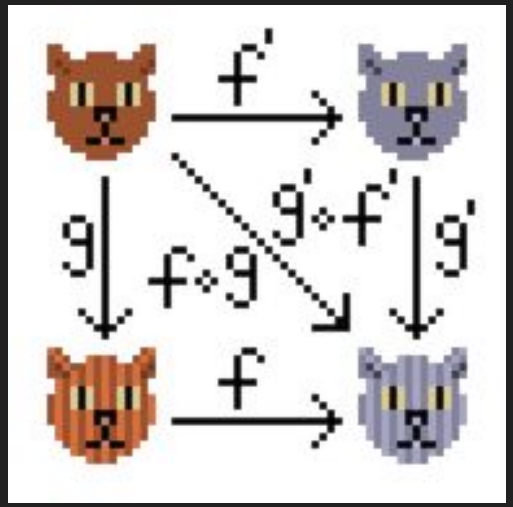
# Monoid for beginners

- A mathematical structure
  - There should be binary operation `|+|` (`combine`) for the Monoid
  - Monoid should have empty value ∅
    - ∅ |+| m1 = m1 = m1 |+| ∅
  - `(m1 |+| m2) |+| m3 shouldBe m1 |+| (m2 |+| m3)`
- An useful and common structure
  - There’re good libraries to handle these common structure
    - e.g. Cats, Scalaz, etc…
- Many things in the world is a Monoid
  - List
  - Option
  - …

モノイドとは空要素と結合操作と結合則を持つ型のこと

# Handling Context using Cats

- Cats: Functional programming library for Scala
- Implementing Monoid instance for Context




Catsライブラリを使ってモノイドとしての
コンテキストを扱う

```scala
implicit val monoidForContext = new Monoid[Context] {
  def combine(x: Context, y: Context): Context =  {
    val spokenByCharacterId = y.spokenByCharacterId |+| x.spokenByCharacterId
    val characterConfigMap = x.characterConfigMap ++ y.characterConfigMap
    val serifColor = y.serifColor orElse x.serifColor orElse spokenByCharacterId
    val tachieUrl = y.tachieUrl orElse x.tachieUrl orElse spokenByCharacterId.fl
    Context(
      voiceConfigMap = x.voiceConfigMap ++ y.voiceConfigMap,
      characterConfigMap = characterConfigMap,
      backgroundImageUrl =
        y.backgroundImageUrl orElse x.backgroundImageUrl, // 後勝ち
      spokenByCharacterId = spokenByCharacterId,
      speed = y.speed orElse x.speed, // 後勝ち
      serifColor = serifColor,
      tachieUrl = tachieUrl,
      dict = y.dict |+| x.dict,
      additionalTemplateVariables = x.additionalTemplateVariables ++ y.additiona
      bgm = y.bgm orElse x.bgm,
      codes = x.codes |+| y.codes, // Map の Monoid性を応用すると、同一idで書かれたコ
      maths = x.maths |+| y.maths,
      sic = y.sic orElse x.sic,
    ) ← #61-76 Context
  } ← #56-77 def combine(x: Context, y: Context): Context =
  def empty: Context = Context.empty
} ← #55-79 implicit val monoidForContext = new Monoid[Context]
```

# がんばってコンテキストがモノイドであることを示す

🤮 (2)

```scala
class ContextSpec extends AnyFlatSpec with Matchers {
  // Verify that Context is a Monoid
  "Context" should "satisfy associative law" in {
    val x = Context(
      Map("zundamon" → VoiceVoxBackendConfig("3")),
      Map("z" → CharacterConfig("z", "zundamon")),
      Some("https://example.com/bg1.png")
    ) ← #9-13 val x = Context
    val y = Context(
      Map("metan" → VoiceVoxBackendConfig("2")),
      Map("m" → CharacterConfig("m", "metan")),
      Some("https://example.com/bg2.png")
    ) ← #14-18 val y = Context
    val z = Context(
      Map("tsumugi" → VoiceVoxBackendConfig("8")),
      Map("t" → CharacterConfig("t", "tsumugi")),
      None
    ) ← #19-23 val z = Context

    import cats.implicits._
    { (x |+| y) |+| z } shouldEqual { x |+| (y |+| z) }
  } ← #8-27 "Context" should "satisfy associative law" in
      windymelt, 4 か月前 • Contextの仮実装
  it should "have identity element" in {
    val x = Context(
      Map("zundamon" → VoiceVoxBackendConfig("3")),
      Map("z" → CharacterConfig("z", "zundamon"))
    )

    val e = Context.empty

    import cats.implicits._
    (x |+| e) shouldEqual (e |+| x)
  } ← #29-39 it should "have identity element" in
```

モノイド則を満たすことのテストも書く

```scala
def fromNode(
    dialogueElem: scala.xml.Node,
    currentContext: Context = Context.empty
): Seq[(Say, Context)] = dialogueElem match {
  case Comment(_) ⇒ Seq.empty // コメントは無視する
  case Text(t) if t.forall(_.isWhitespace) ⇒ Seq.empty // 空行やただの
  case Text(t) ⇒ Seq(Say(t) → currentContext)
  case e: Elem ⇒
    e.child.flatMap(c ⇒ fromNode(c, currentContext |+| extract(e)))
} ← #84-90 : Seq[(Say, Context)] = dialogueElem match
```

😊 We can serialize XML tree using |+|

# モノイドの|+|を使って木を平らに潰す操作が書けた

# Done: flatten

XML原稿を平らに潰してメタデータを各要素に配った

# 2. Map

原稿を画像と音声に変換する

# Materialize `(Sentence, Context)`

- Image
  - Generate HTML using Twirl
    - Currently template is fixed
  - Render HTML using Chromium
    - Use screenshot option
  - Screenshot will be generated into filesystem
  - Use os-lib to spawn process
- Voice
  - Call API for VOICEVOX Engine
  - API returns `audio/wav` byte array
  - Use http4s to make HTTP requests

TwirlでHTMLを生成し、Chromiumでレンダーする
音声はVOICEVOXをhttp4sで叩く

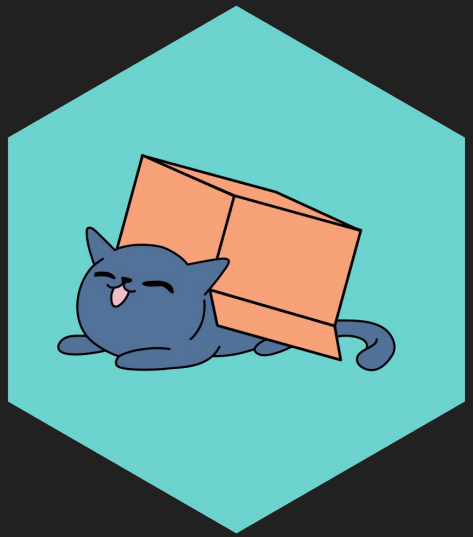
# Asynchronous

- `(Sentence, Context)` makes two operation (side effect)
  - Calling voice synthesizer
  - Calling HTML Renderer / Chromium
- Each is independent
  - ⇒ We can call them concurrently

生成処理は独立しているので非同期・並行して呼び出せる

# Handling asynchronous operation using Cats Effect

- Handle side effect as `IO[?]` type
- Separate definition and execution of side effect
- Programmer: Prepare and Compose `IO` type
- Cats Effect: Execute and schedule `IO` type
  - Fiber (managed thin thread)
  - Parallel execution

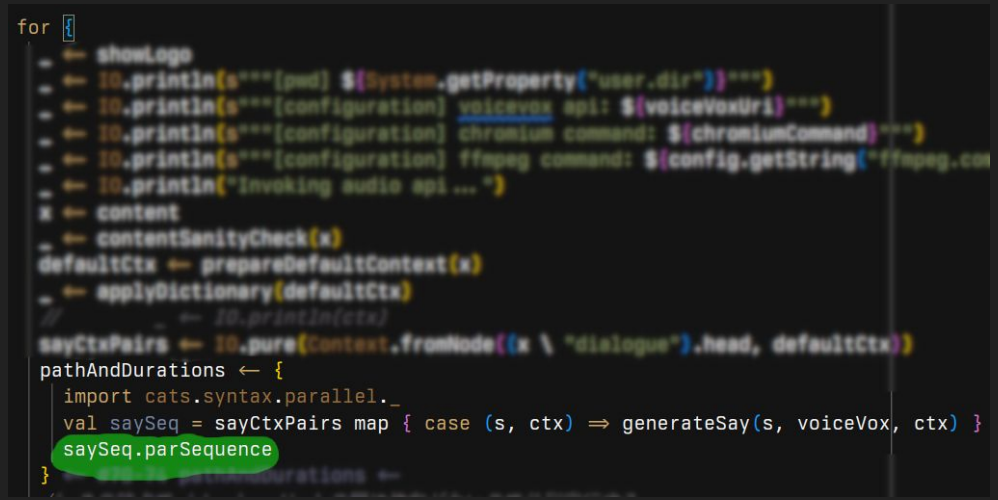


```
private def generateSay(
    sayElem: domain.model.Say,
    voiceVox: VoiceVox,
    ctx: Context
): IO[(fs2.io.file.Path, scala.concurrent.duration.FiniteDuration)] = for {
```

Cats Effectで非同期処理をこなしてもらうことにした

# Handling asynchronous operation using Cats Effect

- `saySeq: Seq[IO[(Path, FiniteDuration)]]`
- `saySeq.parSequence IO[Seq[(Path, FiniteDuration)]]`
  - Cats Effect does parallel execution




parSequenceメソッドで並列実行を指示できる

# 3. Combining

結合する

# Combining – call ffmpeg

- ffmpeg can create video from still image and sound
  - First, overlay all sounds with ffmpeg
    - Voice
    - BGM
  - Next, combine all rendered images into one MP4 file
  - Finally, zip sound and video
- Use `os-lib` to spawn ffmpeg process
- We’ve got MP4 file

最後に画像と音声をffmpegを呼び出して結合する

# Review: What we did

- Flattened XML tree, and made sequence of (speech text, context)
  - Context is merged by making it Monoid
- Mapped sequence of (speech text, context) into sequence of (image, voice)
  - Efficient parallel execution by using Cats Effect
- Combined sequence of (image, voice) into single MP4
  - FFmpeg does this

おさらい: XMLを潰して直線構造にし、画像と音声へ変換を行い、結合してMP4ファイルに変換した

# ZMM – philosophy

ZMMが目指す哲学

# Why I made this application? – License!

- Soft / Open license made VOICEVOX popular
  - MIT license for core library, soft license for each character
- Artists contribute their illustration on soft license
  - e.g. *“You can use it for anything as long as it is within the official terms and conditions.”*

音声合成・キャラクター・立ち絵は比較的オープンなライセンスで提供されていて、だからこそ広まった

# Why I made this application? – License!

- Famous video editing tools known among folks are on proprietary license
  - Free for using, but closed source
  - Once author dead, that's end
  - ZMM is provided on MIT license
- Open ecosystem is getting grow except for video editor
  - Why don't we?

動画作成ツールもオープンなライセンスで作ることで、
より動画コミュニティの世界を広げたい

# ZMM – Future

- Aiming at middle quality application
  - Not perfect, nor very powerful, but good for general use
- Main contributor: @windymelt
  - An user gave me `ci.yaml` (unique contributor now 😂)
- Documentation: WIP at https://www.3qe.us/zmm/doc/
- Web Frontend for folks: WIP at https://github.com/windymelt/zmf (private yet)
  - React + Typescript
  - Help me…

今のところ僕だけが開発しています。一緒にフロントエンドなども作りませんか

# “Connecting Dots”

- Connecting specialist and movie culture = ∞
- We can invent nice tool to connect them!

専門家と動画文化を接続することで、より良い文化へ

Let’s make video community
Bigger, Better, More Opened!

動画コミュニティをより大きく、良く、オープンに

# COPYING

- 「ゆっくり」の素材
- ずんだもん立ち絵素材
- VOICEROID2 結月ゆかり
- CeVIO AI
- VOICEVOX ずんだもん