前端代码
是怎样智能生成的

阿里双11会场模块 79.43% 代码智能生成的秘籍

为你揭秘前端智能化动态，切实提高前端研发效率

阿里巴巴集团 | 阿里云 开发者社区

关注imgcook官网

体验一键智能生成代码

扫码加入imgcook 钉钉答疑群

长按识别小程序码进入圈子

来自微信圈子

目录

序言篇

作为今年阿里经济体前端委员会的四大技术方向之一，前端智能化方向一被提及，就不免有人好奇：前端结合 AI 能做些什么，怎么做，未来会不会对前端产生很大的冲击等等。本篇文章将围绕这些问题，以「设计稿自动生成代码」场景为例，从背景分析、竞品分析、问题拆解、技术方案等几个角度切入，细述相关思考及过程实践。

背景分析

业界机器学习之势如火如荼，「AI 是未来的共识」频频出现在各大媒体上。李开复老师也在《AI · 未来》指出，将近 50%的人类工作将在 15 年内被人工智能所取代，尤其是简单的、重复性的工作。并且，白领比蓝领的工作更容易被取代；因为蓝领的工作可能还需要机器人和软硬件相关技术都突破才能被取代，而白领工作一般只需要软件技术突破就可以被取代。那我们前端这个“白领”工作会不会被取代，什么时候能被取代多少?

回看 2010 年，软件几乎吞噬了所有行业，带来近几年软件行业的繁荣；而到了 2019 年，软件开发行业本身却又在被 AI 所吞噬。你看：DBA 领域出现了 Question-to-SQL，针对某个领域只要问问题就可以生成 SQL 语句；基于机器学习的源码分析工具 TabNine 可以辅助代码生成；设计师行业也出了 P5Banner 智能设计师“鹿班”，测试领域的智能化结合也精彩纷呈。那前端领域呢? 那就不得不提一

个我们再熟悉不过的场景了，它就是**设计稿自动生成代码**（Design2Code，以下简称 D2C），阿里经济体前端委员会 – 前端智能化方向当前阶段就是聚焦在如何**让 AI 助力前端这个职能角色提效升级**，杜绝简单重复性工作，让前端工程师专注更有挑战性的工作内容！

竞品分析

2017 年，一篇关于图像转代码的 Pix2Code 论文掀起了业内激烈讨论的波澜，讲述如何从设计原型直接生成源代码。随后社区也不断涌现出基于此思想的类似 Screenshot2Code 的作品，2018 年微软 AILab 开源了草图转代码工具 Sketch2Code，同年年底，设计稿智能生成前端代码的新秀 Yotako 也初露锋芒，机器学习首次以不可小觑的姿态正式进入了前端开发者的视野。

	识别目标	控件数	结果现状	本质能力
Pix2Code	Bootstrap 基础控件	11	实验数据准确率接近 80%，支持控件少	中后台基础组件识别
Screenshot-2Code	HTML 基础标签	HTML 基础标签	准确率较低	端到端的图片生成代码
Sketch-2Code	手绘稿到控件	9	手绘草图降低了使用门槛	基于手绘稿的组件识别
shots	非智能识别，全人工标注	未知	有 Showcase，新设计稿无法试用，闭源，准确度未知，人工收集了 6000 个设计稿 Case	设计稿生成代码的布局算法能力，含各种场景下丰富的高质量的控件类型标注
Yotako	HTML 基础标签	未知	PSD 生成前端代码，有 Showcase，使用需要会员费，设计稿的要求非常高（几乎每个图层都有命名要求），刚起步	设计稿生成代码，目前规则的比重较大

基于上述竞品分析，我们能够得到以下几点启发：

1. 深度学习目前在图片方面的目标检测能力适合较大颗粒度的可复用的物料识别（模块识别、基础组件识别、业务组件识别）。
2. 完整的直接由图片生成代码的端到端模型复杂度高，生成的代码可用度不高，要达到所生成代码工业级可用，需要更细的分层拆解和多级子网络模型协同，短期可通过设计稿生成代码来做 D2C 体系建设。
3. 当模型的识别能力无法达到预期准确度时，可以借助设计稿人工的打底规则协议；一方面人工规则协议可以帮助用户强干预得到想要的结果，另一方面这些人工规则协议其实也是高质量的样本标注，可以当成训练样本优化模型识别准确度。

问题分解

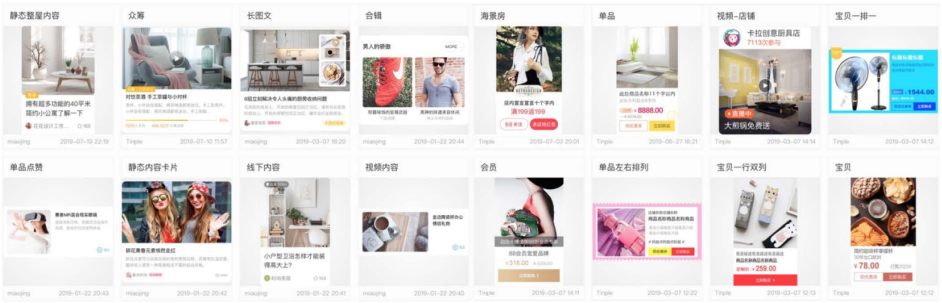
设计稿生成代码的目标是让 AI 助力前端这个职能角色提效升级，杜绝简单重复性工作内容。那我们先来分析下，“常规”前端尤其是面向 C 端业务的同学，一般的工作流程日常工作内容大致如下：

“常规”前端一般的开发工作量，主要集中在**视图代码**、**逻辑代码**和**数据联调**（甚至是数据接口开发，研发 Serveless 产品化时可期）这几大块，接下来，我们逐块拆解分析。

视图代码

视图代码研发，一般是根据视觉稿编写 HTML 和 CSS 代码。如何提效，当面

对 UI 视图开发重复性的工作时，自然想到组件化、模块化等封装复用物料的解决方案，基于此解决方案会有各种 UI 库的沉淀，甚至是可视化拼装搭建的更 HighLevel 的产品化封装，但复用的物料不能解决所有场景问题。个性化业务、个性化视图遍地开花，直面问题本身，直接生成可用的 HTML 和 CSS 代码是否可行?



这是业界一直在不断尝试的命题，通过设计工具的开发插件可以导出图层的基本信息，但这里的主要难点还是对设计稿的要求高、生成代码可维护性差，这是核心问题，我们来继续拆解。

设计稿要求高问题

对设计稿的要求高，会导致设计师的成本加大，相当于前端的工作量转嫁给了设计师，导致推广难度会非常大。一种可行的办法是采用 CV（ComputerVision，计算机视觉）结合导出图层信息的方式，以去除设计稿的约束，当然对设计稿的要求最好是直接导出一张图片，那样对设计师没有任何要求，也是我们梦寐以求的方案，我们也一直在尝试从静态图片中分离出各个适合的图层，但目前在生产环境可用度不够（小目标识别精准度问题、复杂背景提取问题仍待解决），毕竟设计稿自带的元信息，比一张图片提取处理的元信息要更多更精准。

可维护性问题

生成的代码结构一般都会面临可维护性方面的挑战：

- **合理布局嵌套**：包括绝对定位转相对定位、冗余节点删除、合理分组、循环判断等方面；
- **元素自适应**：元素本身扩展性、元素间对齐关系、元素最大宽高容错性；
- **语义化**：Classname 的多级语义化；
- **样式 CSS 表达**：背景色、圆角、线条等能用 CV 等方式分析提取样式，尽可能用 CSS 表达样式代替使用图片；
- ……

将这些问题拆解后，分门别类专项解决，解决起来看似没完没了，但好在目前发现的大类问题基本解决。很多人质疑道，这部分的能力的实现看起来和智能化一点关系都没有，顶多算个布局算法相关的专家规则系统。没错，现阶段这部分比较适合规则系统，对用户而言布局算法需要接近 100% 的可用度，另外这里涉及的大部分是无数属性值组合问题，当前用规则更可控。如果非要使用模型，那这个可被定义为多分类问题。同时，任何深度学习模型的应用都是需要先有清晰的问题定义过程，把问题规则定义清楚本就是必备过程。

但在规则解决起来麻烦的情况下，可以使用模型来辅助解决。比如**合理分组**（如下图）和**循环项**的判断，实践中我们发现，在各种情况下还是误判不断，算法规则难以枚举，这里需要跨元素间的上下文语义识别，这也是接下来正在用模型解决的重点问题。

（合理分组示意）

逻辑代码

逻辑代码开发主要包括数据绑定、动效、业务逻辑代码编写。可提效的部分，一般在于复用的动效和业务逻辑代码可沉淀基础组件、业务组件等。

- 接口字段绑定：从可行性上分析还是高的，根据视觉稿的文本或图片判断所属哪个候选字段，可以做，但性价比不高，因为接口数据字段绑定太业务，通用性不强。
- 动效：这部分的开发输入是交互稿，而且一般动效的交付形式各种各样参差不齐，有的是动画 gif 演示，有的是文字描述，有的是口述；动效代码的生成更适合用可视化的形式生成，直接智能生成没有参考依据，考虑投入产出比不在短期范围内。
- 业务逻辑：这部分开发的依据来源主要是 PRD，甚至是 PD 口述的逻辑；想要智能生成这部分逻辑代码，欠缺的输入太多，具体得看看智能化在这个子领域能解决什么问题。

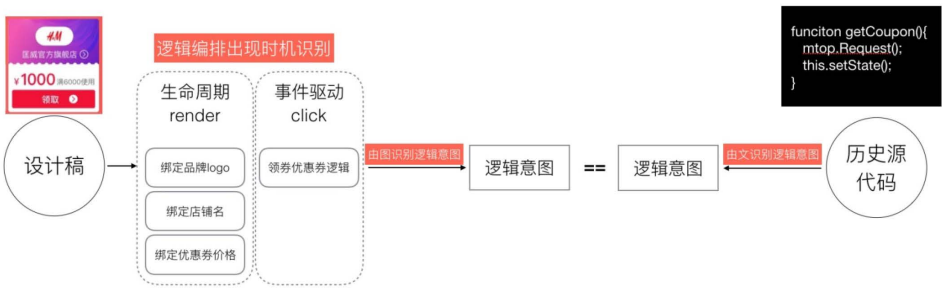
逻辑代码生成思考

理想方案当然是能像其他诗歌、绘画、音乐等艺术领域一样，学习历史数据，根据 PRD 文本输入，新逻辑代码能直接生成，但这样生成的代码能直接运行没有任何报错吗?

目前人工智能虽说发展迅速，但其能解决的问题还是有限，需要将问题定义成其擅长解决的问题类型。强化学习擅长策略优化问题，深度学习目前在计算机视觉方面擅长解决的是分类问题、目标检测问题，文本方面擅长 NLP（NaturalLanguage-Processing, 自然语言处理）等。

关于业务逻辑代码，首先想到的是对历史源代码的函数块利用 LSTM（Long-Short-TermMemory, 长短期记忆网络）和 NLP 等进行上下文分析，能得到代码函数块的语义，VSCode 智能代码提醒和 TabNine 都是类似的思路。

另外，在分析问题中（如下图）我们还发现，智能化在业务逻辑领域还可以协助识别逻辑点在视图出现的位置（时机），并根据视图猜测出的逻辑语义。



综上，我们总结一下智能化现阶段的可助力点：

- 由历史源代码分析猜测高频出现的函数块（逻辑块）的语义，实际得到的就是组件库或者基础函数块的语义，可在代码编辑时做代码块推荐等能力。
- 由视觉稿猜测部分可复用的逻辑点，如这里的字段绑定逻辑，可根据这里文本语义 NLP 猜测所属字段，部分图片元素根据有限范围内的图片分类，猜测所属对应的图片字段（如下图示例中的，氛围修饰图片还是 Logo 图片）；同时还可以识别可复用的业务逻辑组件，比如这里的领取优惠券逻辑。

所以，现阶段在业务逻辑生成中，可以解决的问题比较有限，尤其是新的业务逻辑点以新的逻辑编排出现时，这些参考依据都在 PD 的 PRD 或脑海中。所以针对业务逻辑的生成方案，现阶段的策略主要如下：

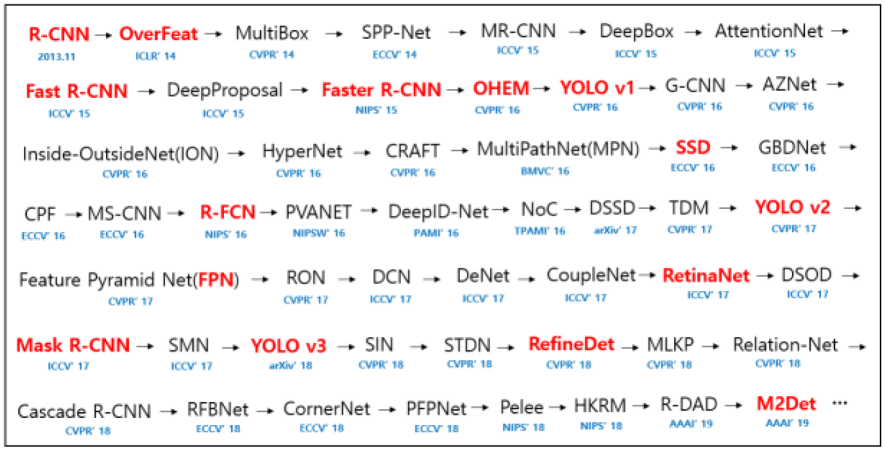
- 字段绑定：智能识别设计稿中的文本和图片语义分类，特别是文字部分。
- 可复用的业务逻辑点：根据视图智能识别，并由视图驱动自由组装，含小而美的逻辑点（一行表达式、或一般不足以封装成组件的数行代码）、基础组件、业

务组件。

- 无法复用的新业务逻辑：PRD 需求结构化（可视化）收集，这是一个高难度任务，还在尝试中。

小结

目前用智能化的方式自动生成 HTML+CSS+ 部分 JS+ 部分 DATA 的主要分析如上，是 DesigntoCode 的主要过程，内部项目名称叫做 D2C，后续系列文章会使用此简称，集团内外的落地产品名称为 imgcook。随着近几年主流设计工具（Sketch、PS、XD 等）的三方插件开发能力逐渐成熟，飞速发展的深度学习那甚至超过人类识别效果的趋势，这些都是 D2C 诞生和持续演进的强大后盾。



（目标检测 2014–2019 年 paper）

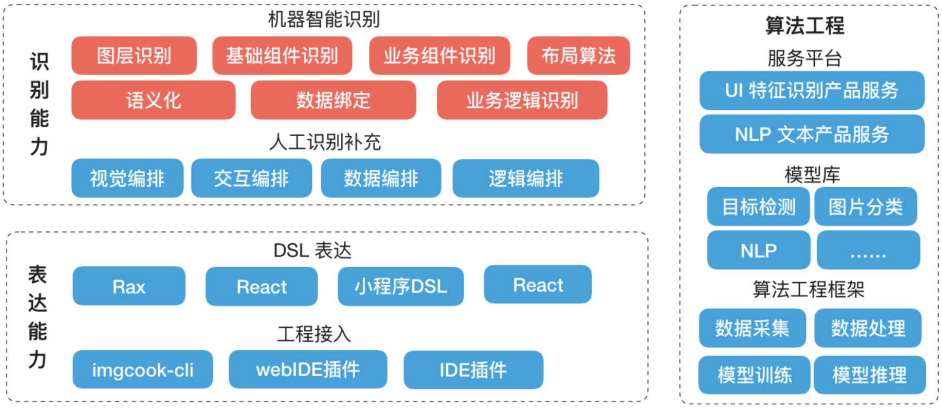
技术方案

基于上述前端智能化发展概括分析，我们对现有的 D2C 智能化技术体系做了能力概述分层，主要分为以下三部分：

- 识别能力：即对设计稿的识别能力。智能从设计稿分析出包含的图层、基础组件、业务组件、布局、语义化、数据字段、业务逻辑等多维度的信息。如果智

能识别不准，就可视化人工干预补充纠正，一方面是为了可视化低成本干预生成高可用代码，另一方面这些干预后的数据就是标注样本，反哺提升智能识别的准确率。

- 表达能力：主要做数据输出以及对工程部分接入
  - 通过 DSL 适配将标准的结构化描述做 Schema2Code
  - 通过 IDE 插件能力做工程接入
- 算法工程：为了更好的支撑 D2C 需要的智能化能力，将高频能力服务化，主要包含数据生成处理、模型服务部分
  - 样本生成：主要处理各渠道来源样本数据并生成样本
  - 模型服务：主要提供模型 API 封装服务以及数据回流



（前端智能化 D2C 能力概要分层）

在整个方案里，我们用同一套数据**协议规范（D2CSchema）**来连接各层的能力，确保其中的识别能够映射到具体对应的字段上，在表达阶段也能正确地通过出码引擎等方案生成代码。

智能识别技术分层

在整个 D2C 项目中，最核心的是上述识别能力部分的**机器智能识别**部分，这层的具体再分解如下，后续系列文章会围绕这些细分的分层展开内部实现原理。

- **物料识别层**：主要通过图像识别能力识别图像中的物料（模块识别、原子模块识别、基础组件识别、业务组件识别）。
- **图层处理层**：主要将设计稿或者图像中图层进行分离处理，并结合上一层的识别结果，整理好图层元信息。
- **图层再加工层**：对图层处理层的图层数据做进一步的规范化处理。
- **布局算法层**：转换二维中的绝对定位图层布局为相对定位和 Flex 布局。
- **语义化层**：通过图层的多维特征对图层在生成代码端做语义化表达。
- **字段绑定层**：对图层里的静态数据结合数据接口做接口动态数据字段绑定映射。
- **业务逻辑层**：对已配置的业务逻辑通过业务逻辑识别和表达器来生成业务逻辑代码协议。
- **出码引擎层**：最后输出经过各层智能化处理好的代码协议，经过表达能力（协议转代码的引擎）输出各种 DSL 代码。

（D2C 识别能力技术分层）

技术痛点

当然，这其中的**识别不全面**、**识别准确度不高**一直是 D2C 老生常谈的话题，也是我们的核心技术痛点。我们尝试从这几个角度来分析引起这个问题的因素：

1. **识别问题定义不准确**：问题定义不准确是影响模型识别不准的首要因素，很多人认为样本和模型是主要因素，但在这之前，可能一开始的对问题的定义就出现了问题，我们需要判断我们的识别诉求模型是否合适做，如果合适那该怎么定义清楚这里面的规则等。
2. **高质量的数据集样本缺乏**：我们在识别层的各个机器智能识别能力需要依赖不同的样本，那我们的样本能覆盖多少前端开发场景，各个场景的样本数据质量怎么样，数据标准是否统一，特征工程处理是否统一，样本是否存在二义性，互通性如何，这是我们当下所面临的问题。
3. **模型召回低、存在误判**：我们往往会在样本里堆积许多不同场景下不同种类的样本作为训练，期望通过一个模型来解决所有的识别问题，但这却往往会让模型的部分分类召回率低，对于一些有二义性的分类也会存在误判。

问题定义

深度学习里的计算机视觉类模型目前比较适合解决的是分类问题和目标检测问题，我们判断一个识别问题是否该使用深度模型的前提是——我们自己是否能够判断和理解，这类问题是否有二义性等，如果人也无法准确地判断，那么这个识别问题可能就不太合适。

假如判断适合用深度学习的分类来做，那就需要继续定义清楚所有的分类，这些分类之间需要严谨、有排他性、可完整枚举。比如在做图片的语义化这个命题的时候，一般图片的 ClassName 通用常规命名有哪些，举个例子，其分析过程如下：

- 第一步：尽可能多地找出相关的设计稿，一个个枚举里面的图片类型。
- 第二步：对图片类型进行合理归纳归类，这是最容易有争论的地方，定义不好有二义性会导致最有模型准确度问题。
- 第三步：分析每类图片的特征，这些特征是否典型，是否是最核心的特征点，因为关系到后续模型的推理泛化能力。
- 第四步：每类图片的数据样本来源是否有，没有的话能否自动造；如果数据样本无法有，那就不适合用模型，可以改用算法规则等方式替代先看效果。

D2C 项目中很多这样的问题，问题本身的定义需要非常精准，并且需要有科学的参考依据，这一点本身就比较难，因为没有可借鉴的先例，只能先用已知的经验先试用，用户测试有问题后再修正，这是一个需要持续迭代持续完善的痛点。

样本质量

针对样本问题，我们需要对这部分数据集建立标准规范，分场景构建多维度的数据集，将收集的数据做统一的处理和提供，并以此期望能建立一套规范的数据体系。

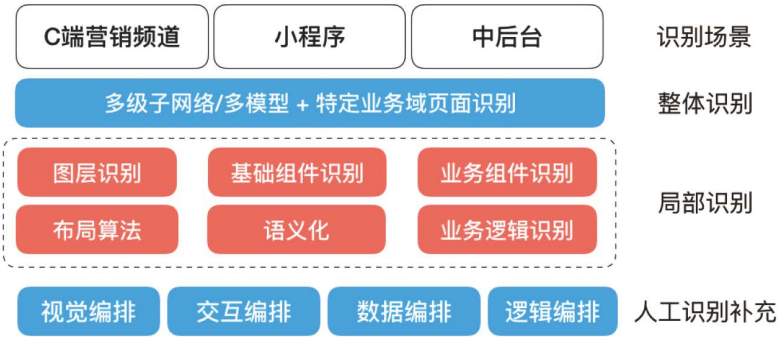
在这套体系中，我们使用统一的样本数据存储格式，提供统一的针对不同问题（分类、目标检测）的样本评估工具来评估各项数据集的质量，针对部分特定模型可采取效果较好的特征工程（归一化、边缘放大等）来处理，同类问题的样本也期望能在后续不同的模型里能够流通作比较，来评估不同模型的准确度和效率。

（数据样本工程体系）

模型

针对模型的召回和误判问题，我们尝试**收敛场景来提高准确度**。往往不同场景下的样本会存在一些特征上

的相似点或者影响部分分类局部关键特征点，导致产生误判、召回低，我们期望能够通过收敛场景来做模式化的识别，提高模型准确度。我们把场景收敛到如下三个场景：无线 C 端营销频道场景、小程序场景以及 PC 中后台场景。这里面几个场景的设计模式都有自己的特色，针对各个场景来设计垂直的识别模型可以高效地提高单一场景的识别准确度。



（D2C 场景）

过程思考

既然使用了深度模型，有一个比较现实的问题是，模型的泛化能力不够，识别的准确率必然不能 100% 让用户满意，除了能够在后续不断地补充这批识别不到的样本数据外，在这之前我们能做什么？

在 D2C 的整个还原链路里，对于识别模型，我们还遵守一个方法论，即设计一套协议或者规则能通过其他方式来兜底深度模型的识别效果，保障在模型识别不准确的情况下用户仍可完成诉求：**手动约定 > 规则策略 > 机器学习 > 深度模型**。举个例子比如需要识别设计稿里的一个循环体：

- 初期，我们可以在设计稿里手动约定循环体的协议来达成
- 根据图层的上下文信息可做一些规则的判断，来判断是否是循环体
- 利用机器学习的图层特征，可尝试做规则策略的更上游的策略优化
- 生成循环体的一些正负样本来通过深度模型进行学习

其中手动约定的设计稿协议解析优先级最高，这样也能确保后续的流程不受阻塞和错误识别的干扰。

业务落地

2019 双十一落地

在今年的双十一场景中，我们的 D2C 覆盖了天猫淘宝会场的新增模块，包括主会场、行业会场、营销玩法会场、榜单会场等，包含了视图代码和部分逻辑代码的自动生成，在可统计范围内，**D2C 代码生成占据了 79.34%**。目前代码的人工改动的主要原因有：全新业务逻辑、动画、字段绑定猜测错误（字段标准化情况不佳）、循环猜测错误、样式自适应修改等，这些问题也都是接下来需要逐步完善的。

（D2C 代码生成用户更改情况）

整体落地情况

我们对外的产品 imgcook，截止 2019.11.09 日，相关的使用数据情况如下：

- 模块数 12681 个，周新增 540 个左右；
- 用户数 4315 个，周新增 150 个左右；
- 自定义 DSL：总数 109 个；

目前可提供的服务能力如下：

- 设计稿全链路还原：通过 Sketch、PhotoShop 插件一键导出设计稿图层，生成自定义的 DSL 代码。
- 图像链路还原：支持用户自定义上传图片、文件或填写图片链接，直接还原生成自定义的 DSL 代码。

后续规划

- 继续降低设计稿要求，争取设计稿 0 协议，其中分组、循环的智能识别准确度提升，减少视觉稿协议的人工干预成本。
- 组件识别准确度提升，目前只有 72% 的准确度，业务应用可用率低。

- 页面级和项目级还原能力可用率提升，依赖页面分割能力的准确度提升。
- 小程序、中后台等领域的页面级还原能力提升，含复杂表单、表格、图表的整体还原能力。
- 静态图片生成代码链路可用度提升，真正可用于生产环境。
- 算法工程产品完善，样本生成渠道更多元，样本集更丰富。
- D2C 能力**开源**。

未来我们希望能通过前端智能化 D2C 项目，让前端智能化技术方案普惠化，沉淀更具竞争力的样本和模型，提供准确度更高、可用度更高的代码智能生成服务；切实提高前端研发效率，减少简单的重复性工作，不加班少加班，一起专注更有挑战性的工作内容！

智能插件篇

作为阿里经济体前端委员会四大技术方向之一，前端智能化项目经历了 2019 双十一的阶段性考验，交出了不错的答卷，天猫淘宝双十一会场新增模块 79.34% 的线上代码由前端智能化项目自动生成。在此期间研发小组经历了许多困难与思考，本次《前端代码是怎样智能生成的》系列分享，将与大家分享前端智能化项目中技术与思考的点点滴滴。

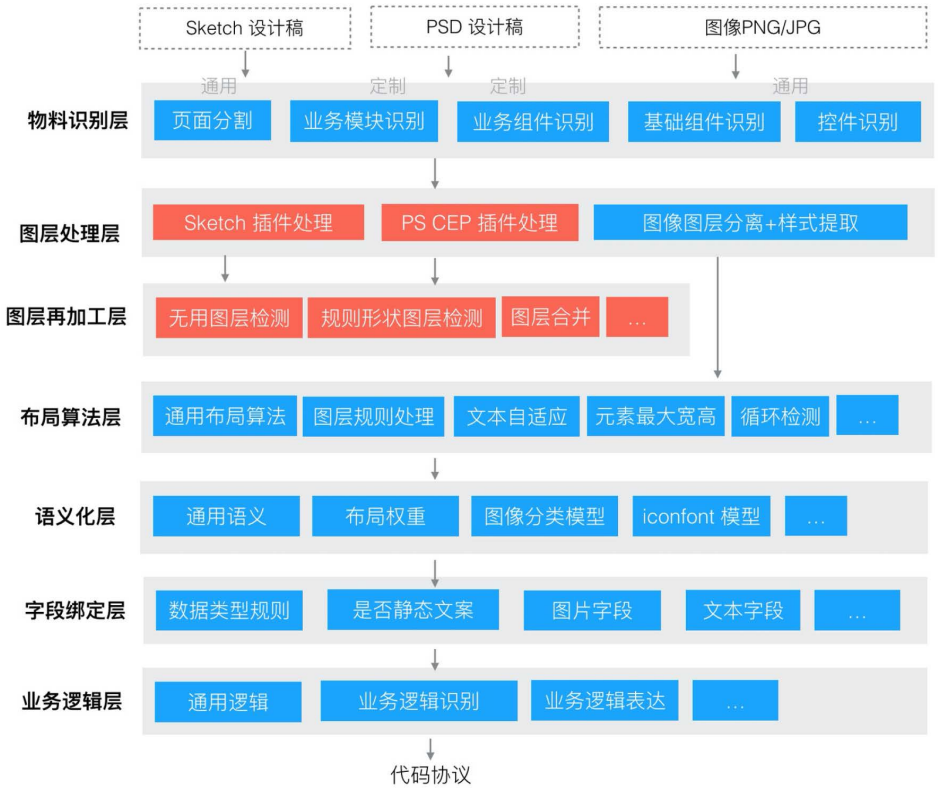
概述

在一个常见的开发周期中往往遵循着产品需求到交互稿到设计稿再到前端开发的过程。所以在 Design2Code（简称 D2C）项目过程中，设计师负责来设计产品视觉效果和产出视觉设计稿，而前端开发工程师以设计稿为输入进行开发。所以同样地，在前端智能化的过程中，我们需要一种能自动解析设计稿信息的能力来替代传统的人工分析和抠图等繁琐的工作。同时，随着近几年主流设计工具（Sketch、PS、XD 等）的三方插件开发能力逐渐成熟，借助官方提供的 API 能够比较好得还原一些基本的结构化信息和样式信息，从而完成对设计稿原始信息的提取。在此篇文章中，我们将以前端智能化的落地产品 imgcook 的 Sketch 插件为例子，详细介绍我们是如何通过插件对设计稿做处理，最终导出以绝对布局为基础的元素信息，供下游布局算法使用。

所在分层

如图所示，链路中的第一层为物料识别层，设计稿将作为这一层的输入。这一层主要是用来识别页面和模块中包含的物料的，比如对基础组件，业务组件和基础控件的识别，从而辅助进行页面分割并对下游输入关于某一元素的相关信息。随后，设计稿的原始信息（文件）将会进入到图像处理层，这一层主要是通过这篇文章所介绍

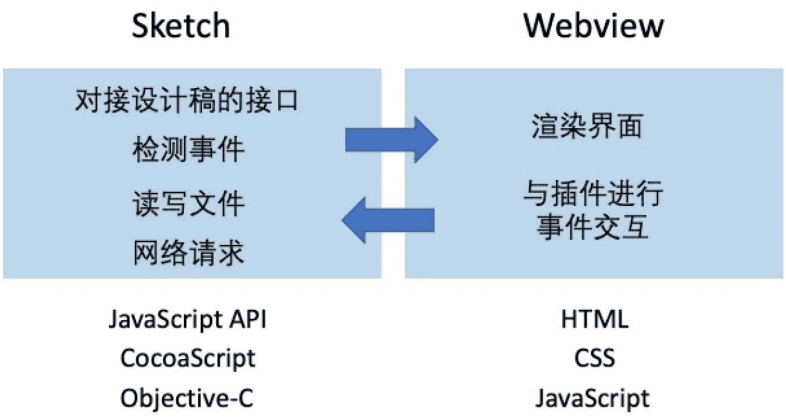
的插件实现，因为插件可以注入到 Sketch 或者 Photoshop 等设计工具中从而借助官方提供的一些能力完成对原始设计稿信息的提取。这一层提取的结果将是一个符合 imgcook 规范的 JSON 结构的数据，内容主要是提取出所有元素的相关信息，包含元素的绝对位置和 CSS 可表达的属性，最终可以理解以绝对定位为基础的模块和页面。同时，由于不同设计师可能遵循一些不同于传统前端开发的规范来组织设计稿的图层和结构，在图像再加工层中我们还将借助计算机视觉和智能化的能力对原始设计稿中出现的图层再加工。例如过滤掉一些无用的图层或是合并一些可以当作一个整体的图层等。



(D2C 技术能力分层)

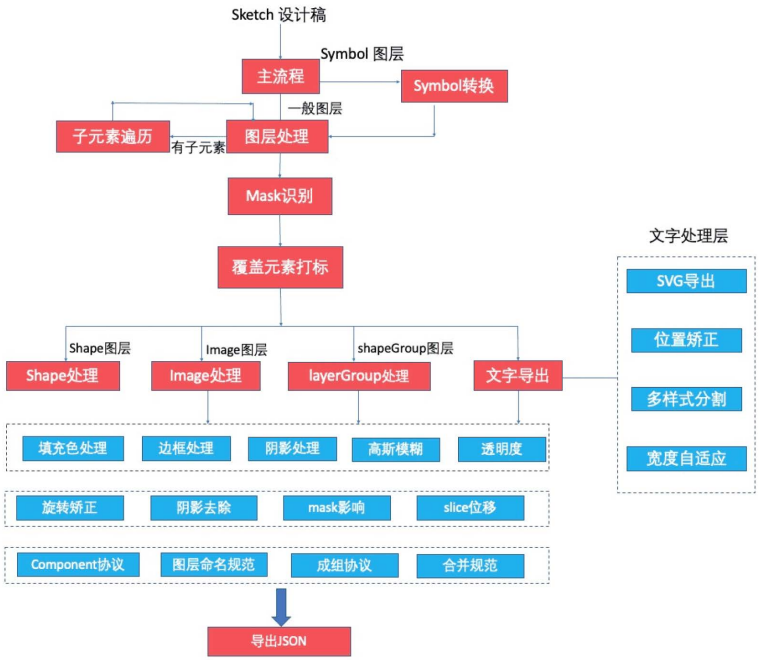
技术选型

如图所示，在对接原始设计稿和获取原始设计稿信息的过程中，我们主要是使用了 Sketch 官方提供的 JSAPI 进行开发，对于一些官方没有包装 JS 接口的功能，我们借助于 CocoaScript 对原始 Objective-C 接口进行调用。同时，我们使用了 Webview 技术可以使用前端技术栈来渲染插件界面。我们采用 SkpmSketch 的插件管理工具的脚手架能力和插件发布能力。



（插件技术选型示意图）

插件图层处理



（插件图层处理流程）

总体流程

如结构图所示，imgcookSketch 插件将读取设计稿，按 Depth-firstSearch(DFS) 的方式循环遍历所有类型的图层，提取图层的基本信息，包括位置和大小。值得注意的是由于 Sketch 里 Symbol 的概念相当于它的 Symbolmaster 的子类，可以覆盖它的 Symbolmaster 的部分属性，所以对于 Symbol 类型的图层，我们要找到它的 Symbolmaster，提取相关信息。之后我们会对所有会被蒙层影响的或者被其他图层覆盖的元素打标，因为这两者会影响到当前图层的视觉输出。之后因为各个类型具体的所拥有的样式不相同，我们会对 Shape,Image,Text 和其他图层分别做处理，把相关的 Sketch 属性转化为 CSS 理解的形式。我们对设计师约定了一些设计协议，可以通过在设计稿中不同的命名给图层指定为成组或者组件，同时带出相关组件信

息。下面我们挑取这个过程中两个值得注意的难点进行分别讲解。

蒙层处理

在 Sketch 里蒙层的作用是相当于一个底板，所有在结构上位于其之上的图层如果区域超出了蒙层，这部分超出的区域就会被截断。对于蒙层的处理主要有以下几个不同于正常图层的点：

- 由于在 HTML 和 CSS 领域并没有直接对应的概念，蒙层并不能直接导出相关 CSS 属性或者 HTML 属性
- Mask 图层不但会影响自己本身的处理，也会影响其他图层的视觉，所以遇到 Mask 需要多图层一起处理
- 由于 Mask 的形状可能是不规则形状，所以需要考虑如何判断合理的区域进行截断

针对以上难点和精准还原视觉的目的，我们开发了一套算法计算 Mask 和其影响的所有图层的区域计算和形状计算，针对区域规则且形状规则的做 CSS 属性上的常规截取处理，对于无法使用 CSS 属性表示的情况对 Sketch 视觉可见的区域进行截图处理。其中，我们会进行无用图层检测，如果一个图层在它的 Mask 区域之外，则此图层将被视为无用图层被删除，同时，如果一个图层完全在它的 Mask 区域内，则此图层不会被 Mask 影响，按照原有逻辑处理。

智能文字位置校准

相对于其他诸如 Shape 和 Image 图层的处理，文字图层会更复杂一点，原因主要有：

- 对于 Shape 和 Image 的每一个图层，我们往往也只需要对应导出一个节点，这个节点包括位置和样式等属性，但是对于文字图层，如果包含多样式，比如颜色，字号，行高等不同，则需要将一个文字图层拆分为多个节点导出

- Sketch 有定宽类型的文本框，但是对于 HTML 中 span 等标签为行内元素，没有宽度等信息，所以需要对 Sketch 中的多行文本做拆分
- 目前 Sketch 中文字图层想要得到位置和样式，需要依赖导出的 SVG 信息，而 SketchSVGExporter 接口导出的 SVG 信息经常出现位置不准的情况

针对第一个问题，我们对 SVG 信息进行循环检测，判断每个 SVG 子节点是否有 CSS 相关的属性发生了变化，如果有变化，就会新建立一个子节点存储相关信息。针对第二个问题，我们会对定宽的文本框导出准确的宽度信息还有行数信息，布局算法会针对此信息作出正确的判断。重点来讲一下第三个问题，由于 svg 信息在对于富文本的情况下会不准确的情况，我们设计了一套基于计算机视觉的算法会对文本框的基线进行矫正，整体流程如下：

- 对文字图层进行检测是否存在富文本文本框
- 由于在 Sketch 里单个文本框不同样式的文本基线一定在一条水平线上，我们的校准目标便是基于此。首先我们会对当前文本框做截图处理
- 借助 OpenCV 库分析截图
- 使用 Canny 边缘检测，分析出字体轮廓，确定基线位置
- 计算不同样式的文本之间的基线位置差
- 传回插件关于位置差的信息，插件对位置以最大字体为基准进行矫正

图层再加工

智能图层合并

设计稿的图层和前端开发之间还有一个差异就是图层合并的问题。往往设计师关注的焦点是能否在设计稿中实现想要的视觉效果，而不会像前端工程师一样关注元素结构和嵌套的合理性。所以有时候在设计稿中，设计师为了实现一个诸如 icon 或者氛围图等视觉效果时，会使用若干的小图层拼接起来，但是从结构角度来讲，这个拼接起来的图形应该是一个整体，在这种情况下我们就需要合并图层（将若干个图层作为一个图层，截图导出）。我们实现了一套自动合并图层的算法，算法会自动检测一

个组下是否有若干个应该被合并的图层，然后自动在导出的过程中合并，以便后续链路可以处理结构。

无用图层检测

在设计稿中，设计师有时会添加一些对最终布局和视觉没有影响的图层，为了结构的合理性和精简性，我们需要对这部分图层进行筛除。如下的情况下图层会被处理：

- 如果有两个 Image 图层视觉是一样的但是引用的 url 不同，则统一到相同的 url
- 如果图层没有 backgroundImage,backgroundColor 和 borderWidth 属性，则图层没有视觉效果，直接过滤掉
- 如果有无某图层没有对图像矩阵产生影响，则过滤掉此图层
- 如果图层被非透明图层覆盖，则过滤掉此图层
- 如果有透明区域小于某个阈值的图片，则过滤掉此图层

插件测试和度量

单元测试体系

由于 Sketch 插件是 imgcook 智能生成代码体系的上游，在每一次代码更改和发布之前，需要对插件做严格的测试以确保功能可用，所以我们使用 Skpm-Test，一个 Skpm 体系下的类 Jest 测试框架建立了单元测试体系，覆盖率达到 95% 左右。

绝对定位布局查看

如之前讲到的，Sketch 插件导出的信息是包含每个子节点的绝对定位的位置和相关的 CSS 属性，每个节点的属性和类型和 HTML 一一映射，所以我们可以将导出 JSON 直接转化为 HTML+CSS 查看导出效果，使用者可以直接通过 https://www.imgcook.com/design-restore 粘贴导出的 JSON 查看效果。

视觉还原度量体系

（插件视觉度量流程）

我们借助计算机视觉处理库 OpenCV 开发了一套算法用于衡量导出的 JSON 数据是否完全还原了原设计稿的视觉效果。

OpenCV 计算视觉还原分数主要分为以下几个步骤

- 图层预处理
  - 如果原始图像和导出图像大小不一致，重置到相同大小
  - 将图像转成灰白图
- 图层对比主逻辑
  - 分析导出的 JSON 信息，对 JSON 里每个子元素进行如下操作：
  - 获取子元素的宽高和位置 :x，y，w，h
  - 记录此元素的内部有多少子元素和横跨的元素
  - 对于每个子元素，取出原设计稿图相对应的区域
  - 模版匹配：使用 OpenCV 模版匹配找出此区域在总还原图中的位置和相似度
  - 如果有元素跨越了此元素并且那个元素已经被处理过，则忽略此元素
  - 如果此元素有子元素并且子元素已经被处理过，则忽略此元素
- 处理完成导出 JSON
- JSON 元素集合的一个数组，对于每个数组中的元素有如下属性
  - originPosition: 原始设计稿此元素的位置

- exportPosition: 导出图层此元素的位置
- similarity: 导出图层和原始设计稿相似度
- width: 原始宽度
- height: 原始高度

可以看出，我们通过计算机视觉已经分析出了每个图层的原始位置和还原后的位置，同时度量了每个图层的相似度，综合的度量分数应该综合考虑以下三个指标：

- 总图层数量
- 还原图层相似度
- 还原图层的位置

从这三个指标出发，我们设计如下公式计算还原度：

$$P = 1 - \frac{1}{2n}\sum_{i=1}^{n}((1-c_i) + (\frac{\delta x_i}{x} + \frac{\delta y_i}{y}))$$

其中 P 表示 restore 分数，n 表示图层的总数量，c_i 表示第 i 个还原图层和第 i 个原始图层的相似度，δx_i 表示第 i 个还原图层和原始图层 x 方向位移差，x 表示总宽度，δy_i 表示第 i 个还原图层和原始图层 y 方向位移差，y 表示总高度。使用此公式，如果全部图层都完全没有被还原 (相似度为 0，x 位移为宽度，y 位移为高度)，则 P=0，如果全部图层都完美还原 (相似度为 1，x 位移为 0，y 位移为 0)，则 P=1，所以我们可以较为精准的度量视觉还原程度。

未来展望

去规范继续升级

目前我们出了一些设计协议要求设计师按照一定的规范来制作设计稿，以便可以达到更好的还原效果；对设计稿的约束曾经高达 20+ 条，我们通过智能图层加工层去掉了大部分规范，目前主要剩下 3 条约规范束，接下来，我们将进一步通过智能化的手段逐步的去掉这些对设计师和前端的约束，达到 0 约束还原。

还原能力升级

我们将在未来的 Sketch 版本中继续提高 Sketch 插件的视觉还原度，目前阶段根据度量体系 Sketch 还原的能力平均在 95% 左右，我们将在之后的版本中继续提高这个能力。

还原效率升级

目前由于在插件中设计到大量的极速过程，导致导出速度有的时候不尽理想，我们也对这个问题进行了一定的调研，发现目前插件的确是在处理多图层，尤其是包含多张图片上传的场景速度比较慢，未来我们也会对还原速度进行一次大幅度的升级。

在科技飞速发展的今天，前端智能化的浪潮已经到来，未来一些简单的、重复性、规律性强的开发一定会逐渐地被机器取代，在这样的过程中，机器对设计稿的理解也一定会更上一个台阶。我们也会保证插件在未来达到更高的智能化水平，从而准确地理解设计师的意图从而更好的为前端服务！

图像分离篇

作为阿里经济体前端委员会四大技术方向之一，前端智能化项目经历了 2019 双十一的阶段性考验，交出了不错的答卷，天猫淘宝双十一会场新增模块 79.34% 的线上代码由前端智能化项目自动生成。在此期间研发小组经历了许多困难与思考，本次《前端代码是怎样智能生成的》系列分享，将与大家分享前端智能化项目中技术与思考的点点滴滴。

概述

一直以来，如何从“视觉稿”精确的还原出对应的 UI 侧代码一直是端侧开发同学工作里消耗比较大的部分，一方面这部分的工作比较确定缺少技术深度，另一方面视觉设计师也需要投入大量的走查时间，有大量无谓的沟通和消耗。

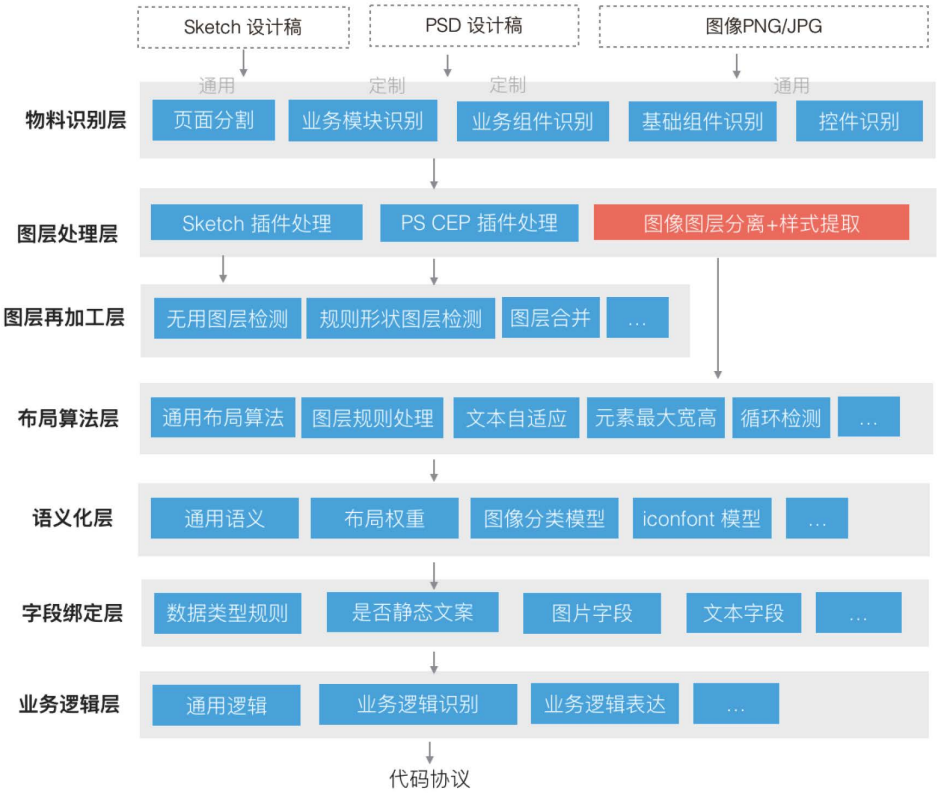
很多人会比较好奇，为什么我们尝试使用图片做为输入源，一方面基于 Sketch 或者 Photoshop 等插件相对容易拿到确定性的信息，图片在某些方面容易丢失一些特征；另外基于图片的分析其实挑战更大。我们做这个选择有以下原因：

1. 首先图片作为最终的产出物，更直观和确定性，另外这个链路里对上游不会有约束性。
2. 视觉稿跟开发代码不一样的地方在于布局的不同，比如 listview，grdview 这类布局类别在设计稿中是不存在的
3. 基于图片的应用场景会更普适，类似场景。例如自动化测试能力的支持，基于竞品直接截图来套用我们自己的数据源找体感，这类场景是其他的方案做不到的。
4. 设计稿有图层堆叠的问题，从图片出发可以更好地合并图层。

图像分离是 D2C 图像处理层的重要组成部分，具体内容包括了版面分析，复杂背景处理，布局识别和属性提取。本篇会从版面分析和复杂背景内容提取两个部分加以介绍。版面分析会将图像分割成若干个区块，并对不同的内容做分割和合并。复杂背景处理会在版面分析的基础上提取一些叠加的元素。

所在分层

本文讲述前端智能化 D2C 里技术分层中的**图层处理**能力层，主要负责识别图像的元素类别和提取样式，同时也为后续的布局算法层赋能。



版面分析

版面分析主要处理 UI 图像的前景提取和背景分析，通过前后景分离算法，将 UI 视觉稿剪裁为 GUI 元素：

1. 背景分析：通过机器视觉算法，分析背景颜色，背景渐变方向，以及背景的连通区域。
2. 前景分析：通过深度学习算法，对 GUI 碎片进行整理，合并，识别。

背景分析

背景分析的关键在于找到背景的连通区域和闭合区间，具体的步骤如下：

- 第一步：判断背景区块，通过 sobel，Lapacian，canny 等边缘检测的方法计算出梯度变化方向，从而得到纯色背景和渐变色背景区域。基于拉普拉斯算子的背景区域提取离散拉普拉斯算子的模板如下：

$$\nabla^2 f(x,y) = [f(x+1,y) - f(x,y)] - [f(x,y) - f(x-1,y)] + [f(x,y+1) - f(x,y)] - [f(x,y) - f(x,y-1)]$$
$$= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

此外，我们通过统计背景运动趋势的方式来判定它是否存在渐变色背景。如果存在，我们将通过第二步进行精细化处理。

- 第二步：基于漫水填充算法，选取漫水的种子节点，滤除渐变色背景区域噪声。

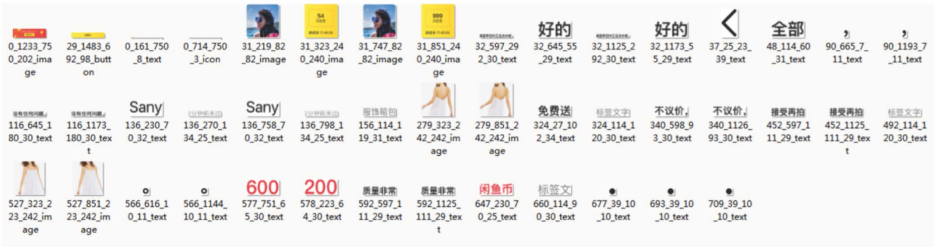
```
deffill_color_diffuse_water_from_img(task_out_dir,image,x,y,thres_
"""
漫水填充：会改变图像
"""
#  获取图片的高和宽
h,w=image.shape[:2]

#  创建一个 h+2,w+2 的遮罩层，
#  这里需要注意，OpenCV 的默认规定，
#  遮罩层的 shape 必须是 h+2，w+2 并且必须是单通道 8 位，具体原因我也不是很清楚。
mask=np.zeros([h+2,w+2],np.uint8)

#  这里执行漫水填充，参数代表：
#  copyImg：要填充的图片
```

```
#  mask：遮罩层
#  (x,y)：开始填充的位置（开始的种子点）
#  (255,255,255)：填充的值，这里填充成白色
#  (100,100,100)：开始的种子点与整个图像的像素值的最大的负差值
#  (50,50,50)：开始的种子点与整个图像的像素值的最大的正差值
#  cv.FLOODFILL_FIXED_RANGE：处理图像的方法，一般处理彩色图象用这个方法
cv2.floodFill(image,mask,(x,y),fill_color,thres_down,thres_up
cv2.imwrite(task_out_dir+"/ui/tmp2.png",image)
#  mask 是非常重要的一个区域，这块区域内会显示哪些区域被填充了颜色
#  对于 UI 自动化，mask 可以设置成 shape，大小以 1 最大的宽和高为准
returnimage,mask
```

处理过后的效果如下：

（原图与处理效果图）

- 第三步：通过版面切割，提取 GUI 元素。

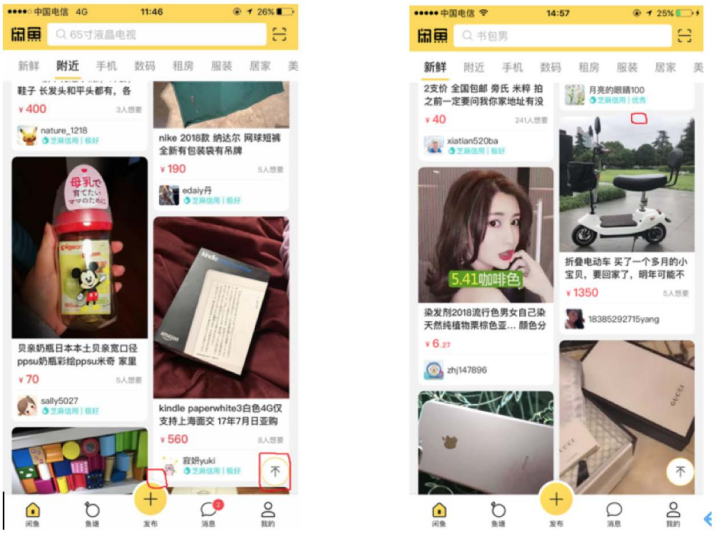


（背景分析后提取的内容模块）

这个时候我们已经成功将图片分层并提取了模块，后续细节可以在前景分析和复杂背景提取中获得。

前景分析

前景分析的关键在于组件完整性切割与识别。我们通过连通域分析，防止组件碎片化，再通过机器学习识别组件类型，再通过组件类型进行碎片化合并，反复执行上述动作，直到无特征属性碎片。我们通过瀑布流中提取一个完整 item 为例：



a) b)

（标红部分是处理难点）

闲鱼页面中瀑布流卡片识别是实现布局分析的一个重要步骤，需求是当卡片完整出现在截屏图像中时 (允许图标遮挡) 需要识别出来，卡片被背景部分遮挡时不应该识别出来。上图的瀑布流卡片样式，由于其布局紧凑且样式繁多，导致容易产生漏检或误检。

基于边缘梯度或连通域的传统图像处理方法能根据图像本身的灰度或者形状特征提取出瀑布流卡片的轮廓，优点是定位精度高、运算速度快。缺点是容易受到干扰，召回率不高。

基于目标检测或者特征点检测的深度学习方法采用有监督的方式学习卡片的样式特征，优点是不易受到干扰，召回率很高。缺点是因为涉及回归过程，定位精度比传统图像处理方法要低，并且需要大量的人工标注，运算速度也比传统图像处理方法要慢。

受集成学习的启发，通过融合传统图像处理方法和深度学习方法，结合两者各自的优点，最终能得到具有较高精确率、召回率和定位精度的识别结果。

传统图像处理算法流程如下图所示 :

1》输入的瀑布流卡片图像转换成灰度图后使用对比度受限的自适应直方图均衡化 (CLAHE) 进行增强

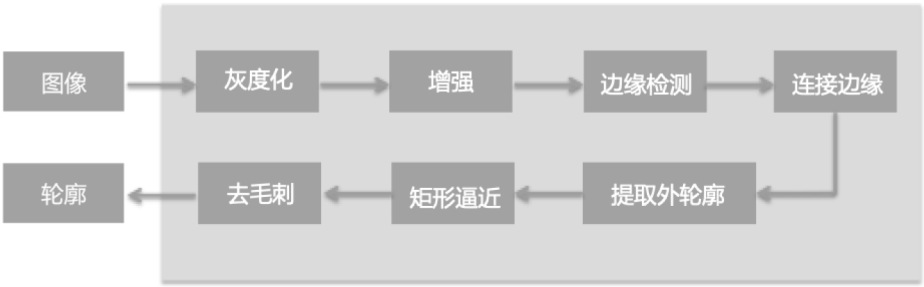
2》使用 Canny 算子进行边缘检测得到二值化图像

3》对二值化图像进行形态学膨胀处理，连接断开的边缘

4》提取连续边缘的外层轮廓，并基于轮廓包含区域的面积大小丢弃面积较小的轮廓，输出候选轮廓

5》使用 Douglas-Peucker 算法进行矩形逼近，保留最像矩形的外轮廓，输出新的候选轮廓

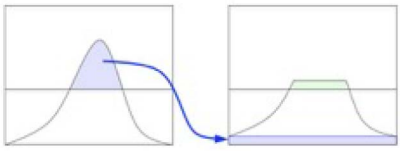
6》最后对第 4 步的候选轮廓进行水平和垂直方向投影得到平滑的轮廓作为输出



（传统图像处理算法流程）

算法流程中 1》-3》可以归为边缘检测部分。

受各种因素影响，图像会出现降质，需要对其进行增强来提高边缘检测的效果。使用全图单一的直方图进行均衡化显然不是最好的选择，因为截取的瀑布流图像不同区域对比度可能差别很大，增强后的图像可能会产生伪影。在单一直方图均衡化的基础上，学者基于分块处理的思想提出了自适应的直方图均衡化算法 (AHE)，但是 AHE 在增强边缘的同时有时候也会将噪声放大。后来学者在 AHE 的基础上提出了 CLAHE，利用一个对比度阈值来去除噪声的干扰，如下图所示直方图，CLAHE 不是将直方图中超过阈值的部分丢弃，而是将其均匀分布于其他 bin 中。



（直方图均衡）

Canny 算子是一种经典的边缘检测算子，它能得到精确的边缘位置。Canny 检测的一般步骤为：1) 用高斯滤波进行降噪 2) 用一阶偏导的有限差分计算梯度的幅值和方向 3) 对梯度幅值进行非极大值抑制 4) 用双阈值检测和连接边缘。实验过程中，需要多次尝试选择较好的双阈值参数。

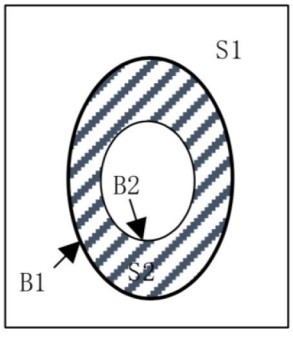
检测出来的边缘在某些局部地方会断开，可以采用特定形状和尺寸的结构元素对

二值化图像进行形态学膨胀处理来连接断开的边缘。边缘检测的结果如下图所示，其中 c) 中 CLAHE 设定对比度阈值为 10.0，区域大小为 (10，10)，d) 中 Canny 检测设置双阈值为 (20，80)，e) 中形态学膨胀结构元素使用大小为 (3，3) 的十字线。

a) b) c) d) e)

（传统图像处理结果图）

算法流程中 4》-6》可以归为轮廓提取部分。二值图像形态学膨胀处理后，首先提取连续边缘的外层轮廓。如下图所示，对于只有 0 和 1 的二值图像，假设 S1 为像素值为 0 的一堆背景点，S2 为像素值为 1 的一堆前景点，外层轮廓 B1 为一堆前景点最外围的点，内层轮廓 B2 为一堆前景点最内部的点。通过对二值图像进行行扫描给不同轮廓边界赋予不同的整数值，从而确定轮廓的类型以及之间的层次关系。提取出外层轮廓后通过计算轮廓包含的面积大小丢弃面积较小的外层轮廓，输出第一步候选轮廓。



（提取轮廓）

闲鱼页面瀑布流卡片轮廓近似于矩形，在四个角由弧形曲线连接。对于提取的候选轮廓使用 Douglas-Peucker 算法进行矩形逼近，保留形状接近矩形的外轮廓。Douglas-Peucker 算法通过将一组点表示的曲线或多边形拟合成另一组更少的点，使得两组点之间的距离满足特定的精度。之后输出第二步候选轮廓。

通过对第二步候选轮廓所处位置对应的第一步候选轮廓进行水平和垂直方向投影，去除毛刺影响，输出矩形轮廓。轮廓提取的结果如下图所示，其中 c) 中轮廓包含面积设置的阈值为 10000，d) 中 Douglas-Peucker 算法设置的精度为 0.01* 轮廓长度。本文所有提取的轮廓均包含输入框。

（不同方法识别结果）

我们再介绍下机器学习如何处理：

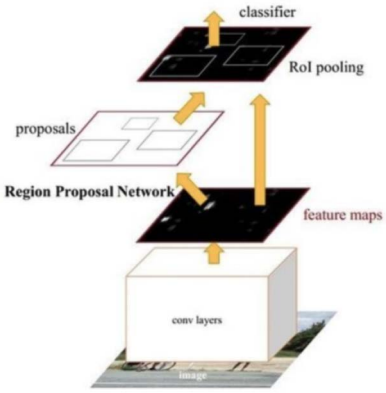
传统算法中提出采用传统图像处理方法提取轮廓特征，这样会带来一个问题：当图像不清晰或者有遮挡的情况下无法提取出轮廓，即召回率不是很高。

基于卷积神经网络的目标检测算法能通过输入大量样本数据，学习到更具有代表性和区别性的特征。目前目标检测算法主要分成两个派系：以 R-CNN 家族为代表的两阶段流和以 YOLO、SSD 为代表的一阶段流。一阶段流直接对预测的目标进行分类和回归，优点是速度快，缺点是 mAP 整体上没有两阶段流高。两阶段流在对预测的目标进行分类和回归前需要先生成候选的目标区域，这样训练时更容易收

敛，因此优点是 mAP 高，缺点是速度上不如一阶段流。不管是一阶段流还是两阶段流，通用的目标检测推理过程如图所示。输入一张图像到特征提取网络 (可选择 VGG、Inception、Resnet 等成熟的 CNN 网络) 得到图像特征，然后将特定区域特征分别送入分类器和回归器进行类别分类和位置回归，最后将框的类别和位置进行输出。

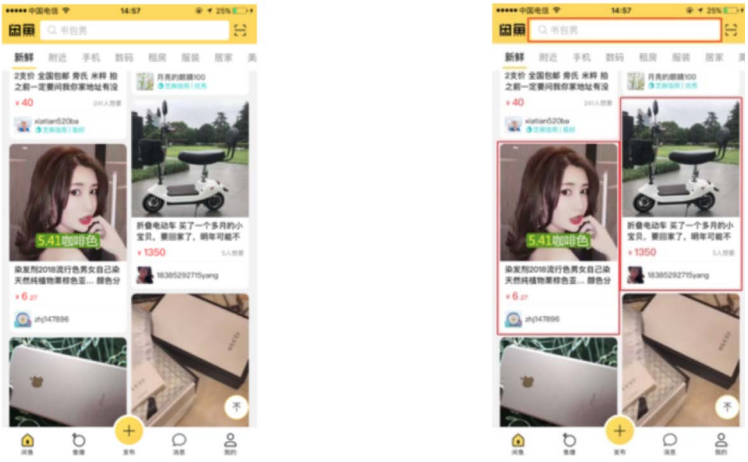
（目标检测网络流程示意）

FasterR-CNN 对 R-CNN 家族最大的贡献是将生成候选目标区域的过程整合到整个网络中，使得综合性能有了较大提高，尤其是在检测速度上。FasterR-CNN 的基本结构如图所示。主要分为 4 个部分：1)convlayers。作为一种特征提取网络，使用一组基础的 conv+relu+pooling 层提取图像的特征，该特征被共享用于后续 RPN 网络和全连接层。2)RegionProposalNetwork。该网络用于生成候选目标框，通过 softmax 判断候选框是属于前景还是背景，并且通过候选框回归修正候选框位置。3)RoIpooling。收集输入的特征图和候选区域，将这些候选区域映射到固定大小并送入后续全连接层。4)classifer。计算候选框的具体类别，并且再次回归修正候选框的位置。



（目标检测网络基本结构示意）

使用 FasterR-CNN 进行瀑布流卡片的识别，得到下图的结果。



a)　　b)

（目标检测结果）

传统算法与机器学习的融合

描述的传统图像方法能够获得高定位精度的卡片位置，但受卡片本身模式的影响，召回率不高）中右边卡片没有检测到。基于目标检测的深度学习方法具有较高的

泛化能力，能获得较高的召回率，但是回归过程无法得到高定位精度的卡片位置）中卡片都能检测出来，但有两个卡片的边缘几乎粘连在了一起。

将两种方法得到的结果融合在一起，能同时获得高精确率、高召回率、高定位精度的检测结果。融合过程如下：

1. 输入一张图像，并行运行传统图像处理方法和深度学习方法，分别得到提取的卡片框 trbox 和 dlbox。定位精度以 trbox 为标杆，精确率和召回率以 dlbox 为标杆
2. 筛选 trbox。规则为当 trbox 与 dlbox 的 IOU 大于某个阈值时（比如 0.8）保留此 trbox，否则丢弃，得到 trbox1
3. 筛选 dlbox。规则为当 dlbox 与 trbox1 的 IOU 大于某个阈值时（比如 0.8）丢弃此 dlbox，否则保留，得到 dlbox1
4. 修正 dlbox1 位置。规则为 dlbox1 的每条边移动到距离其最近的一条直线上，约束条件为移动的距离不能超过给定的阈值（比如 20 个像素），并且移动的边不能跨越 trbox1 的边，得到修正的 dlbox2
5. 输出 trbox1+dlbox2 为最终融合的卡片框

结果

先介绍几个基本指标：

TruePositive(TP): 被模型预测为正的正例数

TrueNegative(TN): 被模型预测为负的负例数

FalsePositive(FP): 被模型预测为正的负例数

FalseNegative(FN): 被模型预测为负的正例数

精确率 (Precision)=TP/(TP+FP): 反映了被模型预测的正例中真正的正样本所占比重

召回率 (Recall)=TP/(TP+FN): 反映了被模型正确预测的正例占总的正样本比重

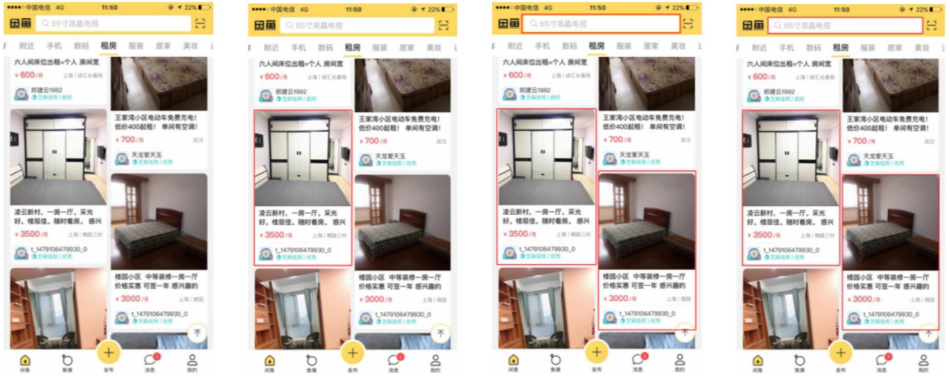
定位精度 (IOU)= 两个框的交集大小 / 两个框的并集大小

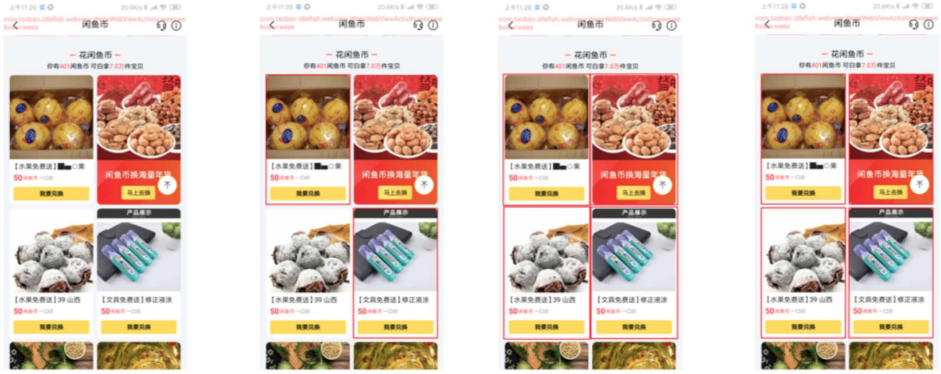
a) b) c) d)

（不同方式得到的检测结果）

上图分别显示了不同方法识别的卡片 ,d) 相对于 b) 的优势是提高了召回率，d) 相对于 c) 的优势是提高了定位精度。图一图二图三显示了一些其他实例图像的识别，每行图像是一类实例图，第一列是原始图像，第二列是传统图像处理识别的卡片，第三列是深度学习识别的卡片，第四列是融合的卡片。

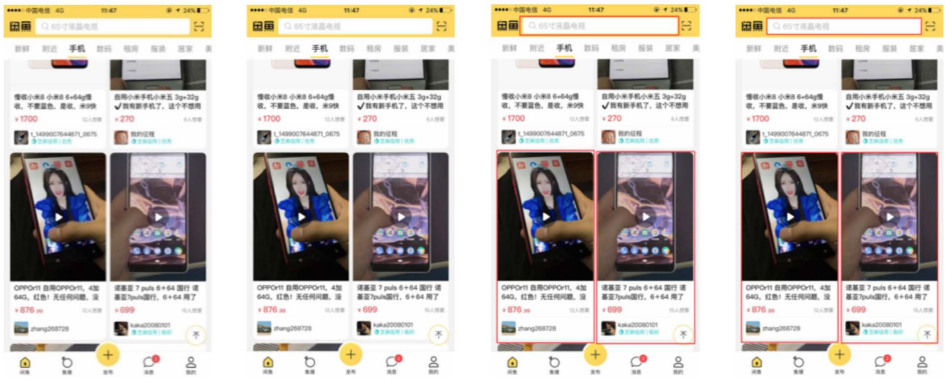
图一图二能够准确识别卡片轮廓：





（前景识别结果示例 I）

图三融合卡片的下边缘并没有完全贴合，这是因为融合步骤中修正 dlbox1 位置时，采用传统图像处理方法寻找临域范围内最近的直线，受到图像样式的影响，找到的直线并不是期望的卡片下边缘。



（前景识别结果示例 II）

实验过程中随机截取了 50 张闲鱼瀑布流卡片图像，共有卡片 96 个 (不包含输入框)，对每张图像分别采用传统图像处理方法、深度学习方法、融合方法得到卡片的识别结果，其中传统图像处理方法共识别出 65 个卡片，深度学习方法共识别出 97 个，融合后共识别出 98 个。精确率、召回率、定位精度如下表所示。融合后识别结果结合了传统图像处理方法定位精度高、深度学习方法召回率高的优点。

不同方法结果

指标 / 方法	精确率	召回率	平均定位精度
传统图像处理	1.0	0.6770	0.9838
深度学习	0.9896	1.0	0.9611
融合	0.9795	1.0	0.9830

前景算法小结

通过对闲鱼页面瀑布流卡片识别过程中的描述，我们简单介绍了前景处理的探索，通过机器视觉算法和机器学习算法协同完成前景元素的提取和识别。

结束语

本篇我们通过对前景提取和背景分析的介绍，提出了一种通过传统图像处理和深度学习相融合的方法，来得到高精确率、高召回率和高定位精度的识别结果。但方法本身还存在一些瑕疵，比如融合过程对组件元素进行修正时也会受到图像样式的干扰，后续这部分可以进一步进行优化。

复杂背景内容提取

复杂背景内容提取指的是从复杂的背景中提取出特定的内容，例如在图片中提取特定的文字，在图片中提取特定的叠加图层等等。

这是一个业界难题，基于传统的图像处理的方法存在准确率和召回率的问题，没法解决语义的问题。而主流的机器学习的方法，例如目标检测无法获取像素级别的位置信息，而语义分割的方法则只能提取像素而无法获取半透明叠加前的像素信息。

本文考虑到这些痛点，本文采用了目标检测网络来实现内容召回，GAN 网络实现复杂背景中特定前景内容的提取和复原。

复杂背景的处理流程分为如下几个步骤：

```
1  内容召回：通过目标检测网络召回元素，即元素是否需要做背景提取操作。
2  区域判断：根据梯度等视觉方法判断所处区域是否是复杂区域。
3  简单区域：基于梯度的方式找到背景区块。
4  复杂区域：采用 SRGAN 网络进行内容提取。
```

内容召回：

内容召回我们采用目标检测网络来实现，例如 Faster-rcnn 或者 Mask-rcnn 等，如下图所示：

区域判断：

根据拉普拉斯算子计算周边梯度，判断所处区域是否是复杂区域。

简单背景：

由于目标检测模型本身的局限性，会导致没法达到像素级别的精确性，因此需要对位置做修正。如果是简单背景就可以基于梯度的思想做位置修正，具体计算方式如下：

$$\textbf{\textit{定义函数}}\ \mathrm{grad}(x,y) = abs(f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y))$$

$$\textbf{\textit{定义函数}}\ \mathrm{IOU}(\mathrm{rectA}, \mathrm{rectB}) = (\mathrm{rectA} \cap \mathrm{rectB})/(rectA \cup rectB)$$

$$\textbf{\textit{原始坐标}} x_1, y_1, w_1, h_1$$

$$\textbf{\textit{求修正坐标}}\ x_2, y_2, w_2, h_2$$

$$\textbf{\textit{目标}} \max IOU\ ([x_1, y_1, w_1, h_1], [x_2, y_2, w_2, h_2])$$

$$\textbf{\textit{满足条件}}:$$

$$\sum_{h=0}^{h2} grad(x_2 - 1, y_2 + h) = 0, \sum_{h=0}^{h2} grad(x_2, y_2 + h) > 0$$

$$\sum_{w=0}^{w2} grad(x_2 + w, y_2 - 1) = 0, \sum_{w=0}^{w2} grad(x_2 + w, y_2) > 0$$

$$\sum_{h=0}^{h2} grad(x_2 + w_2 + 1, y_2 + h) = 0, \sum_{h=0}^{h2} grad(x_2 + w_2, y_2 + h) > 0$$

$$\sum_{w=0}^{w2} grad(x_2 + w, y_2 + h_2 + 1) = 0, \sum_{w=0}^{w2} grad(x_2 + w, y_2 + h_2) > 0$$

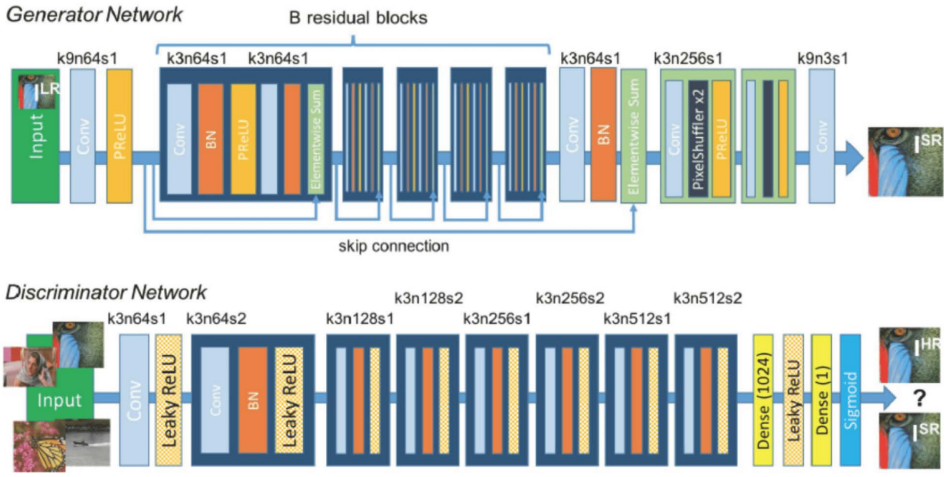
（简单背景位置修正公式）

复杂背景：

背景是复杂背景时，左图是原图，右图是提取的文字区块：

（原图和文字区域）

此时提取出的框不是完全正确，那么此时根据梯度等机器视觉算法已经不能对位置做正确的修正了。本文提出了基于 GAN 网络的方式来解决复杂背景内容提取问题，网络的主要结构如下图所示：



（GAN 网络流程图）

为什么选择 GAN 网络?

1）基于 srGAN 网络，该网络加入了特征图的损失函数，这样可以很好保留高频信息，能更好的保留边缘。特征图的损失函数如下图所示：

$$l^{SR}_{FeatureMap} = \frac{1}{WH}\sum_{x=1}^{W}\sum_{y=1}^{H}(\emptyset(I^{HR})_{x,y} - \emptyset(G(I^{LR}))_{x,y})^2$$

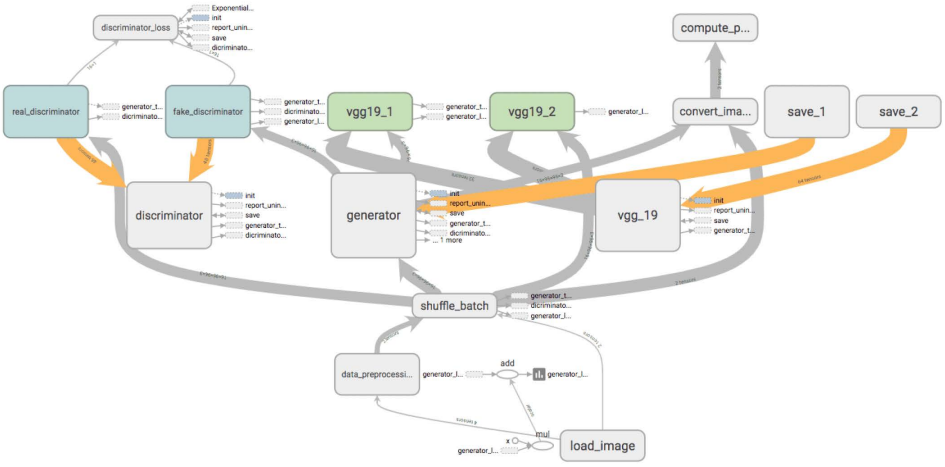
（特征图损失函数）

该公式将原图和生成图特征值差的平方做为损失函数。

2）由于有对抗损失的存在，可以很好的降低误检率。

3）最重要的一点是在有透明度的场景下，语义分割网络只能“提取”元素，无法“还原”元素。而 GAN 网络不仅可以在提取元素的同时还原出未叠加时的像素情况。

网络训练流程图：



（srGAN 网络训练流程）

针对业务场景对 GAN 网络做的改进：

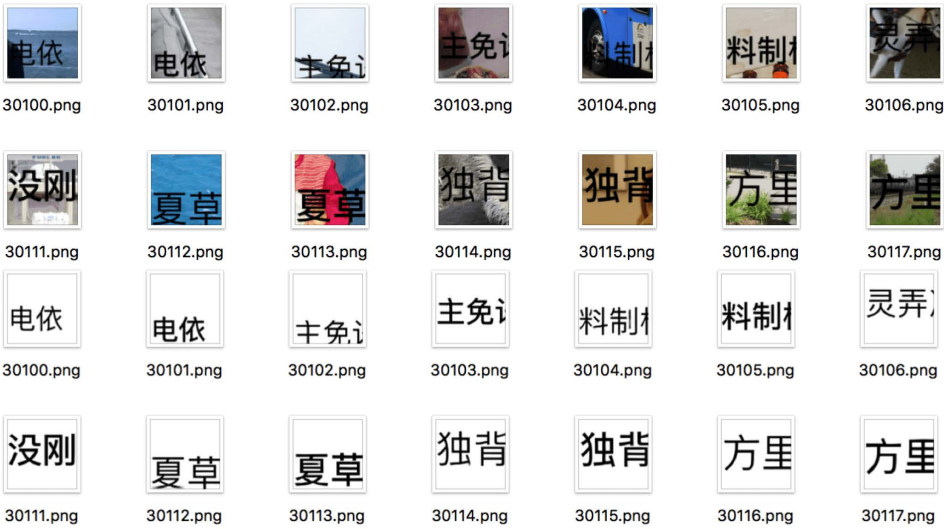
1. 由于我们不是超分辨率场景，因此不用 pixelShuffler 模块做上采样。
2. 由于场景比较复杂，可以引入 denseNet 和加深网络来提高准确率。
3. 内容损失函数对于压制误判的噪点效果不理想，因此加大了误判的惩罚，具体如下图所示：

```
# Compute the euclidean distance between the two features
diff1 = tf.where(tf.equal(gen_output, targets), gen_output - targets, 4 * (gen_output - targets))
diff2 = extracted_feature_gen - extracted_feature_target
content_loss = tf.reduce_mean(tf.reduce_sum(tf.square(diff1), axis=[3])) + \
               0.0061 * tf.reduce_mean(tf.reduce_sum(tf.square(diff2), axis=[3]))
```

预测获取的结果图 I:

（复杂背景的文字内容提取）

预测获取的结果图 II:



（原图和相应的处理结果）

结束语

本篇我们通过复杂背景内容提取的介绍，提出了一种机器学习为主，图像处理为辅去精确获取特定前景内容的方法，得到了高精确率、高召回率和高定位精度的识别结果。

下图分别是传统算法，语义分割方法和本文融合方法的各个指标的情况。

千余张闲鱼复杂背景图

方法 \ 指标	精确率	召回率	定位精度
传统算法	0.67	0.67	0.82
语义分割	0.83	0.94	0.82
本文的方法	0.91	0.96	0.91

（不同算法的识别结果）

业务场景落地

本篇我们提出的方法已经应用在如下场景：

1. imgcook 图片链路中应用，对于通用场景的准确率能达到 73%，特定的卡片场景能达到 92% 以上。
2. 淘宝自动化测试图像内容理解，例如应用在 99 大促和双 11 模块识别中。整体的准确率和召回率都能达到 97% 以上。

未来展望

未来我们打算从图片链路出发，做到如下几点：

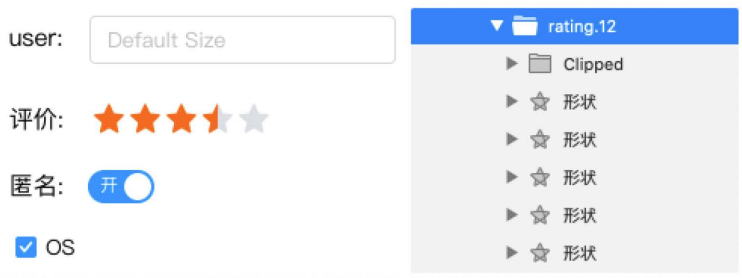
1. 丰富和完善布局信息，能够精确地识别 listview，gridview，waterfull 等布局信息。
2. 进一步提高通用场景的准确率和召回率。针对小目标，我们后续会引入特征金字塔（fpn），Cascade 等一系列技术来提高我们的准确率和召回率。
3. 增加适配的特定场景。现有的场景只覆盖了闲鱼和部分淘宝的页面适配，我们希望后续能够支持更多的页面，进一步提高图像的泛化能力。
4. 引入图片样本制造机，降低特定场景的接入门槛。

组件识别篇

作为阿里经济体前端委员会四大技术方向之一，前端智能化项目经历了 2019 双十一的阶段性考验，交出了不错的答卷，天猫淘宝双十一会场新增模块 79.34% 的线上代码由前端智能化项目自动生成。在此期间研发小组经历了许多困难与思考，本次《前端代码是怎样智能生成的》系列分享，将与大家分享前端智能化项目中技术与思考的点点滴滴。

背景介绍

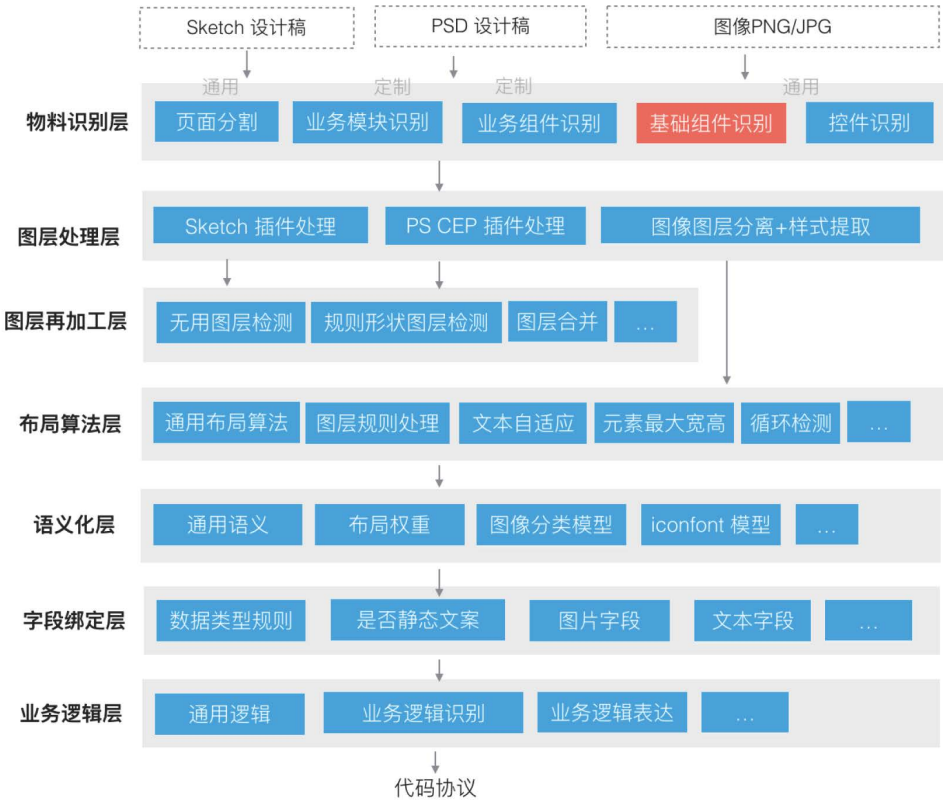
我们在做一些 Deisgn2Code 的时候，一种比较直接的方式是借助设计师使用的设计工具的开发插件来帮我们提取设计稿里的元数据，我们可以快速提取设计稿里的图像、文本、Shape 等原生元素，从而组装生成一个页面。但目前我们的开发体系里还有许多基础组件是不在设计工具体系里，比如表单、表格、开关等组件，虽然像 Sketch 这类工具提供了灵活的方式能让设计师设计出相应的 UI 类型，但是同样的 UI 所对应 Sketch 里的 DSL 描述并不是我们所期望的。此时，我们往往需要借助其他途径来获取相对准确的描述，通过深度学习不断学习所需要识别的组件是一个比较好的方式。



（Sketch 中的设计图层描述）

所在分层

本文讲述前端智能化 D2C 里技术分层中的**基础组件识别**能力层，主要识别图像中可能存在预先定义的一些基础组件，从而辅助下游技术体系中对已识别的图层进行表达优化，比如优化图层嵌套结构从而产出合规的组件树，优化图层语义化的结果等。



（D2C 技术能力分层）

整体方案

参照以往的算法工程方案，我们的整体方案涉及到**样本获取**、**样本评估**、**模型训练**、**模型评估以及模型预测**一整条流程链路。

（算法工程 Pipeline）

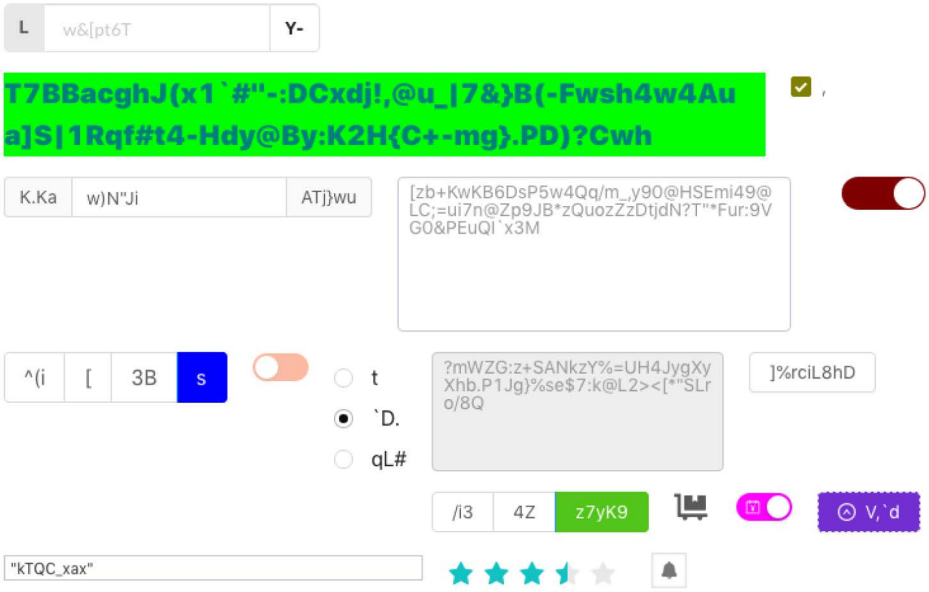
样本获取 & 评估

样本集是整个模型的关键，样本的质量决定了模型质量的上限，算法模型的调优只是不断得往这个质量靠近。我们的样本可以来源于多种途径，因为此模型涉及到组件上下文的识别场景较少，此篇的基础组件识别模型我们选择结合业界使用量比较多的一些 UI 类库自己编码生成样本，确保编程部分对样本数据质量的保障。

在具体的样本编码开始前，我们需要简单对我们需要识别的组件做一个简单的分类，设定好此次分类的类目类型。在对整个样本的生成上，我们需要遵循以下几点原则：

- 数据种类丰富且均等，即每种种类数量一致，尽可能涵盖多种种类（组件属性、样式影响，组件库的样式尽量只在合理的范围内随机化通用的背景色、宽高、圆角等，泛化组件样式）
- 针对一些特定场景（比如覆盖问题），也需要构造相关的场景（图上文本、图上图遮罩等）
- 针对一些线框型的组件（比如 Input），可以在周边 Padding 几像素的空白避免模型处理学习到边缘特征

下图为一个简单样本的 DEMO：



（基础组件识别样本）

在样本生成后，你可能需要对自己的样本集进行一个评估，评估过程中可以适当引入**数据校验**和**类目统计**等相关的工程，来评估整体的样本质量：

- 检测标注数据是否存在错误：参考标注区域结合背景色计算方差等方式
- 统计所有 UI 类型的数据分布情况，每个分类的数量以及数据是否均衡等

模型选型

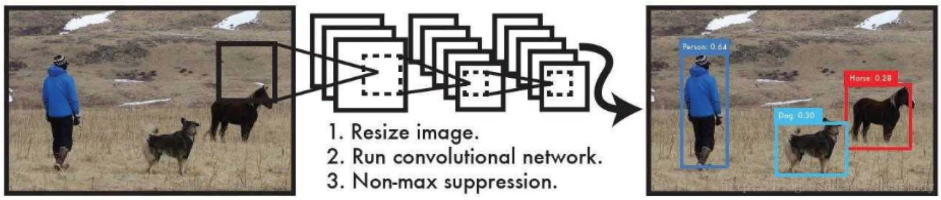
在目前众多的目标检测模型中，根据一份最近的 PASCALVOC 数据集的目标检测模型报告，我们选择了一种排名第一的 YOLOOne-Stage 算法作为我们基础组件模型的迁移学习，来快速进行试验。

	mean	aero plane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	dining table	dog	horse	motor bike	person	potted plant	sheep	sofa	train	tv/ monitor	submission date
** Stronger-yolo ** [?]	**83.3**	91.9	89.1	82.5	**75.2**	72.9	**87.3**	**87.8**	91.0	**71.3**	85.1	70.0	90.0	**90.8**	**90.3**	**91.4**	**67.5**	86.4	74.6	**89.9**	**81.5**	12-Jun-2019
** FCASA-detection ** [?]	82.4	90.9	87.2	83.8	72.3	72.0	86.3	87.7	90.2	69.8	85.1	**71.2**	89.7	90.0	89.3	90.6	61.1	85.3	**75.1**	89.5	80.1	05-Aug-2019
** DOLO ** [?]	81.3	91.7	87.3	83.1	69.1	71.1	85.7	86.6	93.4	64.4	85.5	65.9	92.2	88.5	89.0	88.7	61.0	86.0	71.0	87.4	77.4	21-Sep-2018
** COS-DET ** [?]	81.3	91.7	87.2	82.1	71.6	68.6	86.9	85.3	93.1	63.8	86.8	66.0	92.0	90.4	88.4	88.8	61.2	86.8	73.8	88.1	73.7	26-Apr-2019
** ASSD513 ** [?]	81.3	**92.1**	**89.2**	82.5	71.5	60.4	85.5	84.8	93.9	63.7	**88.6**	67.4	92.6	90.2	89.0	86.5	60.4	**88.2**	73.4	88.6	77.0	18-Aug-2018
FastX-RCNN [?]	81.1	89.7	86.4	**84.1**	70.9	**73.1**	84.6	85.5	**94.3**	64.7	85.3	62.2	**93.4**	90.2	88.8	89.9	62.1	83.8	71.2	88.7	73.9	06-Jul-2018
** BOE_IOT_AIBD_method ** [?]	81.0	88.7	84.5	78.4	71.7	68.7	85.5	82.5	94.0	68.3	87.0	67.6	92.8	90.3	88.7	89.9	63.3	81.9	71.8	87.6	76.3	19-Sep-2019
refine_denseSSD [?]	77.5	89.8	85.8	77.0	64.4	56.7	83.7	81.8	92.1	60.9	83.8	63.2	89.6	85.9	88.1	85.3	54.7	82.3	64.6	88.2	72.4	14-May-2018
** FPNSSD ** [?]	77.0	90.3	78.8	81.7	67.1	53.4	79.5	80.5	93.8	59.9	85.8	61.8	92.5	81.7	84.1	80.8	56.1	84.8	69.2	87.4	71.2	29-Mar-2018
TCnet [?]	76.6	86.6	83.1	78.5	65.6	61.1	80.8	80.3	91.7	56.3	80.1	61.8	90.5	86.1	84.0	83.4	56.6	79.7	70.0	84.5	71.9	02-May-2018
** TCnet ** [?]	76.5	86.8	82.7	78.5	65.3	60.2	79.6	80.0	91.0	56.9	80.9	61.3	90.2	86.8	84.2	83.1	55.4	80.3	70.0	84.7	71.7	29-Mar-2018
** ASSD321 ** [?]	76.4	89.6	84.3	76.7	64.5	49.3	81.7	77.0	92.2	57.8	81.3	64.0	91.6	86.5	85.8	82.1	53.0	80.0	70.9	87.2	71.8	20-Aug-2018
** FSSD512 ** [?]	75.9	88.6	84.7	74.9	62.1	57.2	83.4	83.1	88.7	56.8	78.3	60.8	87.1	84.6	86.3	85.3	54.4	80.1	65.9	86.7	69.0	17-Jun-2019
ATLSSD [?]	74.8	87.6	82.7	72.0	62.2	57.5	83.1	83.8	86.9	56.2	76.3	60.6	84.4	80.4	84.9	85.9	50.1	81.1	65.5	84.9	70.1	26-Mar-2018
DSD [?]	74.5	87.9	82.0	74.8	61.9	51.5	82.1	81.1	89.8	55.8	78.5	58.3	86.8	82.3	82.7	83.4	49.2	79.5	69.1	85.0	69.2	19-Jul-2018
dsa_1050 [?]	73.9	87.4	82.0	72.9	60.7	51.8	80.7	76.8	90.1	54.0	78.7	60.0	89.1	83.5	83.3	81.4	49.7	75.7	64.2	85.2	70.5	18-Nov-2017
MA-SSD [?]	72.9	87.0	81.4	71.2	59.4	49.0	81.3	74.4	88.2	55.5	78.2	61.2	85.9	82.7	82.7	80.3	46.5	76.6	66.8	83.7	66.2	01-Aug-2018

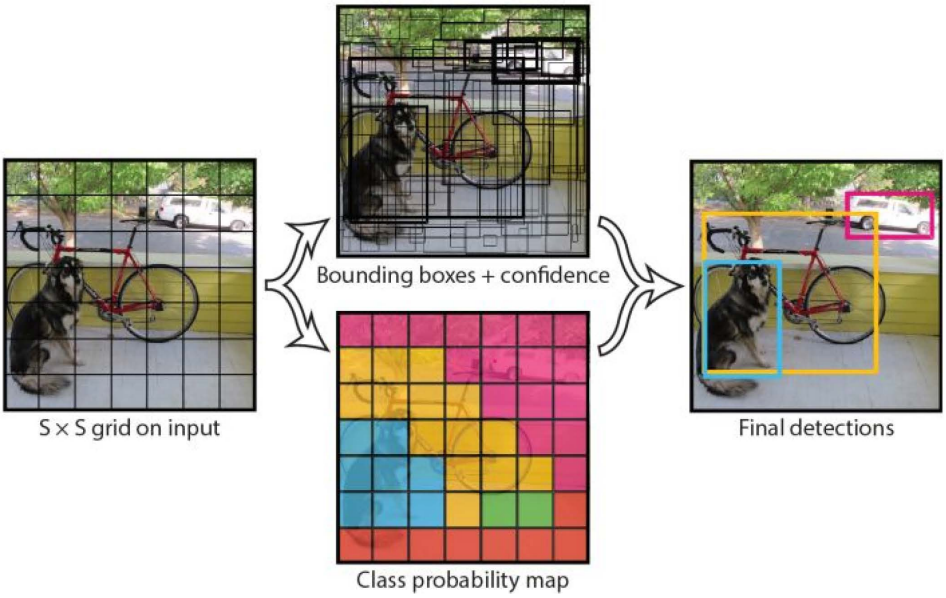
（各大目标检测模型数据分析报告）

简单介绍下，YOLO 即 YouOnlyLookOnce，主要分为三个步骤：

1. 将图像 Resize 到 416*416（v3）尺寸
2. 卷积网络学习样本集里的分类特性
3. 预测的时候进行非极大值抑制，筛选 Boxes



YOLO 采用一个单独的 CNN 模型来实现端到端的目标检测，相比其他的二段流（R-CNN 等）算法，YOLO 的训练和预测速度更快。它将输入的图片切分成 S*S 网格，然后让每个单元格负责去检测中心点落在各个单元格中的目标，每个单元格会预测所在目标的边界框（boundingbox）以及边界框的置信度（confidencescore），置信度包含边界框内含目标的可能性大小以及边界框的准确度。最后将各个单元格预测的结果做整合得到最终的预测结果。YOLO 相关的其他更多设计细节，感兴趣的同学可以访问 yolo 官网来了解。



我们在做 Web 里的基础组件的目标检测时，期望卷积网络在相应的单元格中能够学习到其中每一类组件的特征，从而能够识别并区分出不同组件之间的差异，因此在对组件样本的选取上，要保障组件之间的特性差异，避免卷积网络丢失这部分的学习。

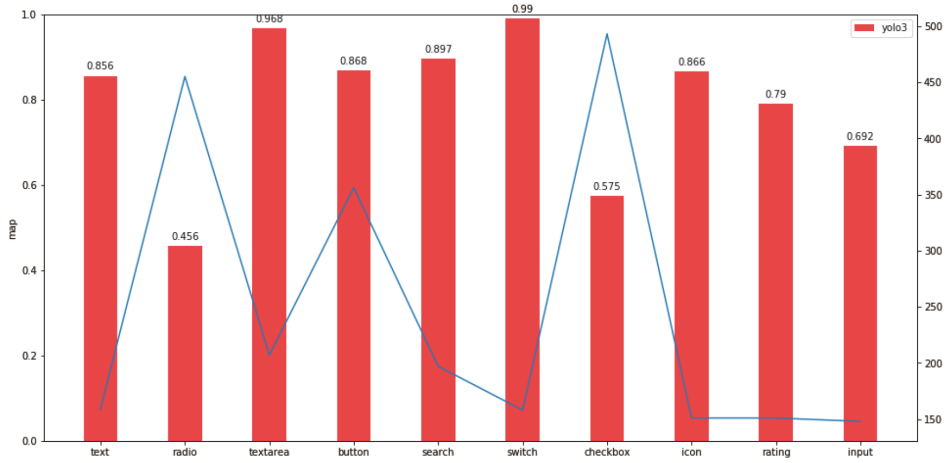
模型评估

在目标检测模型的评估中，我们选择使用均值平均精度 mAP 来进行衡量模型的准确率（基于 COCO 的算法）。可以选择部分测试集数据的模型预测结果，对结果数据跟数据的真实值（GroundTruth）进行比较，从而计算出每个分类的 AP。

```
-------------------------------------------------   switch   ----------------------------
Running per image evaluation...
Evaluate annotation type *bbox*
DONE (t=0.06s).
Accumulating evaluation results...
DONE (t=0.01s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.889
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.990
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.990
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.902
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.880
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.377
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.908
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.908
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.905
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.911
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = -1.000
```

（模型 AP 评估）

此处可以选择 IoU 超过 0.5 的对所有分类进行一个简单的制图比较，可以观察到目前对于小目标的识别检测（部分文本元素影响因素较大）精度较差，后续在样本的处理上可以进一步对此类小目标结合规则部分的预处理做检测加强。



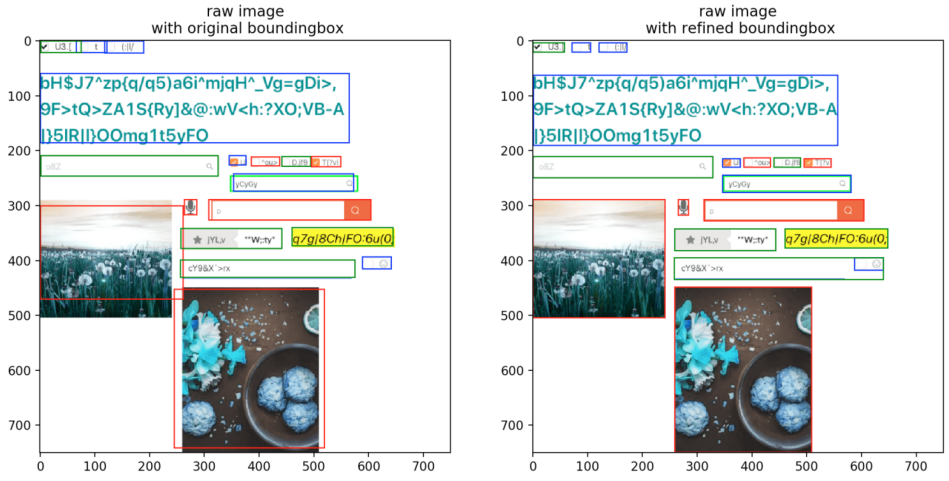
（模型 mAP 评估）

模型预测处理

因为模型的迁移学习上我们选用了 yolo3，在对图像做处理时都会将图像预处理

成 416*416 的尺寸做训练学习，为了在模型预测的时候结合训练数据得到更好的效果，我们可以在预测阶段的前置过程中也可将输入的图像做一定尺寸的适配，经测试适配后的图像预测后的 IoU 比未经适配的图像预测后的 IoU 高出很多（测试集显示有 10%+ 的提升）。

图像尺寸适配后，我们训练完的模型会对输入的图像直接做预测，此时会得到一个大致的类别区域框，但往往我们需要一个围绕 UI 组件的更加精确的框，此时我们可以借助 OpenCV 的能力结合图像本身做一个梯度裁边 Refine 以达到一个更精确的效果，如下图是边框 Refine 前后的对比。



（模型预测结果优化）

总结

目前前端智能小组里的基础组件包含 20+ 种类，后续将继续在样本精细化的分类以及基础组件的属性测量表达上做进一步的投入研发，并将模型的数据样本管理规范化，统一做输出。未来用户可以根据我们面向外部开源的样本集对部分基础组件识别后做特定的表达处理。

表单表格专项识别篇

作为阿里经济体前端委员会四大技术方向之一，前端智能化项目经历了 2019 双十一的阶段性考验，交出了不错的答卷，天猫淘宝双十一会场新增模块 79.34% 的线上代码由前端智能化项目自动生成。在此期间研发小组经历了许多困难与思考，本次《前端代码是怎样智能生成的》系列分享，将与大家分享前端智能化项目中技术与思考的点点滴滴。

概述

在前端智能化领域，特别是中后台智能方向上，表单表格的识别是非常重要的一环。因为表单表格的开发工作，占据了中后台前端开发工作量中的绝大部分，如果能够通过智能的手段，从设计稿图片秒级生成表单表格代码，那么将会是巨大的生产力提升。本文将会揭秘秒级生成表单表格代码的技术细节。

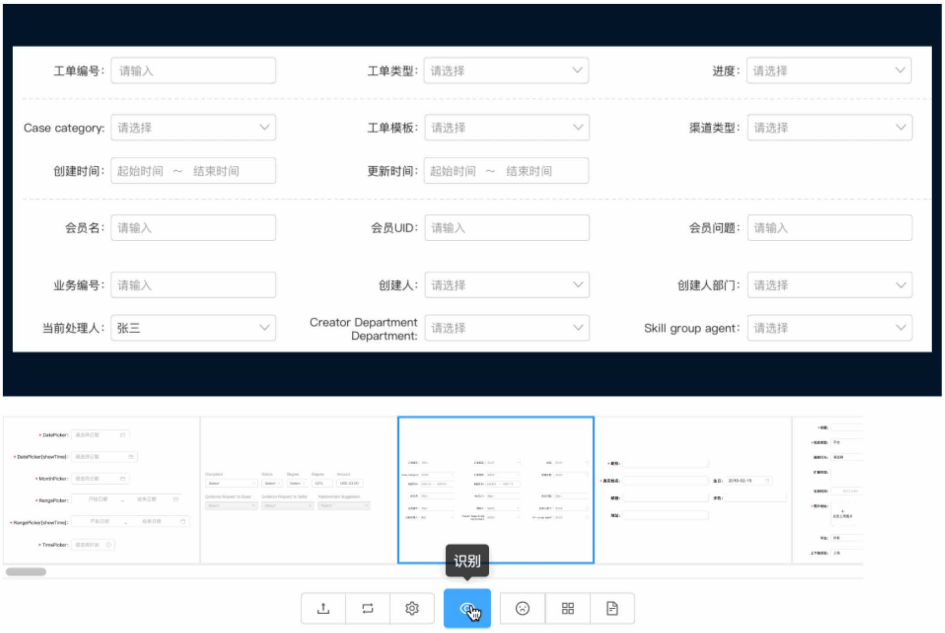
所在分层

表单 / 表格识别使用了组件识别、布局算法、物料属性识别等技术，在这些技术中，核心技术是物料属性识别，在整体分层中的物料识别层中，如图所示。

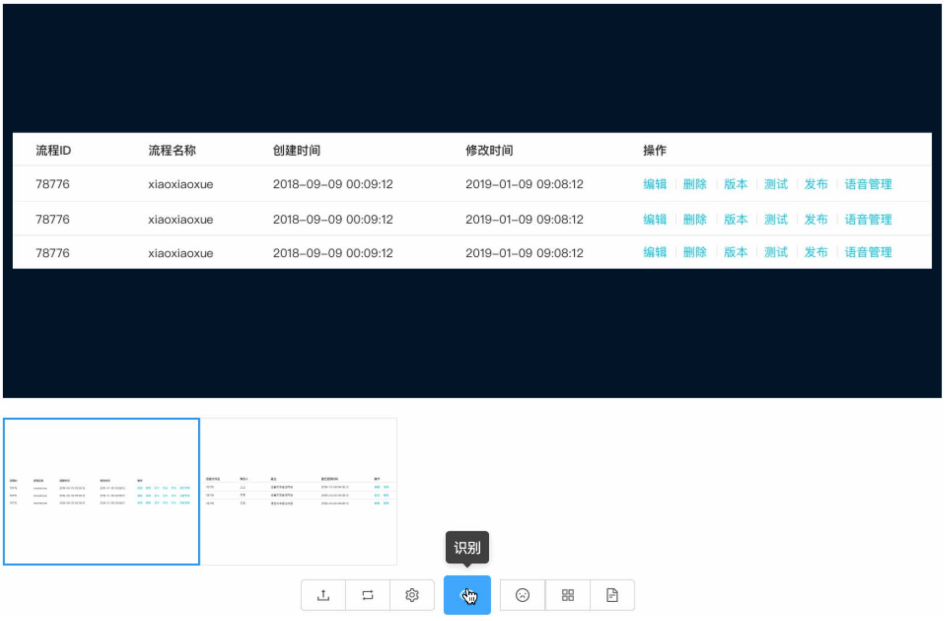
（D2C 识别能力技术分层）

秒级生成表单 / 表格代码效果展示

只需要截图粘贴，并点击识别，就可以秒级生成表单、表格等通用前端组件协议，甚至连字段都翻译好了，还是小驼峰命名法哦！



表单识别效果图



表格识别效果图

表单 / 表格识别技术揭秘

表单识别的主要思路是：

1. 通过目标检测技术检测出所有的组件类型及其坐标。
2. 通过文字识别技术和自动翻译技术识别出所有文字及其坐标并翻译为英文。
3. 通过代码转换器从上述组件信息和文字信息中提取表单 / 表格的布局、label、type、字段等各种属性。

目标检测 + 文字识别

目标检测使用的 fasterrcnninceptionv2 模型训练和预测的，具体细节参考本系列文章中的组件识别篇。文字识别使用的通用的文字识别技术，可以检测出文字内容和坐标，具体细节也不再赘述。

图中红框为目标检测信息，绿框为文字识别信息

代码转换器

本文将重点介绍，如何使用代码转换器提取表单 / 表格的各种属性。

绝对坐标转行列

首先，为了方便处理信息，我们先把目标检测信息和文字识别信息的绝对坐标转为行列，行列的具体数据结构是个二维数组，第一维是列，第二维是行。具体算法思路如下：

- 将所有识别出的带有绝对坐标的框垂直排序。
- 遍历垂直排序后的框，将同一行的框放到一个数组中并水平排序，作为一行。
- 将所有行按先后顺序放到一个数组中，最终生成行列二维数组。

计算表单 / 表格布局

通过上述行列信息，我们可以计算出表单 / 表格的布局信息了。

先看表单，表单项的布局是二维的，与上述组件识别的行列信息一致，所以直接将组件识别行列信息通过嵌套循环 map 为表单协议即可。而表格则是一维的，只要表头就可以确定表格的结构了，那么我们直接从文字识别行列信息中提取第一行作为表头即可。

计算表单 / 表格字段值

计算完了布局，我们来算具体的字段值及其类型。

先看表单，表单字段的值就是 label 翻译为英文，并转为小驼峰。那么如何提取 label 呢？根据常识我们知道表单项的 label 要么在左边，要么在上边，所以我们的算法就是：先提取左边的 label，没有的话，就提取上面的 label。

再看表格，表格字段就是表头，我们从文字识别行列信息中提取第一行即可拿到表头的所有值。为了保险起见，可以 doublecheck 一下，第一行是不是表头，比如通过长度来过滤，长度小于 3 的行就过滤掉了。

计算表单 / 表格的字段类型

由于表单项有 input、select、redio 等类型，表格有纯文本、链接等类型，所以我们还需要计算这些字段类型。

先看表单，表单字段类型从目标检测信息中拿，因为目标检测的任务之一就是为了提取表单项的类型。

再看表格，表格字段类型同样也是从目标检测信息中拿，不过由于表格每一列的类型都是相同的，所以我们只需要提取每一列的第一个类型即可。

计算表单 / 表格其他属性信息

上述信息基本上已经可以帮我们节约大量工作量了，但是如果还想提取更多的属性信息，可以使用如下办法：

- **递归识别**：比如，识别出一个 input 之后，将其剪裁出来，递归到另一个模型中识别，比如可以识别是否是 disabled，是否必选等等。
- **继续提取其他位置的文字信息**：比如表单项中间的可能是 placeholder 或者 defaultValue。

未来展望

逍遥子在今年的云栖大会说过，大数据和算力是数字经济时代的石油和发动机。当前在前端行业，组件化已经初具规模，海量的组件可以作为大数据，同时，业界算力也在不断提升，人工智能技术有很大可能性会改变前端开发的格局，让我们拭目以待。

业务模块识别篇

作为阿里经济体前端委员会四大技术方向之一，前端智能化项目经历了 2019 双十一的阶段性考验，交出了不错的答卷，天猫淘宝双十一会场新增模块 79.34% 的线上代码由前端智能化项目自动生成。在此期间研发小组经历了许多困难与思考，本次《前端代码是怎样智能生成的》系列分享，将与大家分享前端智能化项目中技术与思考的点点滴滴。

概述

无线大促页面的前端代码中，存在大量的业务模块或业务组件（下文统称业务模块），即具有一定业务功能的代码单位。获取页面中业务模块的信息之后，可以用于复用代码、绑定业务字段等后续功能。因此从视觉稿识别出业务模块，在前端智能化领域中成为用途广泛的功能环节。

与面向中后台的基础组件识别和表单识别功能不同，业务模块识别主要面向无线端页面，并且来源主要是视觉稿。相对的，业务模块 UI 结构更加复杂，并且视觉稿提供的内容已经有较多可辨别的信息（如文本内容、图片尺寸等），因此我们没有直接使用图片深度学习的方案，而是从视觉稿产出的 DSL 中提取预定义的特征值，用传统学习多分类的方法来实现模块识别。本识别功能最终返回业务模块的类别、视觉稿中的位置等信息。

总体功能如下图所示。包括：

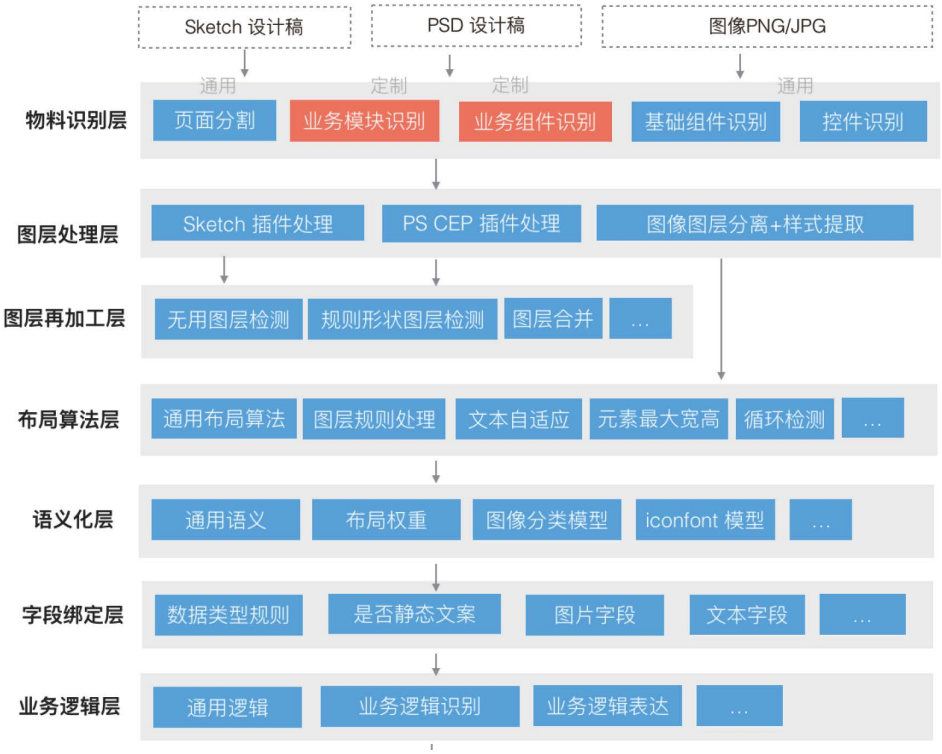
- 样本构造，根据用户配置和自定义的数据增强规则对视觉稿进行 UI 层的增强，以得到视觉多样化的样本。然后在定义好业务字段的基础上，进行特征值抽取并存储。

- 算法选择，目前提供的都是传统机器学习方法中的多分类算法。
- 模型实现，基于集团机器学习平台实现模型搭建及相关算法工程，做到自动化训练与部署。
- 接口提供，模型对外提供预测识别服务以及结果反馈服务。

（总体功能）

所在分层

如下图所示，我们的业务模块识别服务位于物料识别层，为视觉稿导出的 DSL 提供进一步的业务定制化的识别能力，在后续代码生成的过程中渗透到字段绑定、业务逻辑等功能之中。



（D2C 功能分层）

样本构造

机器学习是基于大量真实数据的训练过程，一个好的样本库可以让你的模型训练事半功倍。我们的样本来源是视觉稿（Sketch），但同一个模块的 Sketch 视觉稿可能只有寥寥几张，可获取的样本数量过少。因此首先要解决量的问题。

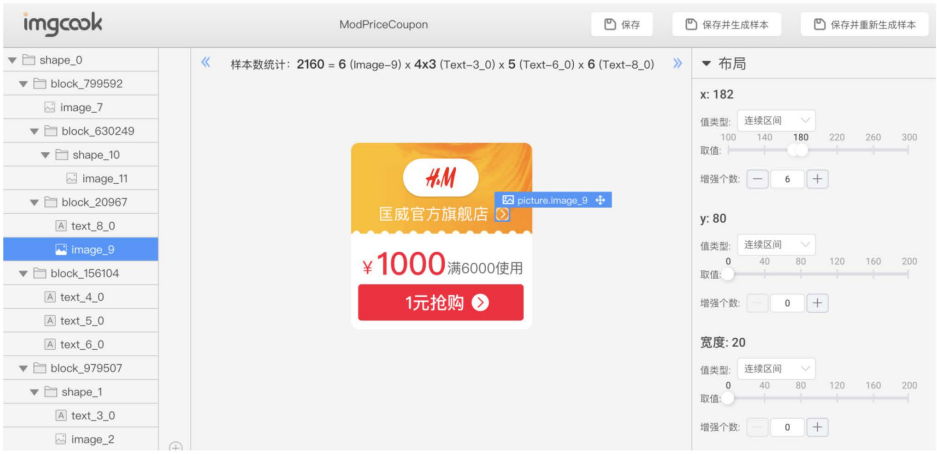
数据增强

为解决样本数量问题，我们采用了数据增强的方法。数据增强有一套默认的规则，同时也是可配置的。用户可自行根据视觉稿上各个元素在真实场景中可能发生的变化，如“是否可隐藏”，“文本字数可变范围”等维度来调整属性，产出自定义的配置项。因此样本制作者可以清晰的知道自己所造样本侧重的差异点在哪里。

我们根据这些配置项对属性进行发散、组合，生成大量不同的视觉稿 DSL。这些 DSL 之间随机而有规律地彼此相异，据此我们可以获得大数量的样本。

增强配置的界面如下图所示，左侧与中部是 DSL 树及渲染区域，右侧就是增强配置的区域。配置项由以下 2 部分组成：

- 增强属性：尺寸、位置、隐藏、前景背景色、内容
- 增强方式：连续范围、指定枚举值



（样本生成的界面）

特征提取

得到大量增强后的视觉 DSL 后，如何生成样本呢？首先明确我们所需的样本格式应该是表格型数据，以配合传统机器学习方法的输入格式：一条样本数据即一个特征向量。因此我们要对 DSL 进行特征提取。

基于此前的模型训练经验，我们发现某些视觉信息对于模块的类别判断尤为重要。因此我们对 UI 信息进行抽象，自定义并提取为特征维度，如 DSL 的宽、高、布局方向、包含图片数量、包含文本数量等。通过各种视觉信息的抽象，我们得到 40 多维的视觉特征。

除了视觉特征维度以外，我们还增加了自定义的业务特征。即根据一定的“业务规则”，将某些元素块定义为具有业务含义的元素，如“价格”、“人气”等，并抽象出 10 个维度的业务特征。在这一过程中同样支持用户自定义业务规则，可通过正则匹配等方式实现。

视觉抽象特征加上业务特征，组成一个特征向量。特征向量加上分类 label，即一个**样本**。

算法与模型

首先我们的输入是 Sketch 设计稿提取出的标准化 DSL，目标是认出该 DSL 是哪个业务模块，可以归结为一个多分类问题。沿着这一思路，前文我们从大量增强后的 DSL 中提取特征值、生成数据集以供训练。我们使用的多分类模型基于算法平台提供的各种组件进行搭建。

随机森林

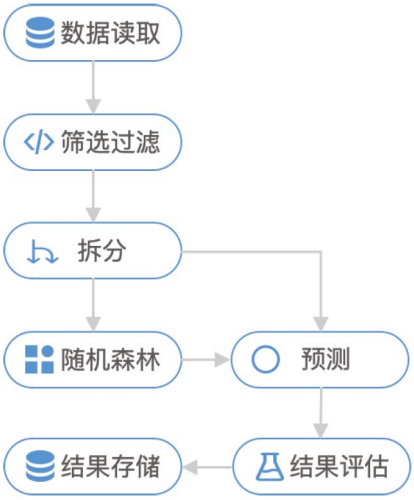
模型搭建

最初我们选择随机森林模型作为多分类模型，因为随机森林的执行速度快、自动化流程顺畅，几乎无需额外操作就满足了我们算法工程的需求；并且对特征值处理的要求较低，会自行处理连续和离散变量，规则如下表所示。

原始类型	解析类型
string / boolean / datetime	离散类型
double / bigint	连续类型

（随机森林变量类型自动解析规则）

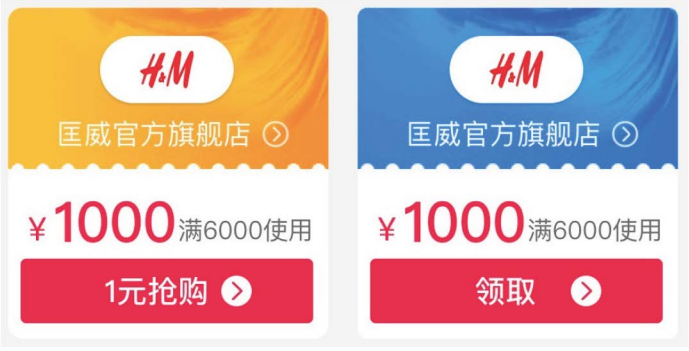
因此可以迅速的搭建出十分简洁的模型，如下图所示。



（线上使用的随机森林模型）

调参过程

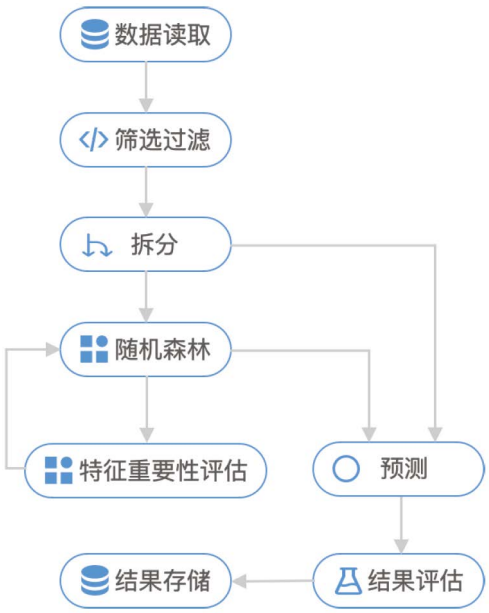
我们发现随机森林对于样本库内的数据，偶尔会有不自信的情况发生，即positivetrue 的置信度较低，被置信阈值卡住。尤其是视觉非常相似的样本，如图所示的两个相似模块就给我们的分类结果带来误差。



（相似模块）

为优化这种“不自信”的问题，我们对随机森林进行了调参，包括单棵树随机样

本数、单棵树最大深度、ID3/Cart/C4.5 树的种类配比等参数，也预接入特征选择组件，效果均不理想。最终在特征值重要性评估后手动反馈到特征选择并重新训练这一链路中取得了较好的结果，如下图所示。但这一过程无法融入到自动化训练流程中，最终被我们放弃。



（调参过程中使用过的随机森林模型）

离散特征问题

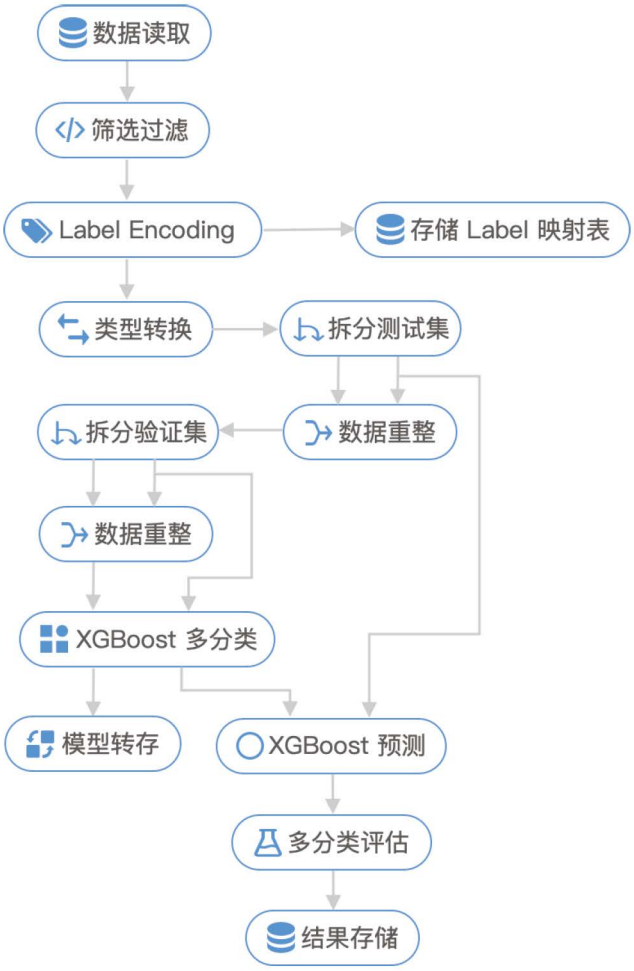
随机森林虽然可以自动处理离散变量，但是如果测试集中出现了训练集以外的离散值，算法无法处理这样的情况。要解决这一问题，需确保每个离散特征的全部取值都出现在训练集中。由于有多个离散特征，也无法通过简单的分层采样来解决。这也是随机森林模型应用中的痛点之一。

综上是我们在随机森林模型上做的工作，随机森林简单易上手、快速出结果，并且在大多数业务场景下都能满足识别需求，成为模块识别功能的 1.0 版本算法。但由于其算法缺陷，我们后来引入了另一种模型 XGBoost。

XGBoost 多分类

模型搭建

XGBoost 通过 Boosting 的方法提升树的“准确率”，相较于随机森林算法在我们的数据集上表现更优越。但是算法平台的 XGBoost 模型有许多流程不标准的地方，因此为了实现自动化链路，我们搭建了如图所示模型。



（XGBoost 模型）

预处理

XGBoost 模型需要更多的预处理方法来实现，包括：

- LabelEncoding：预处理过程。XGBoost 仅支持从 0 开始到（分类数 -1）的 label 数值。但为了映射方便，我们存储的 label 值对应的是平台的分类 ID，并不是 0 ~ N 的，甚至可能不是连续整数。因此需要用 LabelEncoding 组件编码到符合 XGBoost 需求的数值。
- 存储 Label 映射表：数据转存，因为预测接口会用到这一映射表来转义平台分类，因此要额外保存。
- 数据重整：预处理过程，为防止随机拆分算法将训练集的 label 拆分为不完备的数据集，把训练集 label 的缺失数据捞回来。对模型会有一定干扰，但是在数据极少的极端情况下才会发挥作用。

XGBoost 在测试数据上的表现颇为自信，降低了阈值划分的困难，预测结果也能够很好的满足我们“识别正确组件”的业务需求，并且也可以支持自动化流程，因此成为后续我们主推的传统训练模型。

难点问题：OutOfDistribution

值得一提的是，我们无法对当前模块库以外的所有视觉样本进行全面的收集，这样的工程就如同为了做一个阿里内部的面部识别系统，而去收集 70 亿人类的面部照片一样。样本库以外的数据缺失导致我们其实是少了一个隐藏的分类——负样本分类。也就引发了 Out-of-Distribution 问题，即样本库以外数据带来的预测失准问题，其本质是分类结果中 falsepositive 过多。

在我们的场景下，这是一个很难解决的问题，因为收集全部负样本的困难性。目前我们是如何应对这一问题的呢?

阈值设定

我们将分类模型输出的置信度 prob 作为确定分类结果的参考依据，高于某一

阈值则认为匹配到某个分类。这一方法具有经验意义，实践中有效的屏蔽了大部分 OOD 错误。

逻辑控制

对于算法模型的部分 OOD 误判，我们可以通过逻辑关系来辨别。如我们认为 DSL 树的同一条路径上不可能有多个相同组件（否则形成自嵌套），如果该路径上识别出多个相同组件，那么我们通过置信度大小来选择识别结果。此类逻辑帮我们筛选了大部分误判。

负样本录入

我们提供的反馈服务，允许用户将识别错误的 DSL 上传，上传后增强为一定数量的负样本并存储。在此基础上重新训练，可以解决 OOD 问题。

目前 OOD 问题还是依赖逻辑和反馈的方法来规避，算法层面仍然没有解决该问题，这是我们下一阶段计划去做的事。

模型部署

算法平台支持将模型部署为线上接口，即预测服务，通过 imgcook 平台可一键调用部署。为了实现自动化训练、部署的流程，我们还做了一系列算法工程的工作，在此不作详述。

预测与反馈

预测服务，输入为设计稿提取的 DSL（JSON），输出为业务模块信息，包括 ID、在设计稿上的位置等。

在调用算法平台的预测接口之前，我们加入了逻辑上的过滤，包括：

- 尺寸过滤：对于模块尺寸偏差较大的，不进入预测逻辑，直接认为不匹配
- 层级过滤：对于叶子节点（即纯文本、纯图片），我们不认为该节点具有业务含

义，因此也过滤不用。

结果反馈链路包括自动结果检测和用户手动反馈，目前仅提供了预测结果错误的样本上传功能。

我们的业务模块识别功能最终在 99 大促中首次在线上使用。上述的模型、前置逻辑、以及 OOD 规避等环节，最终带来的效果是：**业务场景内的识别准确率可达 100%**（纯模型的实际准确率未统计）。

未来工作

算法优化

难点问题解决

如前所述，OOD 问题是一个难点，目前仍没有很好的解决。针对这一问题我们有一些解决思路，计划在后续工作中进行尝试。

基于 DNN 的 lossfunction 优化：仍基于手动 UI 特征值搭建 DNN 网络，通过 lossfunction 的优化，扩大不同类别之间的距离、压缩同类别内部的距离，在优化后的模型上设定距离阈值来鉴别 OOD 数据。

负样本自动生成的优化：在 XGBoost 算法基础上，增加一个前置的二分类模型，用于区分集合内和集合外数据，并据此对负样本生成的随机范围进行优化。具体方案待调研。

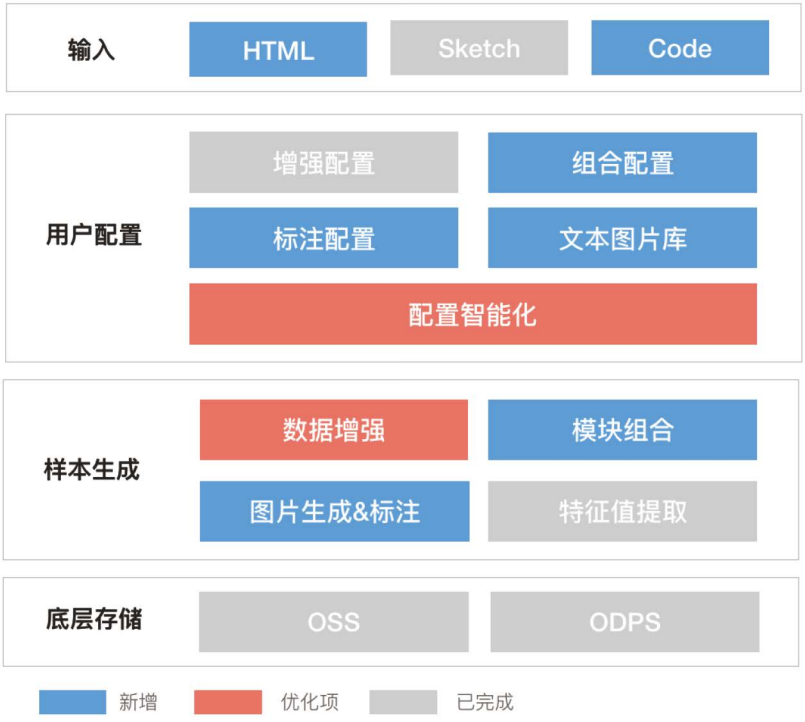
深度学习

手动特征提取的方法虽然快速有效，但是在泛化能力上无法与 CNN 之类的深度学习方法相比。因此后续我们会尝试基于图片的算法，使用 CNN 模型提取 UI 特征向量，再通过向量距离计算或二分类模型比对输入数据与各个 UI 组件的相似度。

在深度学习领域还可以有更多尝试，不限于以上算法设想。

样本平台

目前我们的样本生成功能存在配置效率低、支持算法类型少等问题，因此在后续工作中，我们计划将样本生成进行更丰富的产品化设计。样本平台的功能大致如图所示。



（样本平台产品功能）

来源扩展：目前我们的样本生成链路是从 Sketch 到 ODPS 表格数据，在后续的业务场景中我们还希望能支持从 HTML、前端代码生成样本。不论何种来源，在数据增强这一层都会有许多相通之处，我们将抽象出通用的增强算法服务，开放调用。

算法扩展：最终生成的样本，可以是特征值表格数据，用于多分类；也可以是 PASCAL、COCO 等格式的图片与标注数据，提供给目标检测模型使用。

增强智能化：目前用户在使用样本生成功能时感到配置复杂、难上手，甚至常因为误操作而导致样本不可用。因此我们期望能通过数据增强的“智能化”，来尽量减少用户操作，迅速生成有效样本。

综上，算法优化与样本平台产品化是我们下一期的核心工作。

布局算法篇

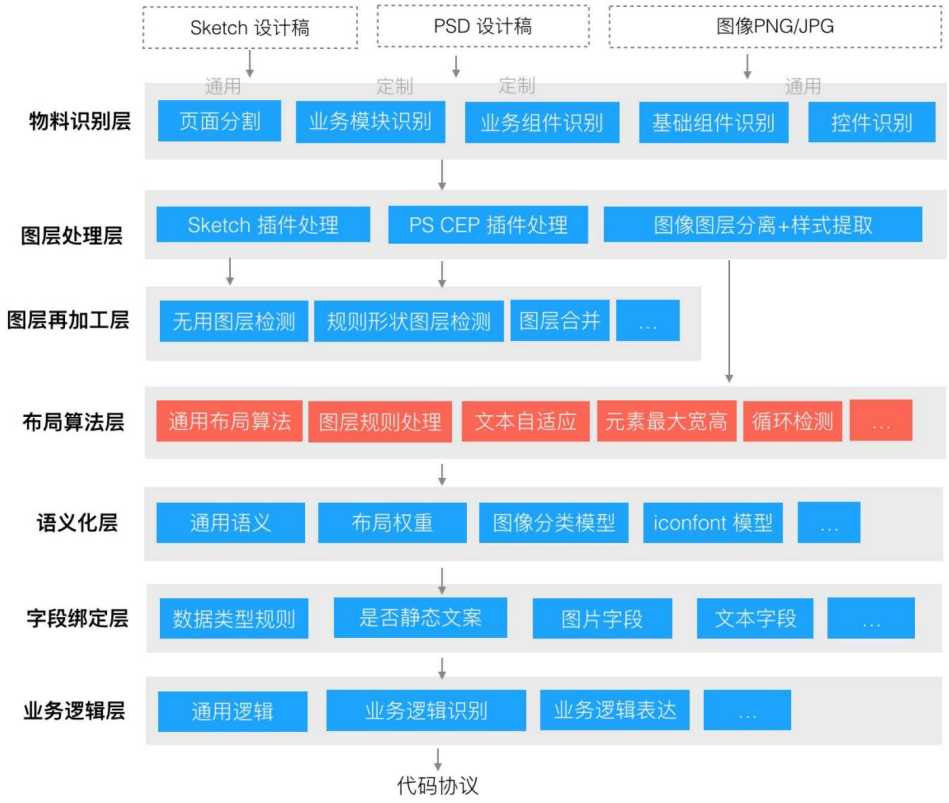
作为阿里经济体前端委员会四大技术方向之一，前端智能化项目经历了 2019 双十一的阶段性考验，交出了不错的答卷，天猫淘宝双十一会场新增模块 79.34% 的线上代码由前端智能化项目自动生成。在此期间研发小组经历了许多困难与思考，本次《前端代码是怎样智能生成的》系列分享，将与大家分享前端智能化项目中技术与思考的点点滴滴。

概述

在 D2C 中，设计稿（Sketch、PSD）、图片（JPG、PNG）经过算法处理之后，最终会导出以绝对布局为基础的元素信息。绝对定位的布局不具备扩展性，可读性也很差，对于开发者来说并不具有可维护性。因此，需要经由布局算法层来处理转化成“前端”眼中可用的代码。在布局算法处理之后，结合语义化、智能字段绑定、智能业务逻辑添加，最终为用户提供一份可读、可维护的代码，本文重点阐述布局算法层的相关处理过程。

所在分层

如图所示，经过链路中的物料识别和图层处理加工，会输出一份以绝对定位为基础，包含了元素位置信息和 CSS 属性的 JSON 数据。这份 JSON 数据接下来会进入布局算法层，对元素的布局结构以及 CSS 属性进行进一步的加工和处理，最终输出一份符合 D2CUI 图层协议规范的 JSON 数据。



(D2C 技术能力分层)

如图所示，经过链路中的物料识别和图层处理加工，会输出一份以绝对定位为基础，包含了元素位置信息和 CSS 属性的 JSON 数据。这份 JSON 数据接下来会进入布局算法层，对元素的布局结构以及 CSS 属性进行进一步的加工和处理，最终输出一份符合 D2CUI 图层协议规范的 JSON 数据。布局算法作为 D2C 链路中的重要一环，需要为下游输出正确的布局结构、正确的样式属性、元素间的关系。目前包含以下：

- 合理布局嵌套：
  - 绝对定位转相对定位
  - 合理的绝对定位

  - 冗余嵌套删除
  - 合理分组嵌套
  - 循环识别
- 元素自适应：
  - 元素本身扩展性：文本、图片等节点位置自适应，大小可扩展
  - 元素间对齐关系
  - 元素最大宽高容错性

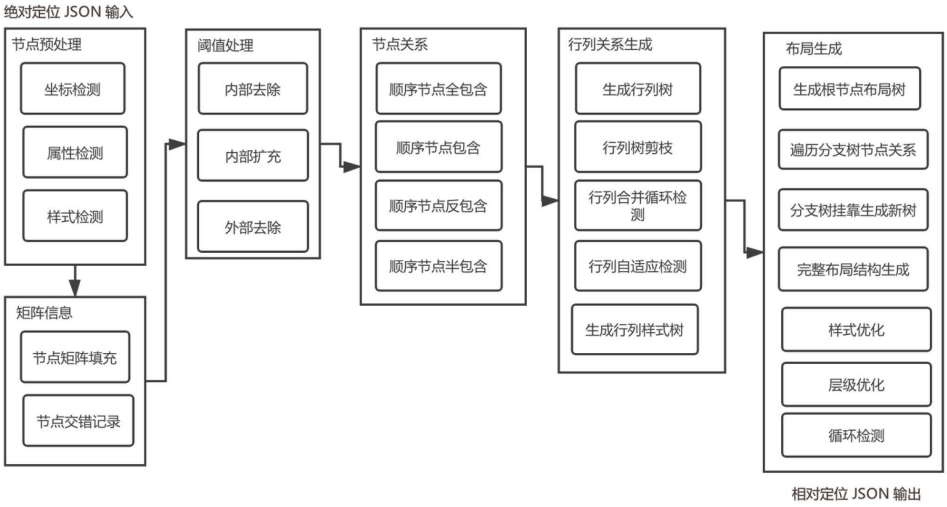
约束和前提

布局算法的目标是对任意视觉稿都能有良好的布局还原结果，但是在实际情况中，不够规范的视觉稿（图层分类混乱，图片大小不准确，元素位置不准确）都会影响布局的结果，因此正确的布局前提是有一份规范的视觉稿，这也是布局算法在现阶段，取得理想布局结果的一个前提。

此外，在真实使用的时候，虽然还没有编码，但是开发者早已洞悉视觉稿要怎么变成代码。但是由于规则的缺陷，或者视觉稿的不规范，生成的代码结果可能会与开发者心中所想有差别。因此我们需要提供能力让开发者可以干预生成的结果，而不是将整个过程包装在黑匣子中。这也是 imgcook 设计稿协议诞生的来由之一。通过这一套协议，开发者能够精确的控制代码生成的结果。换而言之，可以通过调整设计稿，结合设计稿的协议来达到对布局还原结果的控制，让生成的代码满足开发者的需要。

核心功能

布局算法的核心思路是分析布局中各个元素的位置、大小和类型，结合元素间的关系，将其从绝对定位的布局，转化为相对定位的布局。



（算法流程图）

如上图所示，布局算法的整个流程是顺序流式的，对于 JSON 数据处理的完整流程是：

1. 节点预处理
2. 矩阵识别
3. 阈值处理
4. 节点关系分析
5. 行列结构生成
6. 布局样式生成

输入

为了减少 Design 对还原的结果的干扰，如下图的设计稿，图层的组织结构和代码的 DOM 组织结构不同。在设计稿中，改变图层顺序，视觉结果可能都不会有变化，但是如果在代码中，调整了 DOM 元素位置，渲染结果可能会发生巨大的变化。因此，为了减少设计稿对还原的结果的干扰，我们选择将输入的设计稿的层级结构都去掉，让所有的图层变成一个一维结构，还原的时候不依赖设计稿提供的结构信息

（大部分情况下这些信息都是错误的）。

（设计稿示例图）

这带来的问题是，设计稿中的结构信息丢失。最终的 DOM 结构完全由布局算法生成，失去了人为干预的能力。开发者想要调整 DOM 结构，只能在布局算法生成之后。这会大大降低开发体验，提高使用成本。因此，我们设计了成组协议，通过在设计稿图层名称上添加 #group#，告诉布局算法，设计稿中这个结构是正确的，不需要重新计算。通过成组协议，能够满足开发者对于 DOM 结构干预的诉求。

节点预处理

布局的算法输入来源有 Sketch 插件，PS 插件以及图片，同时，随着插件的迭代升级，不同版本的插件导出的格式和内容也会有所不同。所以需要在进入布局算法之前，将这些差异所以抹平，确保布局算法处理的过程中不要为这些兼容性问题进行编码。

矩阵信息

从这一步开始，布局算法正式开始布局的计算。上面已经提到，输入的 JSON 里是不包含元素的结构信息。因此我们需要通过元素的位置和大小来分析出结构信息。

分析的第一步就是构造一个与输入大小一致的矩阵，例如输入是 702x370 的一个模块，那么会构造一个 702x370 大小的矩阵。遍历 JSON 中的所有节点，根据元素的位置和大小在矩阵中标记元素所在的位置。

这一步的主要工作就是填充整个像素矩阵，得到元素的交错记录。

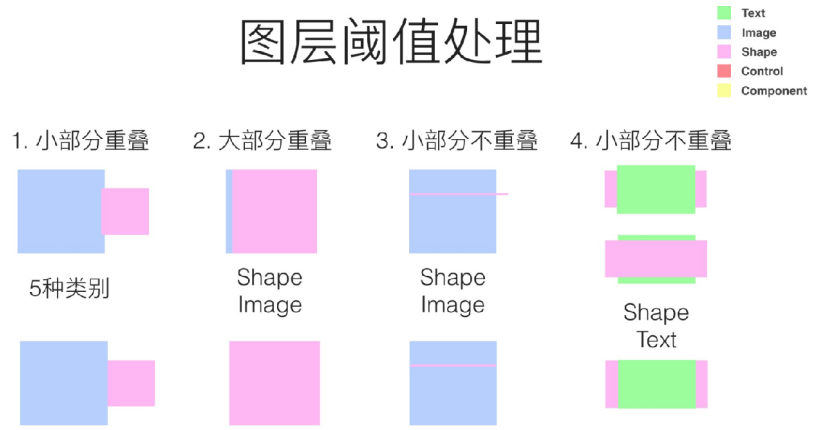
（图层信息转矩阵信息示意图）

如图所示，JSON 结构信息会转化成一个像素矩阵，像素点上会记录包含的元素信息。

阈值处理

设计稿设计的时候可能有误差，图层的位置可能可能会有偏移，肉眼去看的时候

不会很明显。但是在还原步骤中，分析像素矩阵的时候，是会对每一个像素都进行分析，所以像素级别的误差都会被记录进来，对结果可能会产生巨大的干扰。所以需要有一个步骤去规范图层，自动去修复这些误差，降低对还原结果的干扰。



（阈值处理）

如上图所示，元素间存在的误差需要我们去分析处理，将这些误差消除掉，为后续的处理提供一个可靠的结果。

节点关系分析

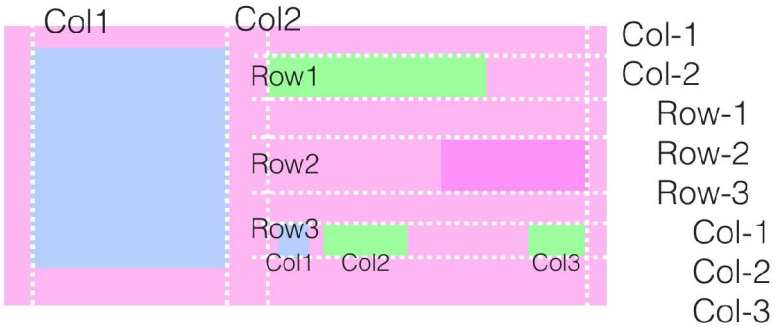
得到准确的矩阵信息之后，就可以来分析元素间的关系。根据元素间的位置信息，可以定义一下三种关系：包含、部分重叠、完全重叠。

（节点关系）

如图所示。A 包含 B 说明 B 节点可能是 A 节点的元素；A 和 B 局部重叠，说明 A 和 B 中至少有一个元素需要进行绝对定位。A 和 B 完全重叠的情况较为少见，但是也更为复杂，说明可能其中有一个元素是多余的，也有可能是一个节点包含另外一个节点，也有可能是有节点需要进行绝对定位。经过这个步骤，我们能够获得所有节点之间的关系，可以输出给下一步进行节点结构的构造。

行列结构生成

布局算法是通过“行列结构”来描述布局结构。“行列结构”记录了布局的行列信息，一行的信息包含有：行的起始位置，行的结束位置，行包含的元素，行里面包含的列。一列的信息包含有：列的起始位置，列的结束位置，列包含的元素，列里面包含的行。



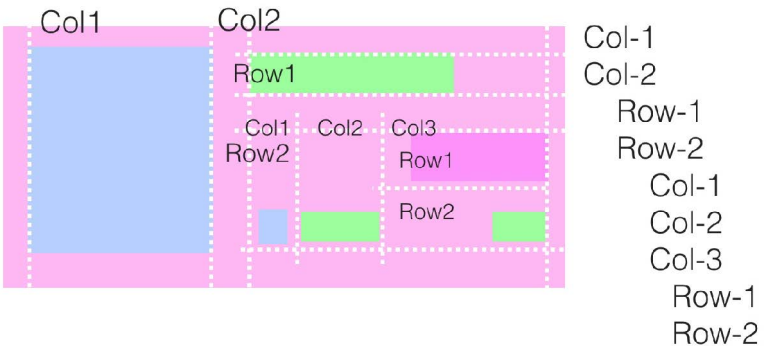
（行列结构示意图）

如图所示的一个模块结构，首先会被分成两列记为 Col1 和 Col2，可以看到 Col2 里面还包含了三行，可以记为 Row1，Row2 和 Row3。Row3 里面又包含了三列，可以记为 Col1、Col2、Col3。如图六右侧结构示意，最终生成从行列结构是一个树状的结构。

通过元素的行列结构信息，我们可以计算元素的布局信息。在图六的这个例子里，根节点包含了两个子节点 Col1 和 Col2，这两个节点在纵向可以通过 margin-

Top 来设置与根节点容器的边距，在横向可以通过 spaceBetween 来设置两个节点的位置。再进一步计算 Col2 中的 3 个 Row 的布局，通过递归的方式，计算所有节点的布局信息，建立完整的布局关系。

这个步骤是布局算法中最为关键的一步，但是行列结构生成的结果并不是唯一确定的。像图六这样的模块，还可以生成下图这样的行列结构。



（另外一种行列结构）

不同的行列结构描述代表了不同的 DOM 结构，从 UI 信息上，有时候并不足够判断哪种行列结构是最优的。所以引入了成组的协议来增加开发者的干预，确保能够生成开发者期望的结构。

需要注意的是，这里生成的 DOM 结构可能不是一棵树，而可能是多棵树，比如有一些 DOM 节点是绝对定位的，这部分节点会独立组成一棵树，而没有在主干上。

布局样式生成

经过上一步行列结构的生成，得到了多棵 DOM 树。接下来进进入了布局样式生成的步骤。这一步会遍历上一步得到的所有行列结构，将分散的多棵树组织成一棵完整的 DOM 树。得到完整的布局结构之后，节点的样式需要进一步优化。

- 宽高优化：上游给到的输入，所有的节点都会有宽高信息，但是经过布局算

法处理之后，部分节点可能不需要这个宽高属性，而是由子节点将容器撑开即可。

- 样式精简：删除插件导出的多余属性
- 属性处理：插件中的 transform 是针对单个节点，但是布局之后的 transform 对子节点也会生效，所以要对 transform 属性进行处理
- 扩展性优化：有一些元素是可扩展的，例如【图三】的牛皮癣节点，需要进行样式的优化，如果设置了容器宽度需要转化为 padding 形式的布局
- 层级优化：对生成的 DOM 树进行层级优化，删除一些多余的层级

经过上述处理之后，就得到了完整的 DOM 树，布局算法的基本任务也已经完成。生成 DOM 树最后，可以进行进一步分析，例如可以对 DOM 树的节点相似度进行检测，标记可能是循环的节点。检测出循环之后，在生成代码的时候节点就可以通过循环的方式渲染。

测试和度量

由于布局算法目前还偏向专家规则系统，深度学习等智能化方案应用较少，所以规则分支较多，改动容易引起关联问题，所以布局算法层对功能的稳定性和效果的度量有非常高的要求。

功能稳定保障

单元测试

布局算法许多函数的输入非常复杂，像是矩阵信息，可能就是一个 702*458 这样的一个矩阵，构造这样的输入很困难。于是我们另辟蹊径，不直接构造函数的输入，而是在设计稿中构造不同的模块，以一个完整模块的运行作为测试用例。

不仅输入的构造较为困难，输出的对比也不太容易。假如要分析矩阵信息这一步骤构造的矩阵是否正确，我们的结果需要录入 702*458 这样的结果。虽然确实可以

通过这种方式测试，但是我们设计了另外一种方法，将多个函数联在一起，最终输出的是一个对象，这样测试结果的可读性也大大增强，同时也能保证的正确性和测试的有效性。

目前单元测试的代码覆盖率达到了 78%，核心功能达到了 100%。

功能测试

除了单元测试外，在布局算法的场景下还可以通过功能测试来保障算法的正确性。精选了线上的 156 个模块作为测试集，包含了无线，pc 等各种场景下的模块，运行布局算法，分析算法运行的结果。在每一次上线前，都会运行一轮功能测试，以保证布局算法功能的正确性。

效果度量

测试上保证的是布局算法对不对，有没有报错。但是对于结果的“好不好”，需要通过另外一套体系来度量。这套度量体系包含了三个方面：UI 还原度量，代码可维护性度量，用户真实可用率度量。

UI 还原度量

UI 还原度量是度量经过布局算法后视觉效果上是否 100% 还原，具体是通过对比模块的视觉图和 demo 截图，分析布局算法是否准确地还原了模块。

（UI 还原度量）

布局还原效果度量方案复用了插件的还原度量方案，度量插件的还原度时，使用的是插件导出的 JSON 数据，而度量布局算法的时候使用的是还原的结果。还原精选了线上 160 个模块，覆盖了多种形式的布局，经过还原对比，确保布局算法的准确性。不过 ui 的度量只能保证说渲染出来的 UI 和视觉稿一致，但是不能确保结构的正确，假如没有经过布局算法的加工，直接是插件导出的数据，可能对比设计稿还原度会更高，因此需要其他的方案来进一步度量还原的结果。

代码可维护性度量

可维护性的度量，主要是考虑度量代码本身的质量。代码的质量可以通过结构的合理性来衡量：嵌套不能过深，节点数不能太多，不能有多余的嵌套。从生成的节点结构来分析布局算法的可维护性，具体度量的方法：

- 嵌套
  - 最大嵌套深度不超过 6 层
  - 冗余嵌套（1 套 1）
- 子节点数
  - 不能超过 6 个

虽然方法较为简单，但是从结果看能够很好的描述生成的结构是否冗余。但是，这种程度的度量也还不足以描述生成的代码足够好，因此我们还引入了用户真实可用率，来进行共同度量。

用户真实可用率度量

如何判断用户的认可度就比较难，如果直接进行用户调研，会有一定的主观因素，也难以量化。我们想到的方法是，计算代码的改动比例，如果生成的代码得到了用户的认可，满足了用户的需求，那么他就不应该去改动代码；反之，如果生成的代码不好，存在问题，用户拿去使用的时候一定会去修改。用户保留下来的代码的比例可以一定程度上用来衡量用户的认可度，从而反应了还原的效果。

度量的计算方案：

- 代码可用率：生成的代码最后留存行数 / 发布阶段的总代码行数

双十一模块的度量结果

此次双十一，双 11 新增模块总数 38，D2C 链路产出 30 个，占比：78.9%，智能生成代码平均可用率：79.34%，视图还原准确度：92.47%。未来重要努力目标之一，就是不断的提高视图还原准确率，提高代码可用率。

未来展望

目前，经过规则协议的推广，我们得到了大量用户标记了成组和合并协议的视觉稿，通过对比去掉协议之后的结构生成结果，可以分析出算法生成的结构与用户期望的结构的差别。借助这些高质量的样本，我们正在通过机器学习的算法，进行智能的成组和合并，进一步的降低用户使用的门槛和成本，解放生产力。另外，补充目前整在努力的一些方向：

1. 布局算法当前只支持 flex 布局，但是中后台场景或者其他场景可能需要不同的布局方案，因此布局算法后续会提供布局的定制，可以从 flex 布局切换到其他布局。
2. 提高布局的准确度，目前有些场景下，布局结果和业务实际场景有关，布局还原的效果不理想。接下来会提供辅助的工具，在插件上引导设计师规范视觉稿，在 web 编辑器上给开发者更好的编辑体验。
3. 降低开发者的使用成本，目前对于设计师，前端，在使用过程中还有调整设计稿的要求，这也提高了 D2C 的使用成本。我们正在努力通过智能化的手段来降低这些成本，让生成代码更加智能，更加简单。

语义化篇

作为阿里经济体前端委员会四大技术方向之一，前端智能化项目经历了 2019 双十一的阶段性考验，交出了不错的答卷，天猫淘宝双十一会场新增模块 79.34% 的线上代码由前端智能化项目自动生成。在此期间研发小组经历了许多困难与思考，本次《前端代码是怎样智能生成的》系列分享，将与大家分享前端智能化项目中技术与思考的点点滴滴。

概述

imgcook 是以各种设计稿图像（Sketch/PSD/ 静态图片）为原材料烹饪的匠心大厨，通过智能化手段将各种原始设计稿一键生成可维护的 UI 试图代码和逻辑代码。

语义化是什么?

语义主要研究符号标记，以及他们所代表的事物之间的关系，语言学中主要研究符号所表达的意思，而在 web 前端开发领域中，**语义化**指编写 HTML 的过程中“用最恰当语义的 html 标签和 class 类名等内容，让页面具有良好的结构与含义，从而让人和机器都能快速理解网页内容”。语义化的 web 页面一方面可以让机器（搜索引擎、爬虫、屏幕阅读器）在更少的人类干预下收集并研究网页的信息，从而可以读懂网页内容，收集汇总信息进行分析；另一方面它可以让开发人员读懂结构，快速理解页面各区块功能，便于二次维护。简单来说就是利于 SEO，便于阅读维护理解。

传统语义化包含了 html 标签的语义和 class 类名的语义。

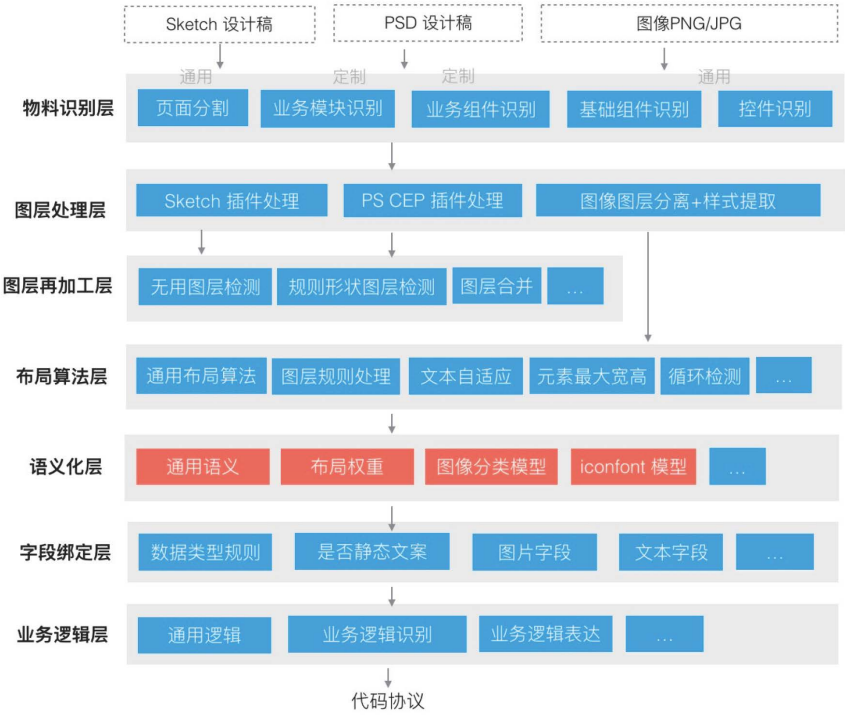
先说 html 标签语义，在 HTML5 出现之前，我们只能用没有语义的 div 来表示页面章节。现在，通过使用 HTML5 语义元素标签，如 body、article、nav、aside、section、header、footer、hgroup 等，我们可以通过读取 html 结构就了

解页面的布局结构。在 react-native、rax 等跨端 UI 体系下，标签通常被各种组件替代，标签语义化也就转换成了“组件语义化”，在合适的场景下使用合适的组件名即可实现结构可读。

class 类名语义化指用易于理解的名称对 html 标签附加的 class 或 id 命名。class 属性本意用来描述元素样式内容的，经过前端领域多年的演变，已经不仅局限于做 HTML 和 CSS 的衔接，而是一个集样式定义、函数钩子、组件类型等多层意义的复杂属性。本文将专注于 class 类名语义化这个问题，尝试运用 D2C 的能力彻底解决。

所在分层

在整体架构中，语义化层负责对布局算法生成的视图进行语义推测，用较为人性化的类名替换初始值，从而达到接近前端自己命名的效果。



(D2C 技术能力分层)

制定 class 类名命名原则

业界规范

BEM（Block，Element，Modifier）

BEM 是块（block）、元素（element）、修饰符（modifier）的简写，由 Yandex 团队提出的一种前端 CSS 命名方法论。用中划线、双下划线、单下划线来做单词间的链接记号，通过将页面分解为一个个小小的可重复使用组件来解决复杂项目的命名问题。比如：.block{}、.block__element{}、.block--modifier{}，都是典型的 BEM 命名模式，他们的命名规范、可读性高，通过组件的修饰符就可以了解组件的形态。

NEC（niceeasycss）

国内的网易团队指定的前端样式规范。通过指定的单字母前端来做功能划分，大体上有以下功能：重置和默认（reset+base）、布局（g-）、模块（m-）、元件（u-）、功能（f-）、皮肤（s-）、状态（z-）、js 钩子（j-）等。基于上述单字母命名，再使用简约而不失语义的后代选择器名称追加其后。典型的 NEC 命名如 m-list、u-btn-hover 等。

AliceUi

用扁平化的方式划分为不同层级。基于 - 符号做命名空间隔离，第一个前缀通常是通用业务标识，各业务线选取自己的前缀，后面依次用组件名、组件状态等填充。组件名必选，且要求表意，模块内部类名需继承上层名称。ui-name-status、ui-page-item-info 都是典型的 AliceUi 命名方式。

业界的 class 类名规范的目的都是解决大规模项目下的样式命名问题，且因为遵循了各种层级结构关系和私有约定，编写出的类名普遍较长、在不了解规范的人眼中需要有一定的适应过程。不是特别适合 D2C 主打的移动端轻量级的模块开发场景。

从实际场景推导

为了寻求适合的场景，我们分析了内部的前端工程师在导购开发业务下真实代码的样式命名。下图是我们对淘系导购页面的真实词频统计结果，左图是样式全名词频，右图是拆分之后的原子结果：

样式词频分析

```
{
  "style": {
    "count": 1793,
    "detail": [
      " title              106  ",
      " container          84   ",
      " desc               48   ",
      " pic                45   ",
      " wrap               41   ",
      " name               33   ",
      " text               33   ",
      " icon               29   ",
      " content            29   ",
      " root               28   ",
      " avatar             27   ",
      " emptyCtn           26   ",
      " titleWrap          25   ",
      " zanIcon            24   ",
      " footerText         24   ",
      " tipsImg            24   ",
      " tipsDesc           23   ",
      " emptyProduct       23   ",
      " summary            20   ",
      " item               19   ",
      " tipsBtn            19   ",
      " tipsBtnText        19   ",
      " navbar             19   ",
      " mask               18   ",
      " tag                18   ",
      " commonName         18   ",
      " refreshText        18   ",
      " info               18   ",
```

样式词频分析（原子）

```
},
"word": {
  "count": 686,
  "detail": [
    " wrap               469  ",
    " text               388  ",
    " title              336  ",
    " ctn                304  ",
    " item               247  ",
    " product            237  ",
    " icon               209  ",
    " empty              186  ",
    " img                184  ",
    " pic                167  ",
    " name               154  ",
    " content            132  ",
    " desc               120  ",
    " container          114  ",
    " tips               112  ",
    " tag                109  ",
    " price              96   ",
    " bg                 95   ",
    " card               91   ",
    " btn                90   ",
    " footer             89   ",
    " info               88   ",
    " banner             87   ",
    " bar                81   ",
    " view               80   ",
    " mask               79   ",
    " header             76   ",
```

（前端样式词频分析）

图中样式命名有如下特点：

- 和 BEM、NEC 等规范的风格不同，实际开发场景中的命名相对简洁
- 准确表达语义，且和节点业务特征强关联
- 单词为主，复合词采用驼峰命名，长度通常不超过 3
- 辅助性的修饰词如：wrap、ctn、empty、desc 等高频出现
- 可使用通俗易懂的简写单词

制定命名原则

D2C 希望优先解决淘系移动业务的问题，总体上以实际场景为主，业界规范为辅，最终确定了以表意为主要原则的样式命名策略：

- **表意**：样式名需准确表达节点意义，选择上优先从移动端业务真实类名中获取
- **简洁**：以单个词为主，所有词都使用不超过三个原子词组合
- **规范**：以驼峰为主，同时在代码转化层面保留了转化为连字符的能力
- **工整**：从模块根节点开始，采用布局信息 + 区块语义 + 语义辅助的整体命名策略

类名策略树

命名原则确定后，我们相当于定义了一个树的最终叶子节点形态，接下来需要构建从枝干一点点按图索骥到这些最终节点的过程，最终构建出我们的类名策略树。

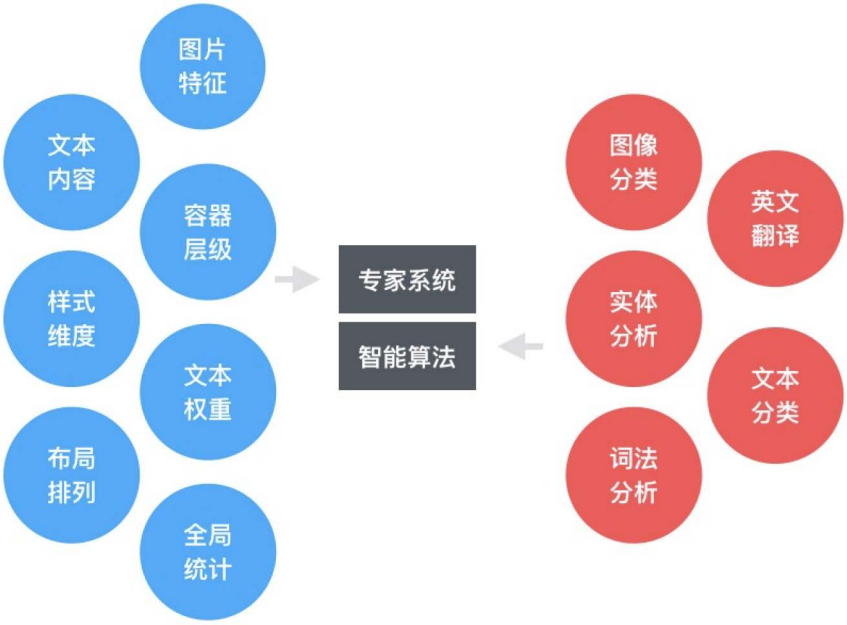
判别维度

在实际对节点类名命名的过程中，我们要考虑的规则往往是多个维度的：

（样式判别维度）

imgcook 对淘系多达几百的模块进行规则提取，根据真实书写习惯，将上述规则做了权重，一般来说我们希望功能类别优先于样式特征，即一个按钮播放器按钮命名为 playBtn 而不是 circleIcon。其他的规则作为辅助决策，在整个树中左右走向。

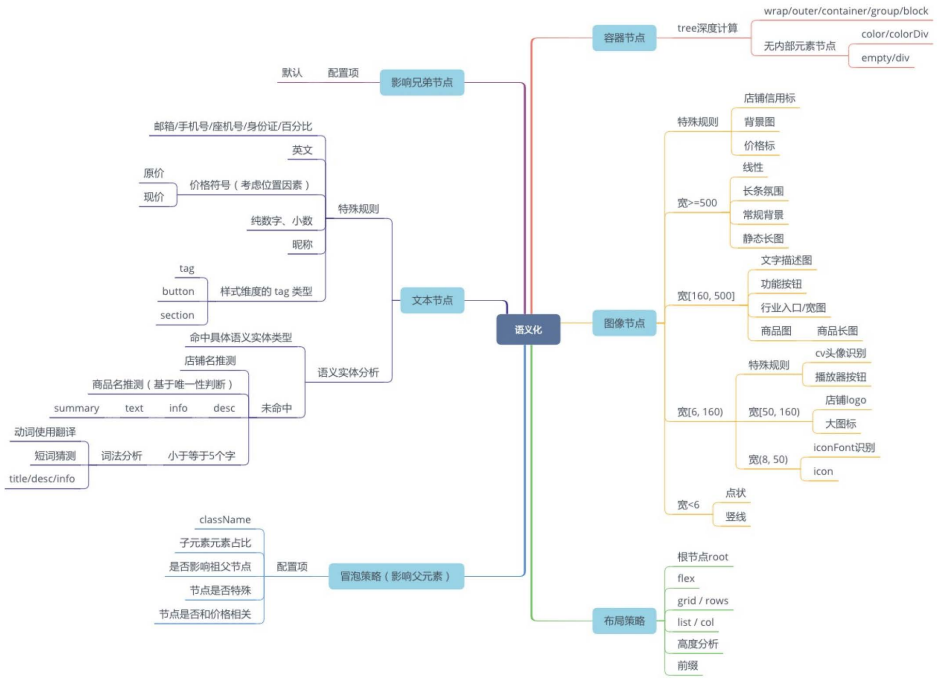
基于节点样式、内容、层级、特征、权重、布局、全局计数对组件节点做了多方位多种类型的鉴别。同时借助阿里内部 sqi 平台和 D2C 自身的智能化能力，实现对一些经验规则解决不了的节点类名的鉴别。



（D2C 样式命名选型）

策略定型

在建设完成阿里内部业务专属的类名专家系统后，结合智能化算法，我们升级了策略判别的流程，并整合出了下面这个最终的策略树。从根节点出发，大部分节点都可以通过此策略树归纳到一个具体语义结果上。

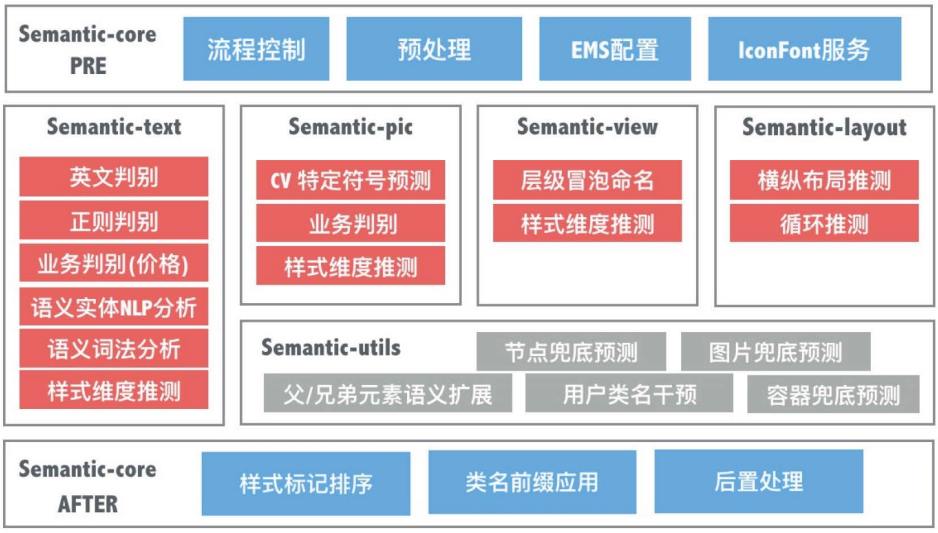


（D2C 样式策略树）

类名识别服务

核心实现

在实现上述样式命名策略树的过程中，我们产出了一个专用于推测 imgcook 模块布局类名的服务：



（语义化 SemanticService 结构图）

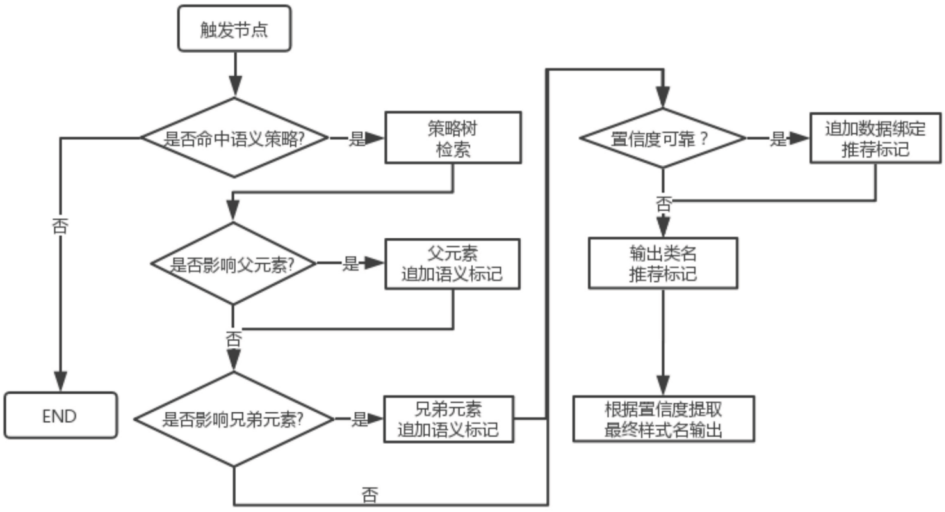
semantic-core 提供整体的节点树遍历流程控制，分为前置和后置两个处理过程。

预处理过程会向组件节点追加检索索引，同时会检索组件树中符合条件、需要调用 iconFont 识别服务的图片统一聚合发送请求。后置处理中会对各个语义项处理的结果标记进行排序、应用前缀类名、执行组件索引清理等。

semnatic-text、semantic-pic、semantic-view、semantic-layout 是 imgcook 内置的语义算法。分别分析文字、图片、容器和布局相关的信息。

每个语义项执行过程如下：

- 判断是否命中语义策略，未命中则结束此语义执行（语义策略下面会有详解）
- 判断是否会影响父元素，是则检索父元素，追加当前语义项的标记
- 判断是否会影响兄弟元素，是则检索兄弟元素，追加当前语义项的标记
- 判断当前命中的语义策略置信度是否可靠，是则为当前节点追加数据推荐标记
- 最终，一个节点会得到很多不同置信度的语义项标记结果。通过统一筛选，得到此节点最后生效的类名

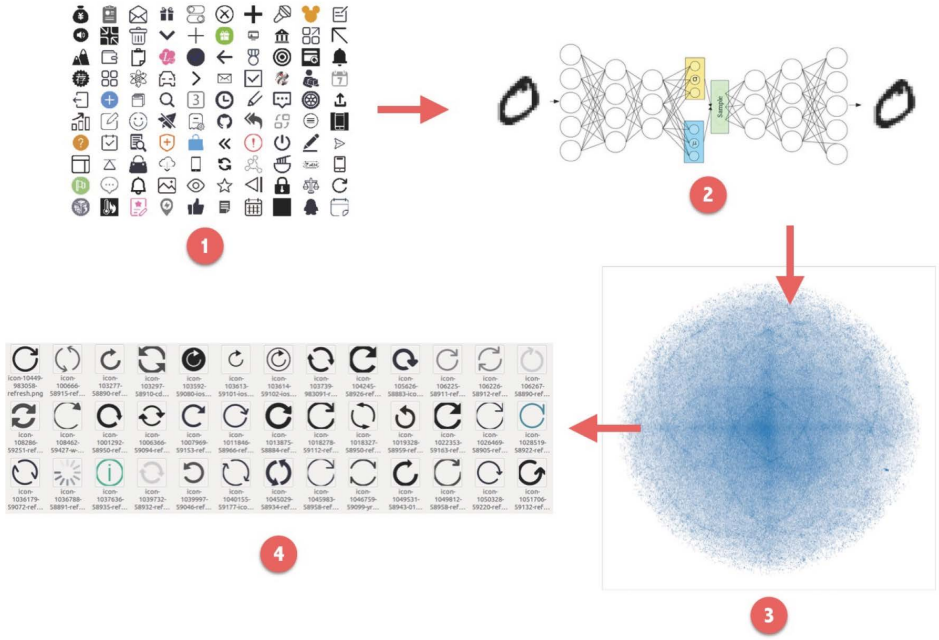


（语义项执行流程）

细节实现

在向各个策略叶子节点的推导过程中，我们会使用最适合的能力实现需求，比如为了解决“鉴别小图标”这个语义判别过程中我们部署了 IconFont 服务来实现，具体流程如下：

1. 从 iconfont 网站上获取 iconfont 的图标文件，并转成 png 格式，如下图
2. 使用一个自编码器把图片编码到特征空间
3. 使用 t-SNE 映射到二维平面上看看效果
4. 在特征空间上使用聚类算法将类似的 icon 聚到一起
5. 手工剔除质量较差的版本，然后将一个类簇中的命名根据已有名称进行选举



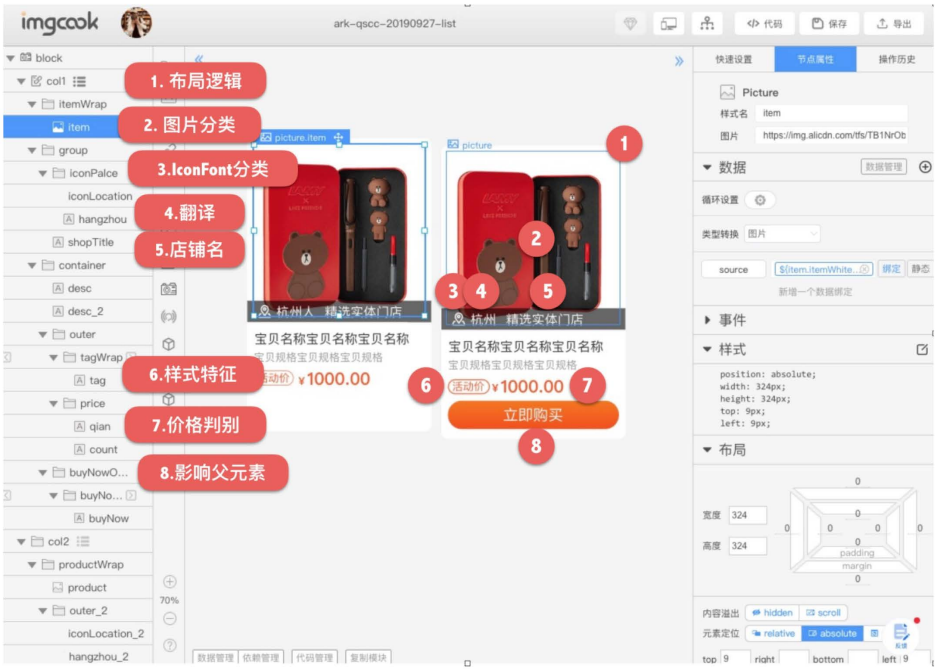
（IconFont 识别服务执行流程）

iconFont 服务用于解决小图标命名问题，至今仍然在持续优化中。

展望未来

典型应用

imgcook 语义化能力自从方案上线以来已支持了一年多的线上业务。以下是 2019 年双 11 的模块仓库中找到的，可验收语义化成果的一个模块：



（语义识别结果）

从该模块的还原效果中可见，依次命中了布局逻辑、图片分类、IconFont 分类、翻译、店铺名、样式特征、价格判断等内置策略。大部分节点都有语义项命中，其中识别较为准确且贴近语义的节点占比 60%+。

衍生方向

语义化本质是推测节点特征的过程，在识别准确度没有要求时可以作为 class 名使用。对于识别准确度极高的预测结果，我们认为节点和数据也有映射成功的可能性。

因为我们希望 D2C 能推测出节点绑定字段并实现在业务中落地，所以基于语义化的思路，D2C 孵化出了目的性更强、对准确度要求更高的节点数据字段推测服务。

感兴趣的同学可以移步本系列的“字段绑定”文章继续深入了解。

字段绑定篇

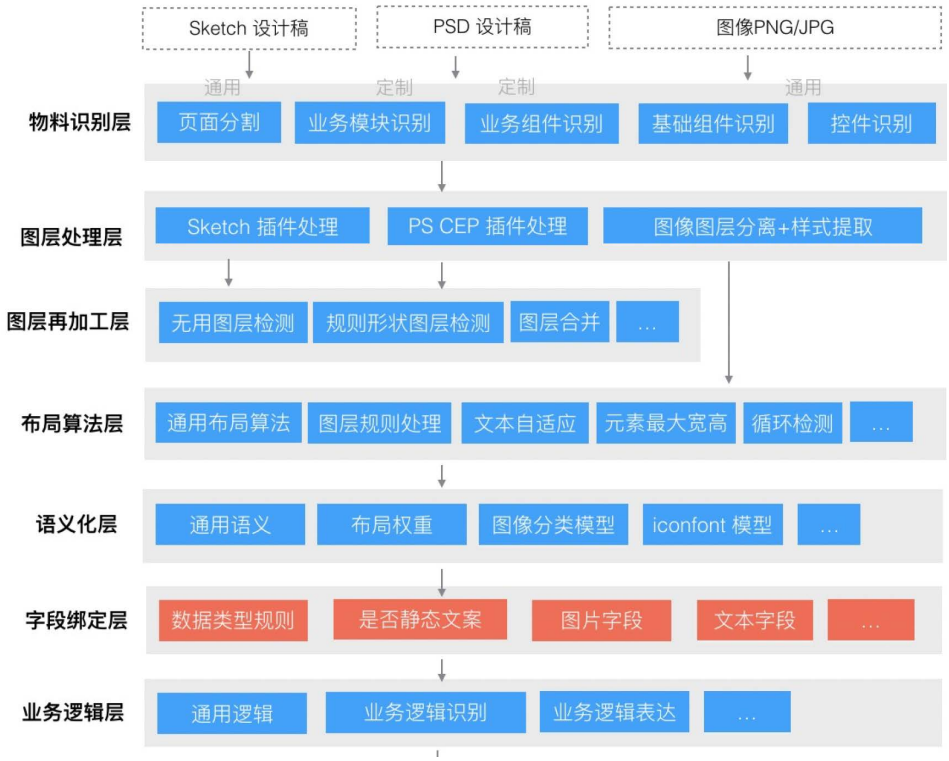
作为阿里经济体前端委员会四大技术方向之一，前端智能化项目经历了 2019 双十一的阶段性考验，交出了不错的答卷，天猫淘宝双十一会场新增模块 79.34% 的线上代码由前端智能化项目自动生成。在此期间研发小组经历了许多困难与思考，本次《前端代码是怎样智能生成的》系列分享，将与大家分享前端智能化项目中技术与思考的点点滴滴。

概述

imgcook 是专注以各种图像（Sketch/PSD/ 静态图片）为原材料烹饪的匠心大厨，通过智能化手段将各种视觉稿一键生成可维护的前端代码，含视图代码、数据字段绑定、组件代码、部分业务逻辑代码。**智能字段绑定是其中一部分，实现准确识别营销以及其他垂直业务视觉稿的可绑定数据字段，大幅提升模块研发效率，从而强化视觉稿还原结果**，其拆分为数据类型规则、是否静态文案、图片绑定和文本字段绑定几个部分。

在 imgcook 大图位置

如图所示，服务位于字段绑定层，包括数据类型规则、是否静态文案、图片字段，文本字段等部分。



(D2C 技术能力分层)

预研

智能字段绑定依赖于语义化层，语义会依据经验为节点做类型标注，标记了这个节点“是什么”后，智能绑定将这个“什么”转为业务域中字段，为了提高准确度，将高置信度规则作为绑定条件。分析如下：

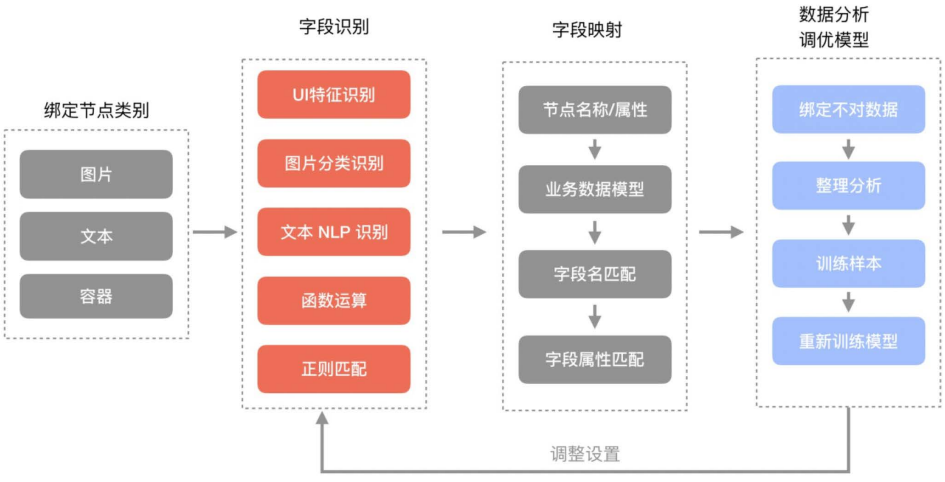
- 语义与字段绑定联系过深
  - 问题：导致的问题语义与字段绑定结果关联过紧，灵活度不够
  - 优化：语义与字段绑定的判定流程分离，移除置信度概念
- 语义层规则层过硬
  - 问题：现有规则层过硬，以判断为主

- 优化：硬规则使用分类算法，统一定性对标到 w3c 的节点标准上

- 语义层机器学习算法使用程度不足
  - 问题：仅使用了实体识别、句法分析和翻译
  - 优化：图像分类使用深度模型，文字分类使用传统机器学习
- 业务域字段频繁变更
  - 问题：映射的字段不同业务域下不同
  - 优化：提供不同配置自身智能绑定映射关系的能力
- 硬规则层扩展
  - 问题：现有规则不够多
  - 优化：根据设计稿，提炼新的规则，扩充规则层

技术方案

字段绑定主要通过文本 NLP 识别和图片分类识别 vdom 中内容来决策映射到数据模型中的字段从而实现智能绑定字段，整体流程入下图



（字段绑定核心流程图）

在字段绑定中主要核心是文本 NLP 识别和图片分类模型，下面着重介绍下。

文本 NLP 识别

调研

淘系设计稿文字所有动态部分分类分析：

业务域下常见的字段和设计稿文字的关系，下面举几个例子

商品名称 / 标题 (itemTitle)

- 设计稿文字：产品产品名称最多十二个字、产品名称十二个字、商品名超过十个字显示
- 真实意图文字：NikeAF1JESTERXX、海蓝之谜睡眠面膜保湿补水神器饱满焕发、Galanz/ 格兰仕 G80F23CN3XL、德国双立人 Specials30cm 中立炒锅套装、SOPOR/ 苏泊尔 CFXB40FC8028

(商品名称 / 标题设计稿)

店铺名称 (shopName)

- 设计稿文字：店铺名最多八个字、店铺名称店铺、店铺名可以八个字、店铺名称店铺名称最多放十五个字
- 真实意图文字：优衣库体验舒适人生、NIKE 海淘精品、匡威官方旗舰店、ZARA 服饰旗舰店、Mays 官方海外旗舰店

（店铺名称设计稿）

店铺利益点（shopDesc）

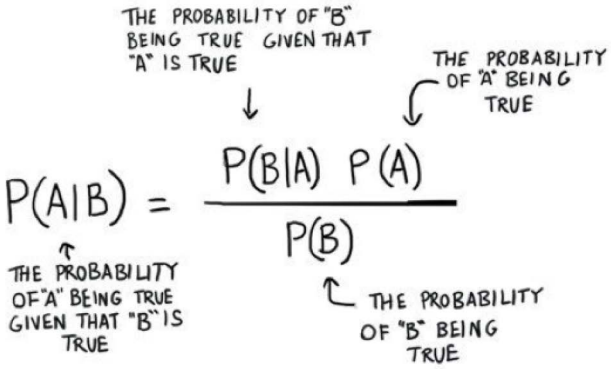
- 设计稿文字：店铺利益点八字内、利益点超过十个字显示、利益点仅七个字、利益点可以两条利益点文案最多十个字
- 真实意图文字：满 199 返 199、进店可享满 5 折优惠、进店可享满 199 送 199、冬新品第二件满减包邮、全场满 1999 送 199

（店铺利益点设计稿）

技术选型

朴素贝叶斯

我们在字段绑定中进行 NLP 识别的一个问题是样本量不够。尤其我们依赖于租户上传自己的样本对他们特定的业务进行训练，往往租户并没有特别大量的数据，在这种情况下，我们考虑选择朴素本叶斯分类器来进行分类，原因是朴素贝叶斯公式源于古典数学，其对于后验概率的得出源于先验概率和调整因子，并不依赖于数据量，对小数据和噪声点非常健壮。核心算法是这个贝叶斯公式：



换个形式表达就清楚多了，如下：

$$P(\text{类别}|\text{特征}) = \frac{P(\text{特征}|\text{类别})P(\text{类别})}{P(\text{特征})}$$

最终求 P(类别 | 特征) 即可。

分词

对于每条样本，我们在分类前需要首先进行特征点提取，在这个问题中也就是对样本进行分词，在机器学习平台中默认使用了 AliWS 来进行分词，AliWS(Alibaba-WordSegmenter 的简称) 是服务于整个阿里巴巴集团的词法分析系统。被广泛应用于集团的各项业务中。AliWS 的主要主要功能包括：歧义切分，多粒度切分，命名实体识别，词性标注，语义标注，支持用户自己维护用户词典，支持用户干预或纠正分词错误。其中，在我们的项目中，命名实体识别包括：简单实体，电话号码，时间，日期等。

模型搭建

我们主要是使用了机器学习平台的能力进行了快速模型链路的搭建，机器学习平台对于 ALiWS 的分词算法和朴素贝叶斯多分类进行了很好的组件封装，下图是我们的模型搭建

（文本 NLP 模型训练链路）

从上图看第一步会执行 SQL 脚本从数据库拉取训练样本，然后对样本进行分词操作处理。之后会按找一定比例将样本拆分为训练集和测试集，对于训练集的样本，进行朴素贝叶斯分类器，对于测试集，则是拿到分类器的结果进行预测和评估，最后会把模型结果通过 odpscmd 指令上传存储到 oss。

图片分类模型

调研

从业务场景中，我们总结出常用的八种分类，分别是标、icon、头像、店铺 logo、场景图、白底图、氛围图、高斯模糊图。

- 标 :label 为矩形小图，背景纯色，上有白色短文字点名心智
- icon:icon 大概率圆形。通常是抽象化的符号，
- 头像 :avator 通常圆形，中心为人物面部
- 店铺 logo:logo 通常用于突出性展示一个概念，文字或抽象化的图片作为主体
- 场景图 :picture 最常见的商品图，内容较多，主体通常不为一个。突出商品或人物在真实环境中的表现。
- 白底图 :purePicture 主体单一，背景纯白。突出主体自身。
- 氛围图 :pureBackground 有明显色系，形状构成的背景图
- 高斯模糊 :blurBackground 高斯模糊效果的背景图

在之前的模式下，我们主要是根据图片大小和图片位置等相关信息通过一些规则来进行图片识别。但这种模式下的识别存在不准和不灵活的问题，比如不同业务下可能 icon 的大小不尽相同，以及位置等信息存在极大的不确定性，同时由于基于这些类别进行分析，发现图片本身的内容已经足够区分开来，所以我们考虑使用深度学习

模型进行图片分类识别。

技术选型

CNN 网络

图片分类问题，我们首选是当前图像处理最热门的 CNN 网络，卷积神经网络的想法来源于**人类的视觉原理**，而这种通过卷积核分析图片相较于传统的神经网络极大的降低了待训练参数数量。同时，相较于传统的机器学习模型，CNN 在特征提取上表现出了极高的优势。

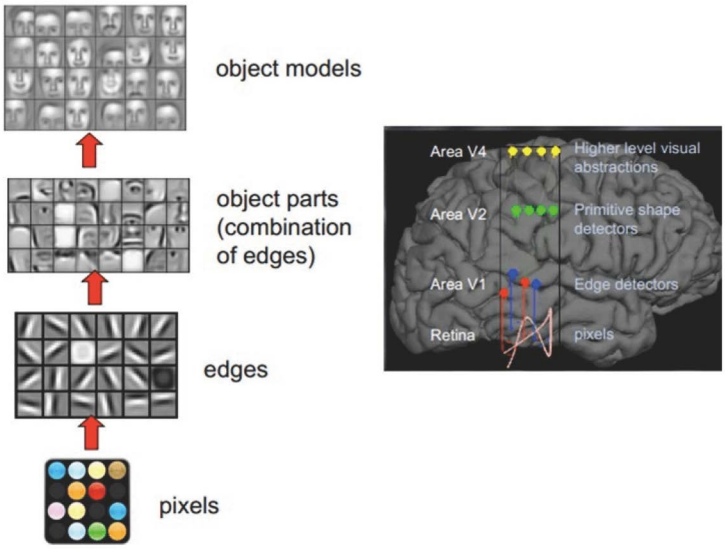
简单介绍下 CNN 网络如何实现的，在介绍卷积神经网络络前，我们先看看人类的视觉原理

人类的视觉原理

- 深度学习的许多研究成果，离不开对大脑认知原理的研究，尤其是视觉原理的研究。
- 1981 年的诺贝尔医学奖，颁发给了 DavidHubel（出生于加拿大的美国神经生物学家）和 TorstenWiesel，以及 RogerSperry。前两位的主要贡献，是“发现了视觉系统的信息处理”，可视皮层是分级的。

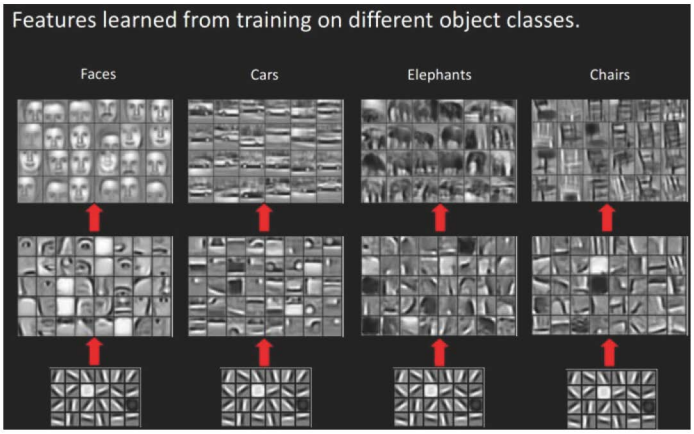
人类的视觉原理如下：

（人类的视觉原理）



（人脑进行人脸识别示例，图片来自网络）

对于不同的物体，人类视觉也是通过这样逐层分级，来进行认知的：



（人类视觉对不同物体逐层分级，图片来自网络）

我们可以看到，在最底层特征基本上是类似的，就是各种边缘，越往上，越能提取出此类物体的一些特征（轮子、眼睛、躯干等），到最上层，不同的高级特征最终组

合成相应的图像，从而能够让人类准确的区分不同的物体。

那么我们可以很自然的想到：可不可以模仿人类大脑的这个特点，构造多层的神经网络，较低层的识别初级的图像特征，若干底层特征组成更上一层特征，最终通过多个层级的组合，最终在顶层做出分类呢？答案是肯定的，这也是许多深度学习算法（包括 CNN）的灵感来源。

卷积神经网络 (CNN) 的基本原理

卷积神经网络包括输入层、隐含层、输出层，其中隐含层包括卷积层、池化层和全连接层 3 类常见构筑，这 3 类分别负责的分工是卷积层对输入数据进行特征提取，池化层用来大幅降低参数量级（降维、防止过拟合），全连接层类似传统神经网络的部分，用来输出想要的结果。

（CNN 的基本原理）

迁移学习

由于我们的图片数据主要来自于内部网络，同时受制于计算资源的问题，我们需要选择一种训练方式来尽可能的适应数据量少和计算资源少的问题，于是我们考虑使用迁移训练。迁移训练是一种基于其他已经训练好的模型进行再训练的技术，基于诸如 ImageNet 等数据集经过大量运算训练出的如 VGG,Resnet 或 MobileNet 等模型本身已经具备了很好的提取图像特征和输出信息的能力，这就好比站在前人的基础上做事情，只需要再这基础上让模型适应我们的数据就好，这会大大节省训练的成本。

ResNet

在我们的项目中，我们考虑在使用 Resnet 的基础上来进行迁移学习。ResNet 最根本的动机就是所谓的“退化”问题，即当模型的层次加深时，错误率却提高了，这是由于深度加深导致的优化变的苦难的问题，残差网络通过短路连接，在网络向后传播梯度的时候，永远不会出现梯度消失的情况，很好的解决了网络过深带来的问题。

Tensorflow 和机器学习平台

机器学习平台，为传统机器学习和深度学习提供了从数据处理、模型训练、服务部署到预测的一站式服务。机器学习平台底层支持 Tensorflow 框架，同时支持 CPU/GPU 混合调度和高效的资源复用，我们将借机器学习平台的计算能力和 GPU 资源进行训练，同时将 inference 模型部署至机器学习平台的在线预测服务 EAS。

模型搭建

数据预处理

- 数据清洗：通过 odps 的大促表爬取了每个类别约 1000 张图片，但是其中很多图片由于是商家上传的，可能会有无效数据，丢失数据甚至是错误数据，比如我们在处理这些图片的时候发现很多白底图和商品图是混淆的，我们将会对这些数据首先进行一轮清理。
- 人工样本：在常见类别中，我们发现诸如氛围图这类很难爬取到很多，同时这类样本具有明显的特征，于是我们将根据这种特征进行样本制造。我们使用了 node-canvas 人工制作了约 1000 张样本，同时，高斯模糊这一类别实际上往往就是一些商品图进行模糊之后的效果，所以我们对爬取到的商品图使用 opencv 进行高斯模糊，得到样本。
- 数据增强：由于我们场景的特殊性，我们不能采用一些传统的数据增强的方式，比如高斯模糊（因为我们有一类就是高斯模糊），但是我们进行了一些简单的诸如位移和轻微旋转等数据增强方式。

- TFRecord 转化：TFRecord 是 Tensorflow 官方设计并推荐的一种数据存储格式，每个 TFRecord 内部存储了多个 TFExample, 可可以想象每个 TFExmaple 就是对应一组数据（X，y），TFExample 其实是一种谷歌官方开发的数据框架序列化格式，类似于 Javascript 序列化输出的 JSON 或者 Python 序列化输出的 Pickle 等格式，但是 protobuf 体积更小，数据更快，效率更高，从 Tensorflow 源码中也可以随处可见这种数据格式。以下从我们代码中截取的片段是针对一组数据创建 TfExample。

其中我们制定了三个 Feature 我们之后再训练中将会用到的，image/encoded 就是图片的 bytes 流，label 是分类的类别，image/format 是图片类型，将会在之后 slim.tfexample_decoder.Image 函数解析 Tfrecord 中使用。

模型构建

迁移训练模型建立

TF-slim 是 Tensorflow 轻量级的高阶 API, 可以很方便的定义，训练和评估模型。TF-slim 官方提供了很多知名的 CNN 网络的预训练模型。用户可以通过官方 Github 进行模型参数的下载，同时调用 tensorflow.contrib.slim.net 中的方法加载模型结构，以下是我们定义的 predict 函数，此函数将在训练时和预测时提供定义流图中经过模型的部分。注意预训练模型只提供了卷积层的实现，为符合我们的分类问题，我们还需要把输出的卷积结果压平，同时加一层全链接层加 softmax 函数进行预测。

模型训练

我们通过 slim 提供的 assign_from_checkpoint_fn 函数加载下载的 mobileNet 预训练模型的参数，使用之前定义的数据流图进行训练，同时在训练的过程中输出 checkPoint 和相关 Log

模型预测

在模型训练期间，我们会定时保存训练数据，保存数据主要通过 Tensorflow 的 tf.train.Saver 相关方法来实现的，Tensoflow 的保存会生成以下四种文件类型：

- .meta 文件：保存了模型的数据图
- .ckpt.data 文件：保存了模型变量的信息，包含 weights,bias 等信息
- .ckpt.index：描述了张量的 key 和 value 的对应信息
- .checkpoint：保存的模型和模型的相关信息

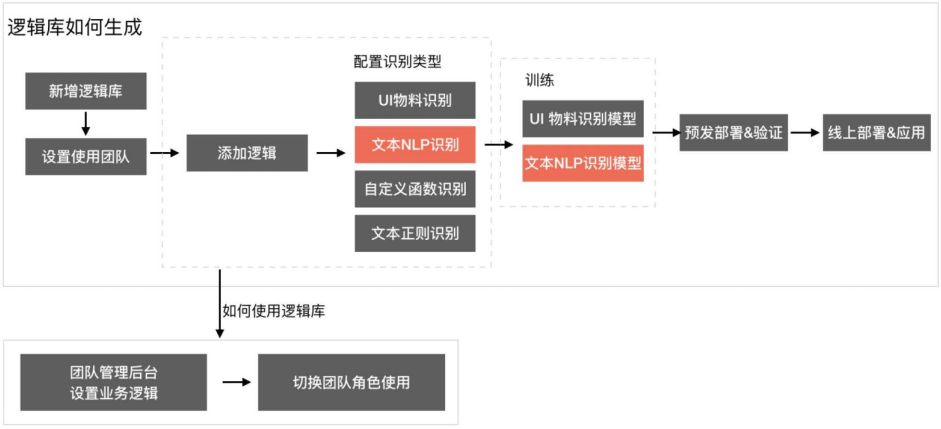
实际上可以看到模型保存时会生成相当多的信息，而其中的很多信息其实在使用模型进行预测时并不是必须的，那么我们就需要对导出的记录信息进行优化，实现高性能的预测。

首先，我们将对保存的模型进行冻结，Tensorflow 模型冻结是指把计算图的定义和模型权重合并到同一个文件中 , 并且只保留计算图中预测需要的部分，而不再需要训练相关的部分，下面我们的代码片段就是将计算图中所有的变量转化为常量。

产品方案

我们考虑到不同业务的数据模型和智能识别及绑定的字段不一样，因此实现了一套可以在线新增分类并配置样本，在线训练模型生成服务，然后通过配置方式使用。

流程图



（产品流程图）

还原模块自动绑定效果

（字段绑定效果）

未来展望

- 对不同场景静态文案分析，我们分析影响绑定准确率的一个部分原因是不同设计和模块里的静态文案多样性，导致的识别出符合预期的结果，后面我们会着重针对不同业务场景静态文案梳理，识别前对静态文案过滤，同时完善通用配置及识别能力。
- 对 NLP 识别和图片分类识别模型优化，对识别绑定不准的数据进行反馈召回，再对反馈召回的数据分析重新优化识别模块和链路，最终提高绑定准确率。
- 字段标准化推进

业务逻辑智能生成篇

作为阿里经济体前端委员会四大技术方向之一，前端智能化项目经历了 2019 双十一的阶段性考验，交出了不错的答卷，天猫淘宝双十一会场新增模块 79.34% 的线上代码由前端智能化项目自动生成。在此期间研发小组经历了许多困难与思考，本次《前端代码是怎样智能生成的》系列分享，将与大家分享前端智能化项目中技术与思考的点点滴滴。

【 阅读提示 】

- 全文较长，部分术语阿里淘系内部营销领域特有。有困惑之处可以评论交流。
- 本文重点介绍 D2C 面对阿里内部淘系大促场景遇到的问题和解决方案，更多的研发场景我们将在不远的将来一一涉足。
- 本文提及的部分功能只有阿里内部版 imgcook 才有，社区版 imgcook 暂时无法体验，敬请期待。

概述

imgcook 是以各种设计稿图像（Sketch/PSD/ 静态图片）为原材料烹饪的匠心大厨，通过智能化手段将各种原始设计稿一键生成可维护的 UI 视图代码和逻辑代码。

逻辑开发是前端开发的需求动线图中最后，也是耗时最多的一步。从整个前端的开发过程上看，除了初始的静态视图编写，所有的数据映射、添加动效、函数编写、事件流、埋点日志等代码本质上都是对静态视图信息的一种补充。

下图中，需求的产出是产品、交互设计师、视觉设计师共同协作的结果，而需求落地全由程序员实现。如果说“视觉稿结合 PRD 交互文档”等于最终可交付开发的需求文档，那么“静态视图 + 逻辑”就等于一个前端页面的最终代码。

（需求动线图）

探索历程

前端开发工作属于 GUI（GraphicUserInterface）编程的一种，从命令行时代进入图形用户界面时代之后直到现在，对便捷的进行界面开发的探索就没有停歇过。在年头尚浅的前端领域，也有 MVC 和 MVVM 的设计思想落地和 jquery、Backbone.js、Angular、vue、react 这些优秀框架类库的涌现。

关注点分离（Separationofconcerns）是 GUI 开发领域的指导思想，通过将 **View 视图**和 **Model 数据**分离来简化软件内部结构。事实上大部分界面开发领域设计思想和框架都是遵循此基础思路实现的，html+css+js 的 web 技术也是此思想的一种体现。

集团推崇的 react 的思想比较接近这个最初的核心理念，仅提供 V 和 M，以及一个两者关联的渲染过程。简单来说就是 Ui=render(Data)，我们认同并作为依据之一开展了基于视觉稿视图还原代码的 D2C 项目。

我们本文所要探索的业务逻辑是项目代码中除了 **View 视图**以外的内容。如果说 D2C 是一个视觉稿到代码的过程，那么距最终可上线页面代码的缺失部分就要交给本文的业务逻辑层来实现。

所在分层

业务逻辑层处于 D2C 核心能力的最下游，所有服务化的智能化能力都需要在逻辑层实现最终的落地。

(D2C 技术能力分层)

挑战和难点

现状分析

在 D2C 的体系里，大部分技术体系都是基于设计稿视觉稿维度出发的，目标是解决布局结构、字段类名、内联组件识别的准确性和合理性。而业务逻辑需要补全 D2C 欠缺的能力，技术方案上和整个项目都不太一致。

同时，业务逻辑层作为承上启下的关键层，负责承接 D2C 智能化能力，输出到可视化编排平台。智能化结果的是一个概率，是一个有几率不准确的值，而下游的可视化编排 IDE 却是一个需要有确定性协议的程式，这样才能保证输出的代码可最终上

线。业务逻辑能否实现智能化的稳定落地是我们一个极大的挑战。

在现有的 D2C 链路图里，业务逻辑层的输入信息除了布局算法转换后的 UI 结构，还有以下输入项：

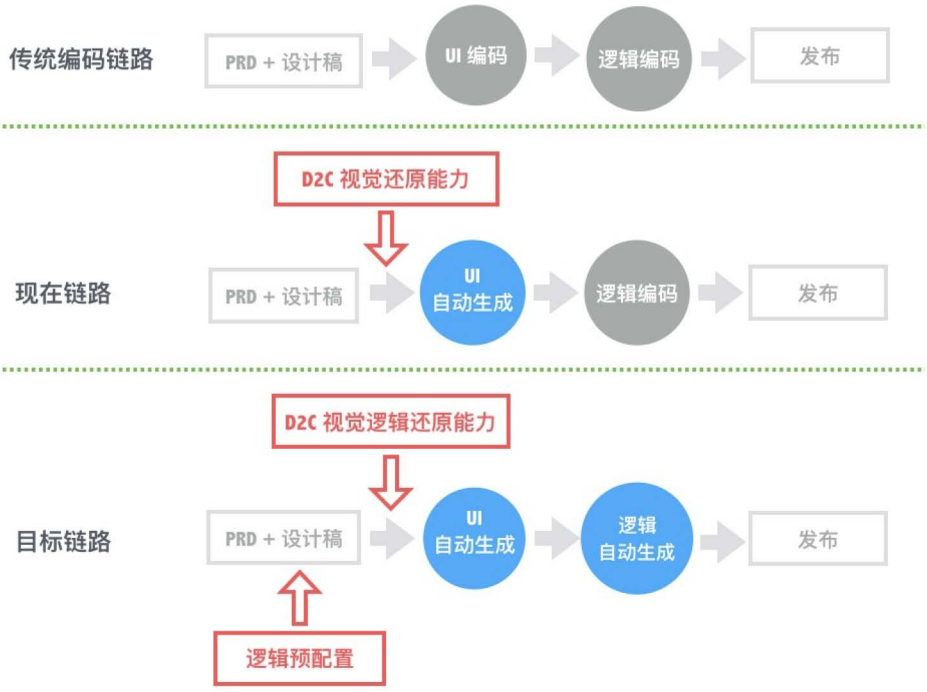
- 语义化推测出的类名
- 字段绑定猜测出的可绑定字段（含图片分类和 NLP 分类）
- 组件物料识别出的小组件
- …

业务逻辑层的产出结果是一份携带逻辑的可视化编排协议。这份协议的字段通过最终实现的功能来看又可划分成如下几种：

- 视图模型
- 字段绑定
- 函数逻辑
- 自定义组件转换
- …

目标链路

传统开发链路中，UI 编码和逻辑需要人工编码。在当前的 D2C 链路中，基于 D2C 视觉还原能力，我们可以将 UI 编码实现自动化开发，基本省略 UI 的开发时间。而 D2C 业务逻辑层的目标，就是实现逻辑编码的自动化。我们希望将 D2C 能力进行全面升级，实现视觉 + 逻辑的统一还原，达到前端编码的零投入。



（传统编码链路、D2C 链路与目标链路）

上图中，蓝色的部分实现了自动化开发，红色的部分是 D2C 能力介入的位置。在我们期望的链路里，D2C 将还原并实现前端全部代码，而设计稿作为唯一的输入必然存在一些需求的缺失，为此我们在还原步骤之前增加逻辑预配置阶段，实现对缺失的需求进行补充。

问题分析

步骤一：思考真实的逻辑开发过程

我们在实现逻辑智能自动化之前，要分析一个逻辑代码是如何编写出来的。

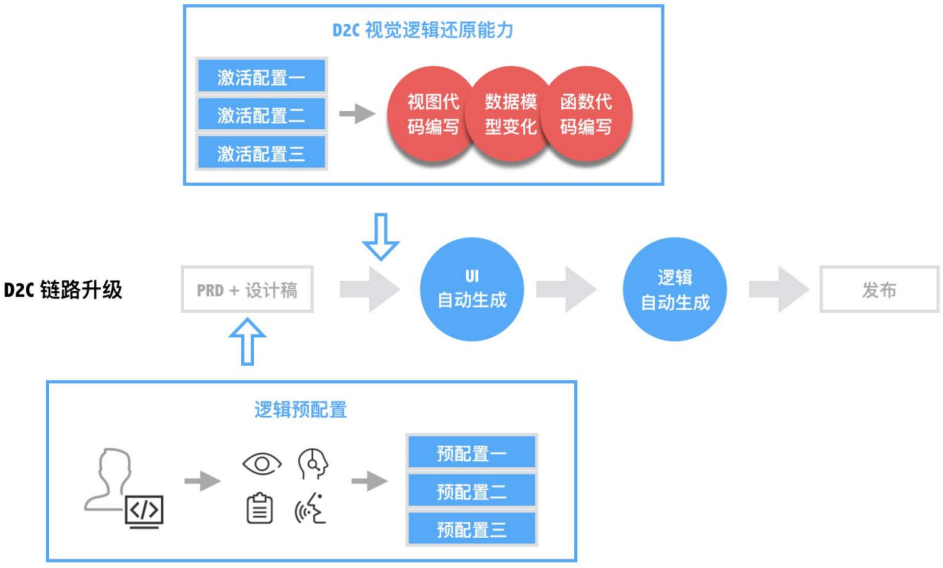
假设我们已经开发好了静态视图，接下来需要为页面增加的逻辑代码需要一个输入的过程，这个输入源可以是我们的视觉稿、过往开发经验、框架下的特殊规则、PD 的需求文档等，从需求的输入源里我们获取需求相关的信息并具象为一个个逻辑点。

比如：视觉稿中有一个写着搜索的按钮，我们观察的同时检索脑海中过往经验，这个搜索大概率是一个点击触发一个网络请求的逻辑，借助需求文档和与相关接口人沟通，我们知晓了这个网络请求的方式和返回内容。据此，一个需求的形态就具象成了一个逻辑点，下一步就是逻辑编码并提测。

（前端需求编码操作动线）

为了实现逻辑的智能生成，以上操作动线必须实现全自动。我们观察需求编码动线，可以明显的发现：以需求具象为分割线，逻辑编码过程可拆分为前后两个大阶段。前一阶段实现需求的收集，后一阶段实现需求的实现，中间具象化的一个个需求点就是我们开发中要编码解决的工作。

在我们期望的链路中，业务逻辑层将为 D2C 能力提供逻辑预配置和逻辑还原两个增量能力。这两个增量能力对应到需求动线上就是需求的收集与需求的实现。在 D2C 体系里，我们称其为**逻辑识别**与**逻辑表达**，并分别融入到链路下述两个位置上。



(D2C 逻辑链路)

步骤二：如何识别逻辑?

D2C 体系里，逻辑识别是一个预先配置的过程。不同的逻辑点可提前配置好不同的预案，用户使用 D2C 时命中了哪一项配置就认为此处存在哪个逻辑。

在 D2C 首先落地的**淘系营销场景**里，我们尝试分析逻辑智能化生成面临的最大问题。

思考一：“如何判别模块所含有的逻辑点?”

（如何观察模块含有的逻辑点）

扫一眼上述模块的布局模式，是一个 1 排 2 模块，需要一个循环逻辑。

“扫一眼”这个动作，通过分析布局算法给出的页面结构可以识别。

分析下各个文字，“跨店满减，错过等明年”是一个利益点类型文案 .

人“分析文字”是对文字提取特征的过程，不同的目的下通常要提取不同的特征。此处文字中“跨店”、“满减”等利益点相关的词高频出现，我们基于 ALiWs 的分词算法和朴素贝叶斯多分类可以有效区分。

“已售 1000 件”看起来需要绑定到月销字段上；

此处也是个分析文字的过程，用正则可以准确区分。

左下方的券在含有一定显著的视觉特征，对于这种小区块通常抽象为一个业务组件；

左下方的券是一个业务组件，其样式，文字，节点数目存在数据特征，我们可以运用传统机器学习进行分析。

商品的图片是一张白底图，大概率绑定到白底图字段上；

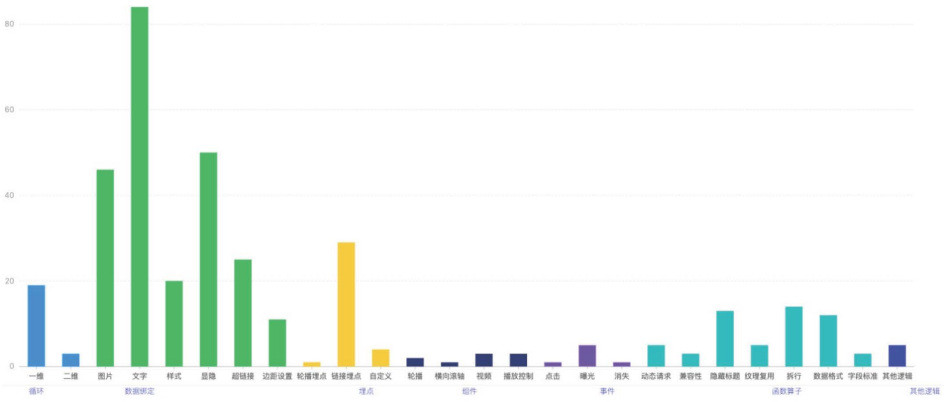
商品图是一类特征明显的图，基于图像分类算法可较为轻松的识别。

立即购买按钮可能需要一个跳转到详情页的事件，也有不跳转，转而交给模块外层来跳转。

这个逻辑只能通过领域经验来识别。

思考二：“是否可以覆盖我们的场景?”

类似的分析过程还有很多，我们暂不赘述。为了能解决命题，我们迫切的需要知道实际场景中上述逻辑点的构成情况，为此我们分析淘系营销大促相关模块，得到如下逻辑点分布柱状图：



（营销模块逻辑分析柱状图）

其中数据绑定类逻辑数量占比 50% 以上，其次是埋点相关逻辑，循环相关逻辑，处理业务的函数相关逻辑，组件逻辑等。总的来说，淘系营销场景的模块开发逻辑是一套有规矩可循、有规范可依，可枚举、可复用、模式化、体系化的程式。经过多年双 11 验证，基本可以确定当下的规范可以满足业务需求，不会在短时间内有大规模

的未知需求涌入。

（淘系营销模块开发规范）

思考三：“我们的识别手段有哪些?”

D2C 体系本身具有许多底层智能化手段，辅助以专家经验，可以对上述逻辑进行全面检索识别。举例如下：

随机森林算法：随机森林是一个包含多个决策树的分类器，并且其输出的类别是由个别树输出的类别的众数而定。既可以用于回归也可以用于分类任务，并且很容易查看模型的输入特征的相对重要性。

xgboost 多分类：XGBoost 全名叫（eXtremeGradientBoosting）极端梯度提升，经常被用在一些比赛中，其效果显著。它是大规模并行 boostedtree 的工具，它是目前最快最好的开源 boostedtree 工具包。

文本 NLP 分类：基于阿里 PAI 平台提供的 ALiWs 的分词算法和朴素贝叶斯多分类进行文本分析。AliWS 的主要主要功能包括：歧义切分，多粒度切分，命名实体识别，词性标注，语义标注，支持用户自己维护用户词典，支持用户干预或纠正分词错误。

图片分类：对业务场景中的图片进行分类，使用 CNN 网络，在 ResNet 的基础上进行迁移训练。同样部署于 PAI 平台之上，和文本 NLP 分类产品化链路完全一致。

语义化服务：D2C 基于移动场景定制的类名语义服务。内部运用专家系统制定策略树，在具体判别过程中运用 AIinlp 语义实体、词法分析、翻译等二方服务，并自建 iconFont 服务实现了小图标的鉴别。

布局算法：D2C 基于自创的行列扫描策略发展出的绝对定位转 flex 布局的规则算法，同时提供了循环检测、局部成组等关键性功能。

等等。

除此之外，我们有一些业务域下独有的逻辑是没有特征的，这部分逻辑使用人工干预手段来实现。

最终，我们决定选用多种识别手段，从布局视觉、文本语义、图像特征、经验规则的层面实现逻辑的判别，并补充必要的额外信息供逻辑表达使用。这些对模块逻辑进行识别的程式我们命名为**逻辑识别器**。

每一个识别器都会基于自身擅长的领域给出鉴别结果，通过全方位的检索视觉稿，达到近似人工思考的目的。

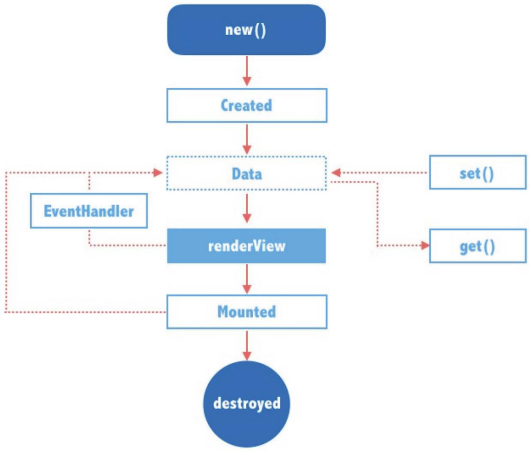
步骤三：如何表达逻辑?

逻辑表达依赖于两个要素：逻辑的表达形式和逻辑的具体内容。

思考一：“逻辑以何种形式表达?”

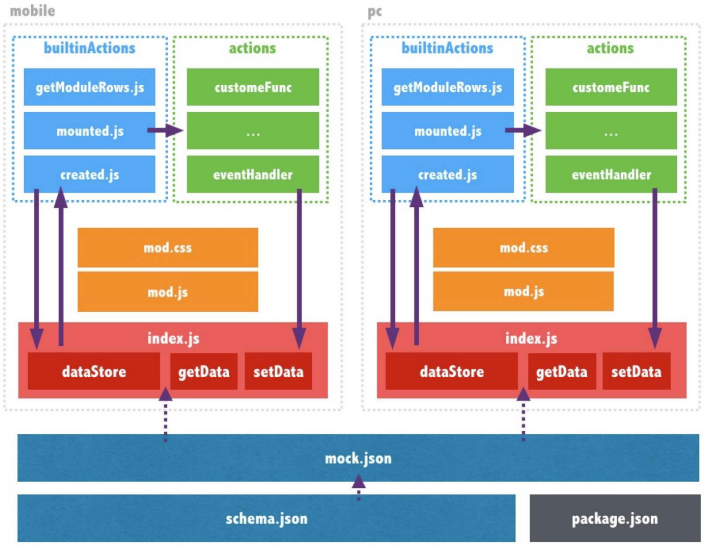
为了明确逻辑的表达形式，我们 D2C 需要一个可承载智能化成果的场景，具体到实现上就是一套承载智能化成果的协议和一个智能化干预平台。

imgcook（D2C 能力落地应用）结合 react、vue 等优秀的前端框架，参考各种竞品，实现了一套简版的基于数据驱动（Ui=render(Data)）的生命周期，并希望用户能基于此规范进行组件的开发和编写。



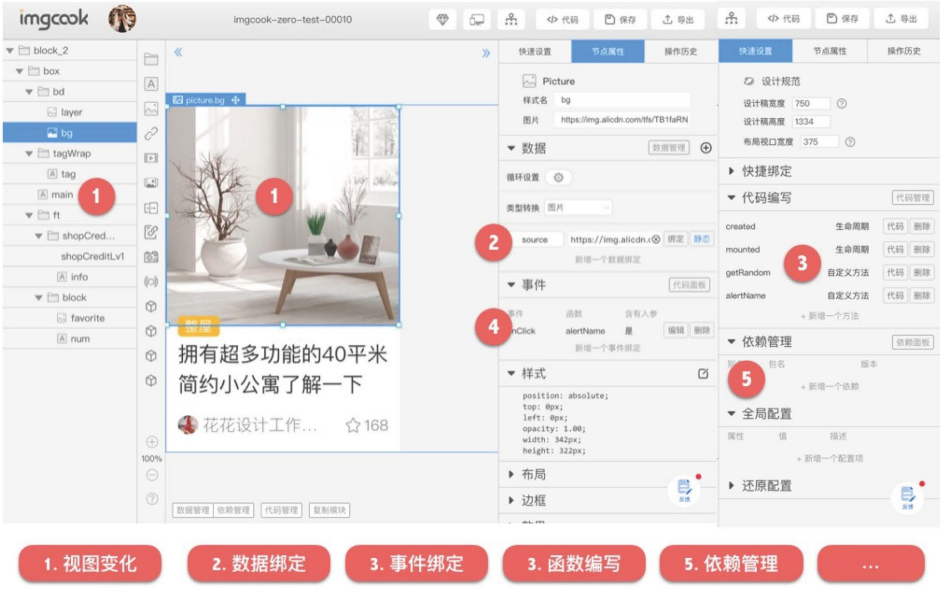
(imgcook 自定义生命周期)

阿里淘系营销于 2019 年将营销模块规范升级到了基于 hooks 的 rax1.0 体系，imgcook 针对新的组件规范，实现了 mobile 和 PC 各一套的代码生成服务。这样开发者在 hooks 下也有生命周期可以使用，对开发者来说，不需要关心模块是 hooks 还是其他技术方案，只需要认准 imgcook 规定的模块开发方式就可以。



(imgcook 天马模块项目结构组织图)

约束了用户的编码规范之后，imgcook 提供了可视化的操作来实现代码。imgcookIDE 目前可实现大部分静态模块的可视化编写，下面是模块逻辑可视化高频操作的面板：



(imgcook 可视化面板【内部版】)

以淘系营销中的典型需求举例，在 imgcook 的可视化编辑器内，我们通常会这样实现（假定模块当前数据是一个单循环一排二商品模块，循环迭代对象 **item**）

- **为节点的价格绑定数据**：点击节点属性 -> 数据 -> 新增一个数据绑定，值为 item.itemActPrice。
- **模块不足一行的时候进行截断**：点击快速设置 -> 代码编写 -> 新增一个方法 -> 编写截断代码，将 items 的长度不被 2 整除的数据去除。
- **埋点逻辑**：普通埋点需要点击节点属性 -> 类型切换为埋点链接，点击新增一个数据绑定，增加 data-track-type、exp-type 等属性，并为其增加数据绑定；主会场实时埋点在做节点类型转换和数据绑定之余，还需要在循环节点里新增一个用于实时曝光的节点，其曝光类型需要设定为实时曝光。

- **加载更多**：下滑式加载更多需要为模块增加曝光事件，事件句柄里编写加载更多代码。点击加载更多需要为模块增加点击事件，事件句柄里编写加载更多代码
- ...

对各个逻辑的操作步骤进行抽象化梳理，得出了如下逻辑实现步骤：

	视图变化操作	数据绑定操作	事件绑定操作	函数算子编写	依赖注入
活动价		数据绑定			
掉坑逻辑				函数算子	
埋点(普通埋点)	视图变化	数据绑定			埋点组件依赖
埋点(实时埋点)	视图变化	数据绑定			埋点组件依赖
加载更多(下滑式)			事件绑定	函数算子	
加载更多(点击式)			事件绑定	函数算子	

（逻辑实现步骤抽象梳理）

表格里每一个列都是 imgcookIDE 一类抽象化的能力，可见大部分逻辑都可以通过配置的形式来实现。由于大规模业务逻辑的营销模块较少，所以我们决定基于复用而不是基于推测来生成函数算子，仅使用执行顺序和是否有返回值来控制流程，更为细粒度的算数、逻辑操作符和流程控制语句暂不在我们的考虑范围内。

思考二："逻辑以何种内容填充?"

可视化和真实代码之间的媒介就是 imgcook 的协议，接下来我们就需要实现协议的自动化生成。

自动化的协议生成的核心是**内容**。节点将执行何种操作不是关键，要绑定到哪个节点、生成的代码里面内容才是关键。为此，我们逻辑识别器执行时会将当前处理过

程涉及到的节点信息、全局变量、人工配置等内容进行传递，在逻辑表达的运行时里执行注入，这些当前逻辑要表达清晰准确所必需的数据在 D2C 业务逻辑层里被命名为**逻辑上下文**。

下面来看一些真实的**逻辑上下文**：

逻辑一：为节点绑定活动价

```
1// “逻辑上下文”
2constrecognizeResult={
3     element:"Text_0",         // 哪个节点需要绑定数据
4     meta:    {
5              expression:"item.itemActPrice"// 绑定具体表达式是什么
6     }
7}
```

```
1// 最终翻译到 imgcook 协议里的示例协议（imgcook 私有协议里将数据绑定都提到了最外层
2constlayoutJson={
3     id:"root 节点 ",
4     children:[...],// 节点树
5     dataBindStore:[
6     {
7     belongId:"Text_0",        //{{element}}
8     value:{
9     isStatic:false,
10    expression:"item.itemActPrice"//  {{meta.expression}}
11    }
12    }
13    ]
14    ...
15}
```

逻辑二：不足一行截断渲染数组

此逻辑函数内容是一个可复用的 xtpl 模板，可以直接访问 recognizeResult 作为渲染上下文。

```
1// “逻辑上下文”：scope 是逻辑层核心分析页面布局的处理结果之一，每个逻辑上下文都可访问
2constrecognizeResult={
3     "element":null,// 此逻辑没有挂载节点
4     "meta":{},// 此逻辑也不需要额外参数
5     "scope":{
```

```
    "gSize":2,// 当前模块是一个 1 排 2 模块，不满足 2 个的行将被删除
    "loop":"items",// 当前模块以 data.items 这个数组属性做循环
    "loopArgs":"item",// 循环内部迭代对象名为 item
    }
}
```

```
// 最终翻译到 imgcook 私有协议里的函数里的示例协议
constlayoutJson={
    id:"root 节点 ",
    children:[...],// 节点树
    scirptStore:[
    {
    content:`
...
{{~#if(ctx.userLogicConfig.sliceFloor)}}
    // 掉坑处理，根据用户是否使用 sliceFloor 来使用
    constcount=Math.floor(data.{{scope.loop}}.length/gSize)*gSiz
    {{scope.loop}}=data.{{scope.loop}}.slice(0,count);
{{~/if}}
...
`,
    name:"getModuleRows",// 截断逻辑写在拆行函数里
    type:"custom"
    }
    ]
    ...
}
```

使用渲染上下文可以在最终要增加的协议中“占坑”，实现协议的准确表述。

通过冗长的分析，我们业务逻辑智能生成的命题已经细化到了**如何运用智能化能力识别逻辑并产出上下文**和**基于上下文实现逻辑自动化表达**两点上了。这两点确认可行后，我们开始进行正式设计。

智能逻辑层设计

方案概述

根据对命题完整的推导，我们已经有了业务逻辑层能力的完整轮廓。

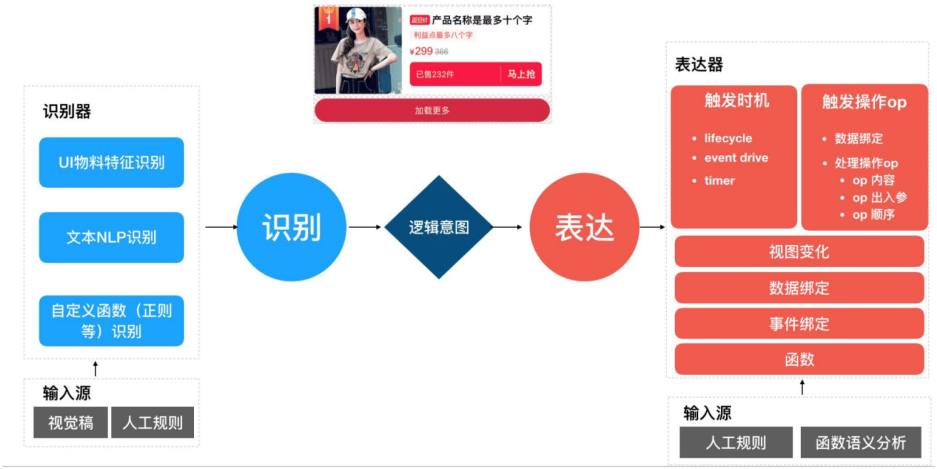
逻辑识别 + 逻辑表达 = 逻辑意图

逻辑识别器需要您根据实际场景决定使用何种方式来执行识别。此阶段接收视觉稿和人工规则，输出为识别结果。

类比人的思考过程，D2C 此时已经“确认了模块的需求”。

逻辑表达器是 imgcook 可视化操作的预置版，将识别结果进行翻译，将此条逻辑带来的影响直接表现在最终的模块上。

类比人的思考过程，D2C 此时已经“写完了这个需求”。



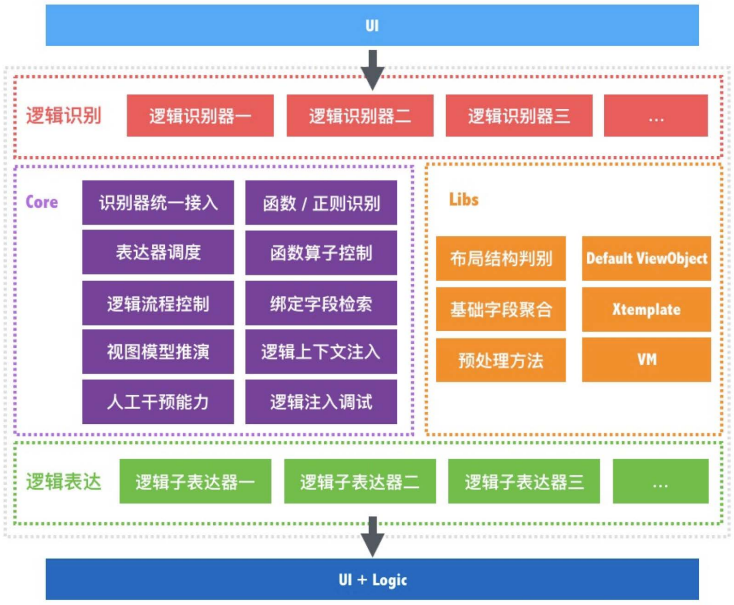
（D2C 业务逻辑层能力）

功能划分

基于上述推导过程和职责界定，我们将业务逻辑层划分为**逻辑识别**、**逻辑表达**、**逻辑核心**等模块。

- 逻辑识别提供智能化能力统一接入，确保业务逻辑可以被指定识别器准确识别并产出统一逻辑上下文。
- 逻辑表达负责对逻辑进行可视化协议的配置，能自动应用逻辑上下文，在逻辑被识别出来之后自动表现到模块上。

- 逻辑层核心部分提供两者的串联整合和扩展能力，比如执行时序控制、布局模式支持、兜底 VO（ViewObject）生成、逻辑上下文注入、人工干预等。从全局把控整个静态设计稿逻辑化的过程。
- Libs 提供基础能力，一部分供核心调用，一部分可在逻辑上下文中供识别函数调用。



（D2C 业务逻辑层结构图）

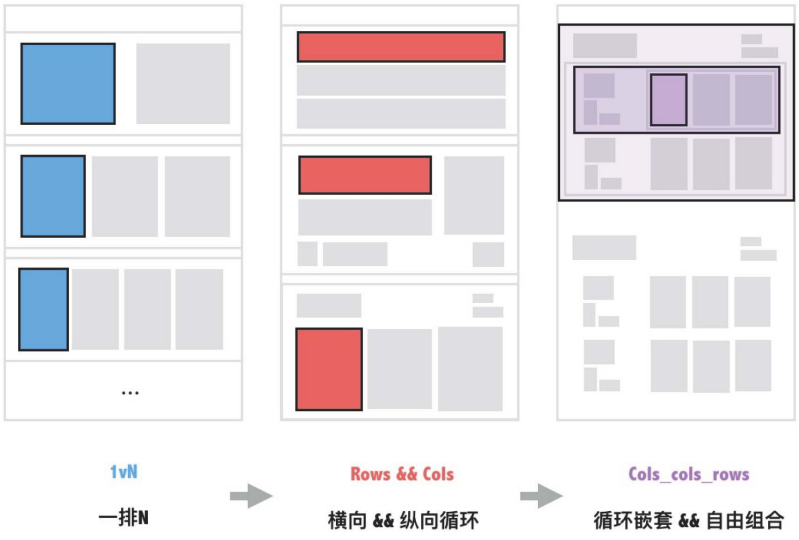
前文也说过，D2C 试点的是阿里内部淘系营销领域，为了方便以后接入集团其他业务场景，我们有了附着于团队的**逻辑场景**概念。逻辑场景是解决一个业务域的逻辑合集，只要某个业务域下的业务逻辑可枚举、可规范、可定制，就可以构建自己的逻辑场景，方便自己的团队的其他同学进行模块开发。

逻辑核心功能拆解

布局模式识别

D2C 支持对一排 N 类型模块，横向纵向循环和任意层级的循环嵌套这些布局的

识别，覆盖营销域下大部分的静态模块布局。值得注意的是，D2C 对设计稿的规范程度要求较高，在双 11 这种级别的活动里对模块 100%，这就要求智能化必须有规则化做兜底。为此我们升级了 D2C 设计稿布局还原准确度的要求必须是协议，您可以在设计稿上用标注的形式规整设计稿，确保布局还原结构准确、循环检测正常。



（D2C 已支持的布局模式）

视图模型推演

智能化的前提是标准化。D2C 识别出的视图需要和数据模型（schema）映射上才可以正常表达逻辑。对模块视图布局识别的过程中 D2C 会同步检索 schema，确保循环层级和每一层的字段能对应上。然而现阶段开发者不能保证模块具有成型的 schema，为此 imgcook 实现了视图模型推演，在没有 schema 时能自动推测出模型，确保仅从视觉稿视角出发的 D2C 也是一个完备的体系。

推演是一个构建数据模型树的过程，在布局结构的过程中，我们将循环布局视作枝干，将每个触发绑定数据绑定的节点视作枝干的叶子，叶子的具体内容从淘系过往模块数据聚合结果中获取。

渲染上下文注入

函数识别器和视图表达器是逻辑层两个基于 nodeVM 执行函数的地方，从接口定义上可以看来他们之间的联系。

函数识别器

```
1// 函数识别器入参
2exportinterfaceLayoutJson{
3     children:LayoutJson[];
4     style:any;
5}
6exportinterfaceLayoutResult{
7     ctx:Ctx,// 开发上下文。略
8     scope:Scope,// 全局变量。略
9     UserLogicConfig:UserLogicConfig,// 用户输入。略
10}
11exportinterfaceOptions{
12    utils:Utils,// 工具方法。略
13    _:any,//lodash//https://www.lodashjs.com/docs/latest
14}
15// 函数识别器出参（仅在有识别结果时输出）
16exportinterfaceRecognizeResult{
17    order:number;// 当本次逻辑需要表达顺序控制的时候，将按照 order 自小到大排列
18    element:number;// 本次逻辑挂载的节点 id
19    meta?:any;// 其他识别结果
20}
```

视图表达器

```
1// 视图表达器入参
2exportinterfaceLayoutJson{
3     // 同识别器入参一
4}
5exportinterfaceRecognizeResult{
6     // 识别器的出参
7}
8exportinterfaceOptions{
9     // 同识别器的入参三
10}
11// 视图表达器出参
12exportinterfaceViewResult{
13    layout:LayoutJson// 处理后的布局 json
14}
```

函数算子时序控制

实际场景中，需要经由 D2C 自动化生成的编码相关的逻辑较少。我们采用时序和是否有返回值来做简单数据流向控制。流程控制只会出现在生命周期事件或节点事件的句柄函数里，举个例子，假设我们有三个逻辑需要在 created 里面实现，分别是：“向循环数组塞入一个图片”、“根据一排几来截断数组”和“发送一个曝光埋点”。那么通过配置顺序和是否有返回值，我们可以得到这样的 created 函数。

```
1functioncreated(){
2     letdata=this.get();
3     //created-flow//created 流程开始
4     data=this.addImage(data);// 顺序为 1，有返回值
5     data=this.sliceArray(data);// 顺序为 2，有返回值
6     this.expTrack(data);// 顺序为 3，无返回值
7
8     returndata;
9}
10exportdefaultcreated;
```

人工干预

我们也知道，许多逻辑是无法通过设计稿 Design 获取到信息的。为了解决这种无特征逻辑的生成，我们加入了人工干预来进行协调。

协调方式一是**参数问询**：定义自定义表单获取用户的输入，在内部链路的

layoutResult.userLogicConfig 中访问。

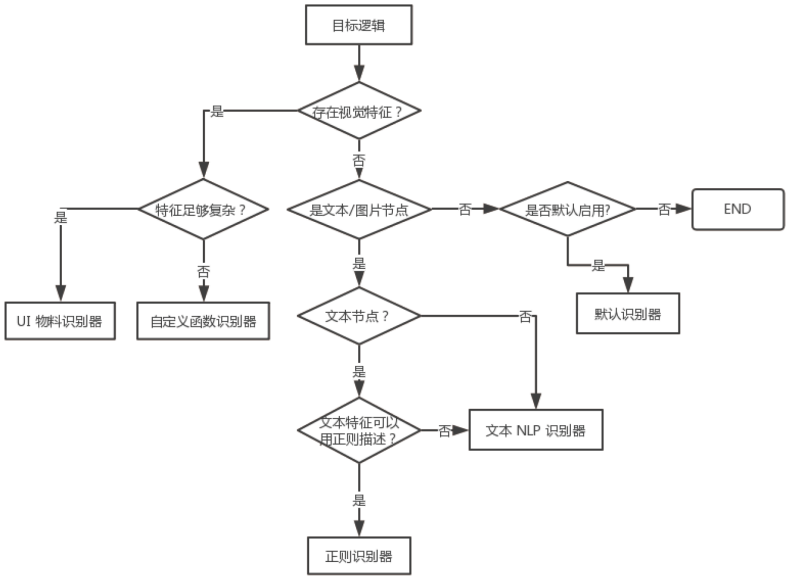
协调方式二是**逻辑过滤**：每个逻辑的配置都有**开发者干预**选项，勾选之后，模块开发者有权对此逻辑进行屏蔽。

这些管控措施都会体现在模块还原的询问弹窗上，若不了解弹窗内容的具体意图，需要联系当前逻辑场景的负责人。

（D2C 开发问询面板）

逻辑识别器一览

逻辑识别器是一个 N 选一的配置方式，对于一个逻辑，我们有如下图示来引导用户使用正确的识别器：



（选择一个正确的 D2C 逻辑识别器）

1. UI 物料识别器

- 特点：
  - 使用 xgboost 算法对模块中含有逻辑的视觉特征节点进行识别，相较于随机森林算法在我们的数据集上表现更优越
  - 适用于子组件视觉特征足够明显，逻辑组件具有一定复杂性的场景
  - 物料识别器本质是一个分类器，只告诉管理员某个节点是某个逻辑的载体。而不会告诉更多信息
  - 当 UI 物料识别器不满足预期时，可以使用**设计稿注入协议**来进行此条逻辑的兜底

2. NLP 文本识别器

- 特点：
  - 基于 ALiWs 的分词算法和朴素贝叶斯多分类实现文本分类模型。对您输入的文本样本进行训练，有助于您对自己问题域下文本进行有效归类。
  - 当您拥有大规模文本体量时，推荐用此方法进行文本的分类
  - 含有**内置识别结果，涵盖不方便走文本 NLP 训练链路的一些常用分类**。比如：价格、原价、商品图、白底图等。

3. 自定义函数识别器

- 特点：
  - 当目标逻辑可以通过 **javascript 代码**分析样式、结构、文字等信息的方式识别出时使用
  - 在没有样本训练 UI 物料识别器和 NLP 文本识别器的情况下，它是一个比较不错的逻辑库建设方案
  - 函数识别器可接收用户传参来做逻辑决策。比如天猫业务场景下对于埋点有两套截然不同的逻辑表述，我们可以编写两个埋点逻辑，在各自的识别函数里做二选一。除此之外，函数识别器可以攫取组件树信息，给逻辑表达器提

供更为强大的逻辑上下文

4. 默认识别器

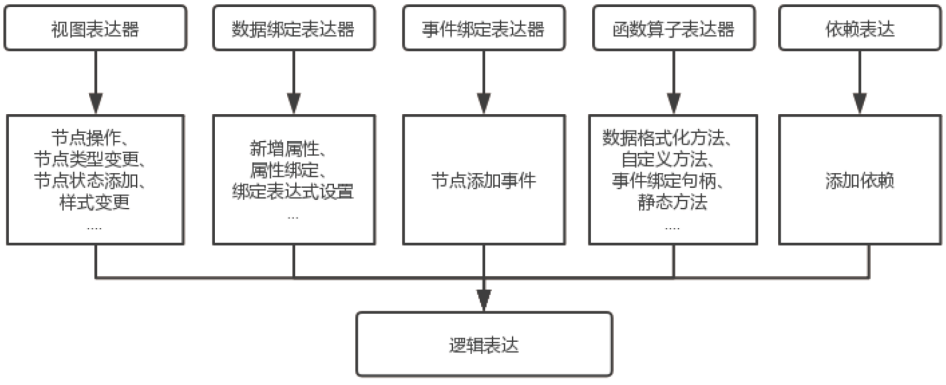
- 特点：
  - 默认逻辑会被所有模块应用，它的逻辑识别结果永远为 true
  - 用于一些视觉层面没有特征的、较为通用的逻辑
  - 多与“**开发者干预**”配套使用，由开发者决策是否使用
  - 默认识别器无法获取组件树信息，如果有攫取信息的需求请移步函数识别器

5. 正则识别器

是 NLP 的正则分析版，能力是 NLP 识别的子集。

逻辑表达器一览

逻辑表达器是多个抽象化的子表达器组合的结果。我们将一个逻辑的实现方式拆解为最细粒度的可视化操作，通过分析此逻辑的具体实现方式，依次在后台配置表达式，进行逻辑组装。当识别器告知表达器当前逻辑激活时，则自动化实现该条逻辑的代码。



（逻辑表达器功能划分）

1. 视图子表达器

- 可以处理视图层面的变更，能力极强，理论上可以覆盖其他所有表达器的能力。未来 imgcook 希望对视图操作也进行可视化配置，所以希望视图表达器只关注视图层面的变化，职责划分明确，不要涉足其他表达器的能力范畴
- 此表达器接收一个被 vm 执行的视图处理函数

2. 数据绑定子表达器

- 可以新增一条标准数据绑定，自动去重。大部分时候都需要借助逻辑上下文内的属性做动态表达
- 此表达器接收一条数据绑定的配置

3. 事件绑定子表达器

- 可以新增一个事件绑定，此事件会默认带一个事件执行句柄，每个事件执行句柄内部可以用函数算子填充流程。
- 此表达器接收一条事件绑定的配置

4. 函数算子子表达器

- 构造一个自定义方法，并决定在哪个句柄里调用。一般来说，函数算子只能在事件执行句柄和生命周期函数的流程中被调用，并通过排序和是否有返回对象来控制流程。
- 此表达器接收一个函数的配置，内容可使用 xtemplate 语法编写

5. 依赖管理子表达器

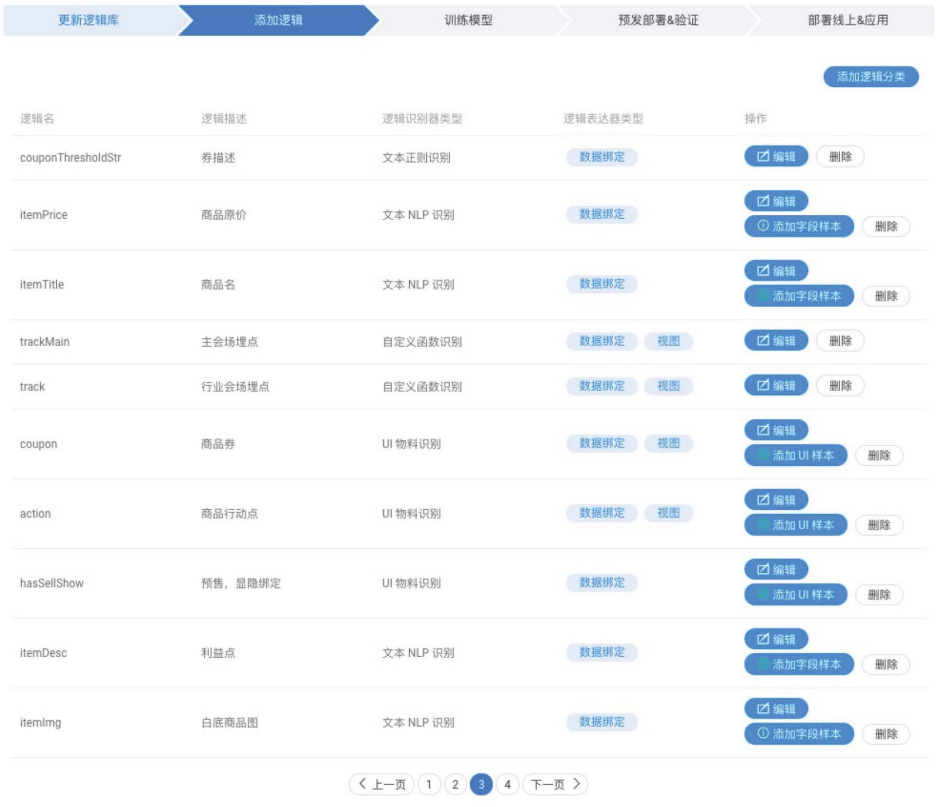
- 在前几个子表达器需要引入三方依赖的时候使用
- 此表达器接收一个依赖的注入

落地成效

双 11 逻辑场景建设

本次双 11 模块的开发中，基于本文提及的智能逻辑链路，imgcook 构建出了一套淘系营销活动独有的业务逻辑场景。内置了基于淘系营销视觉规范的边距设置逻辑、基于淘系营销埋点规范的埋点逻辑、基于 rax-hookssolution 的模块渲染拆行逻辑等默认逻辑。

除此之外，结合文本 NLP、UI 物料分类算法等识别器提供的智能化能力，配置了大量的数据绑定、组件识别的逻辑，当开发者视觉稿中含有此类特征时会自动将逻辑代码运用到结果上。

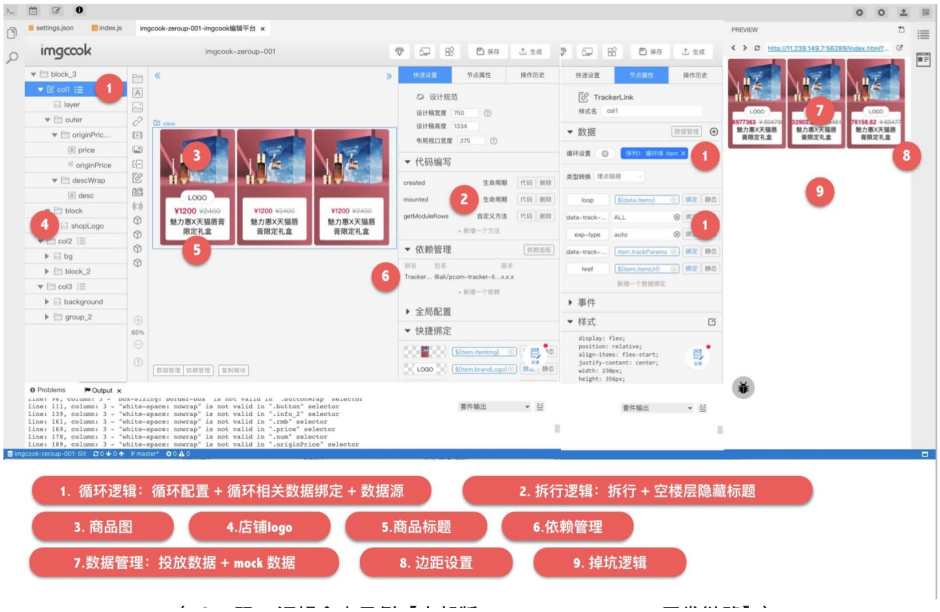


(D2C 双 11 逻辑场景【内部版】)

双 11 逻辑还原指标

根据统计，2019 双 11 有 78.94% 左右模块使用 imgcook 业务逻辑生成链路，生成的模块代码有 79.34% 数量留存至此次还原之后的上线代码中，平均命中的业务逻辑数量约为 14，也就是说平均每个基于 D2C 新链路开发的新模块帮开发者少写了至少十几条逻辑。

在弱逻辑的静态 UI 的模块上，相关指标更高。以下方模块举例，基本可达到一键还原至可提测状态，大大减轻了开发者的工作量。



(D2C 双 11 逻辑命中示例【内部版 imgcookWEBIDE 开发链路】)

未来展望

当前不足

在双 11 落地的过程中也暴露了很多问题，比如流程不够友好，淘系模块开发流程是需求 -> 设计稿 -> 模块开发 -> 前后端联调 -> 模块上线。作为一个为设计稿赋

予逻辑，使其直接转为可上线模块的全新理念的体系，没有为之预留的开发时间。当下我们通过加大人力在开发前提前介入来弥补上述不足，未来我们将会把需求和设计稿的产出过程都纳入业务逻辑层范畴，对于可支持的模块提供一站式研发闭环体验。设计师负责设计 UI，PD 基于 UI 添加需求，开发负责在后台维护可用逻辑。结合团队未来趋势，D2C 在业务逻辑领域的实战经验将在未来有效的助力整个体系。

未来计划

D2C 智能逻辑体系已经验证了自己的想法并迈出了坚实的一步，未来，逻辑智能体系 2.0 将聚焦在以下几个方向：

产品形态升级

如本文开始所述，Design+PRD 才等于需求。D2C 是一个基于设计稿的技术体系，我们将在未来接入需求 PRD 结构化能力，替代 imgcook 目前的人工干预链路，实现全链路零研发。

指标驱动优化

基于双促模块统计结果，我们产出了**代码可用度**的概念，即 imgcook 产出的代码占上线代码的比例。未来，我们将扩展更多的逻辑识别器算法接入、提供更抽象易用的逻辑表达器、实现业务逻辑层内核的组织。imgcook 将基于指标驱动开发，让智能化生成的代码与实际业务产生真实的碰撞，并最终朝着提供更优异的智能化服务而迈进。

ImgcookStudio

imgcook 编辑器内核升级。基于内核我们将衍生出营销业务、社区业务、小程序业务等业务平台，并将逻辑场景的建设扩展到集团级别，在更多的业务场景里实现定制。最终实现前端智能化能力的稳定输出。

小结

简单来说，我们希望有更强的智能化能力，更广的服务场景，更高的提效诉求。使整个体系真正成为一个从视觉稿视图推演并生成全部模块逻辑的智能化服务。从前端编码占比 79.34%，到前端“零研发”，到需求“零研发”，最终到整个需求的“零投入”。

关注imgcook官网

体验一键智能生成代码

扫码加入imgcook 钉钉答疑群

关注imgcook掘金专栏

imgcook 设计稿智能生成代码

关注imgcook知乎专栏

为你带来前端智能化前沿资讯

关注 Alibaba F2E

了解阿里巴巴前端新动态

访问开发者社区

扫码领取更多免费电子书