

# D2C

## D2C 方向介绍

文中的视觉效果，不仅包含看上去的「静态视觉效果」，也包含「响应式」「交互行为」

## 背景

### 什么是 D2C？为什么需要 D2C？

D2C -- design to code，即设计稿转代码。

直到现在为止，设计研发的协作流程都是，设计同学画视觉稿，研发同学还原，期间可能会有各种问题需要沟通，如响应式，交互状态，不符合约定的尺寸(如 13px 的宽高)、颜色(之前没有配置过的色值) 等。等研发同学还原之后，再交由设计同学进行验收。

这个过程中产品的视觉效果由设计控制，但需要传达给研发，交给研发还原。流程繁琐，容易出问题(就事实而言，沟通一定会带来成本，还会带来信息损失)。造成这一现象的根本原因就是，设计自身无法控制产品的视觉效果，需要依靠研发通过代码的方式来实现。

而 d2c (design to code)，核心目的就是为了让设计师控制视觉效果，为视图层代码负责，降低协作过程中的沟通成本，提升效率。

### 我们为什么做 D2C？

目前其实做 D2C 的很多，但就实际而言几乎都只是在解决某一类问题，如活动页，静态页。这也造成了大家对 D2C 的印象 —— 适合活动页 / 静态页等一次性消费的场景 —— 推论：不需要维护的场景。

大多数 D2C 产品都只关心「静态视觉效果」，不信任(大概也不理解)设计师的工作产出(设计稿)，这种思维模式也影响了他们的算法 / 流程设计，最终导致：

1. 假设算法出错，只能靠调整静态视觉效果 / 标注信息来进行修正。在实际使用中，前者基本是不可能的 ...
2. 设计师对产物失去掌控，无法通过自身擅长的领域知识来修正 / 调整最终的产物
3. 响应式缺失，只能靠算法推测(但是推测效果是否符合预期，以及所谓的预期又是谁来指定 ...)

我认为这可能也是 D2C 的产品虽然不少，但都还只是解决垂直场景下的问题，没有真正的进入到日常开发流程中的原因 ...

我们希望设计师能最终为产品的视觉效果负责，研发只需关注产品的业务逻辑即可

## D2C 怎么做？

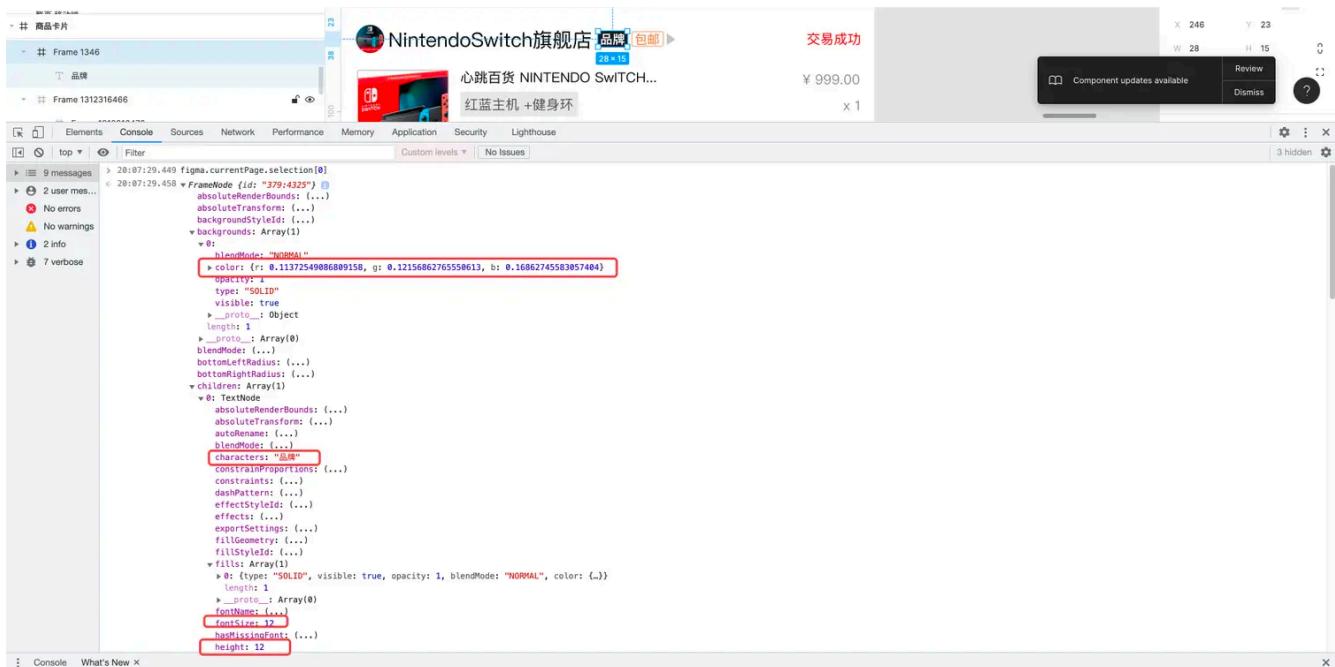
## 设计稿 -- 结构化？

设计稿和图片(PNG/JPG)不一样，它并不是纯视觉的产物，相反，它是一个结构化的东西，只不过由于设计师不关心结构，才导致它的视觉意义大于结构意义。

无论是 PS/Sketch/Figma，都有自己的一套结构说明 & api，如某个元素的坐标(x/y), 宽高(w/h), 颜色(rgb/hsl)等都是可以直接获取到的。

这其实也很容易理解吧，假设让大家去做一个设计软件，总还是需要抽象一个数据结构来描述这些「视觉信息」的

以 Figma 为例，图中的「底色 + 品牌(文字)」，可以获取到如下的「数据结构」，包含如颜色，位置，宽高，字体内容，字号等诸多信息 ...



所以简单概括一下 D2C 做的事情：把一个(大概率)结构混乱但视觉良好的设计稿，转换成结构和视觉都良好的代码

转换后之所以要求「结构优良」，本质上是因为 研发 和 设计 的关注点不一样

研发关注点在于 结构是否清晰、合理。这直接影响到后续研发的开发成本 / 维护成本。

由于设计稿本身是结构化的，并且就静态视觉还原而言，信息是全量的(甚至是过剩的)，所以很自然能得到以下两种做法：

### 方法一：直接映射结构

最直观的想法，直接把设计稿的结构映射到 dom 结构，然后根据设计稿中节点的定位信息去还原。

这个做法其实是 ok 的，并且成本也不算高，因为需要用到的所有信息都是可以通过 api 获取到的，问题在于

暂时无法在文档外展示此内容

- 节点冗余（因为设计师不关心结构），维护成本 +
- DOM 结构混乱，维护成本 ++
- 绝对定位（因为设计稿的定位都是 xy），维护成本 +

就视觉还原而言，这个思路其实是可行的，只不过是产物代码几乎无法维护。

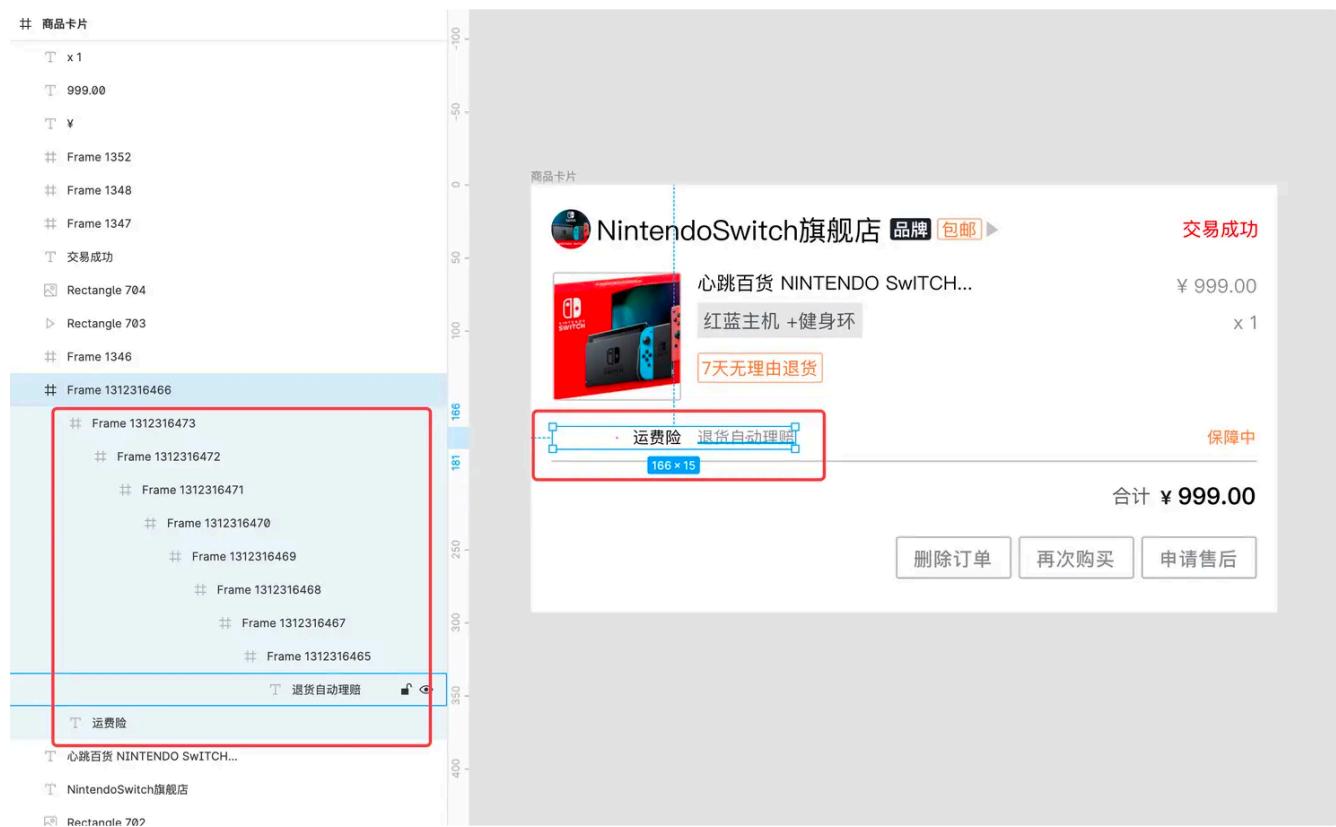
上述「视觉还原」不考虑复杂视觉效果，如「Mask 遮罩」就不能直接映射结构得到 ...

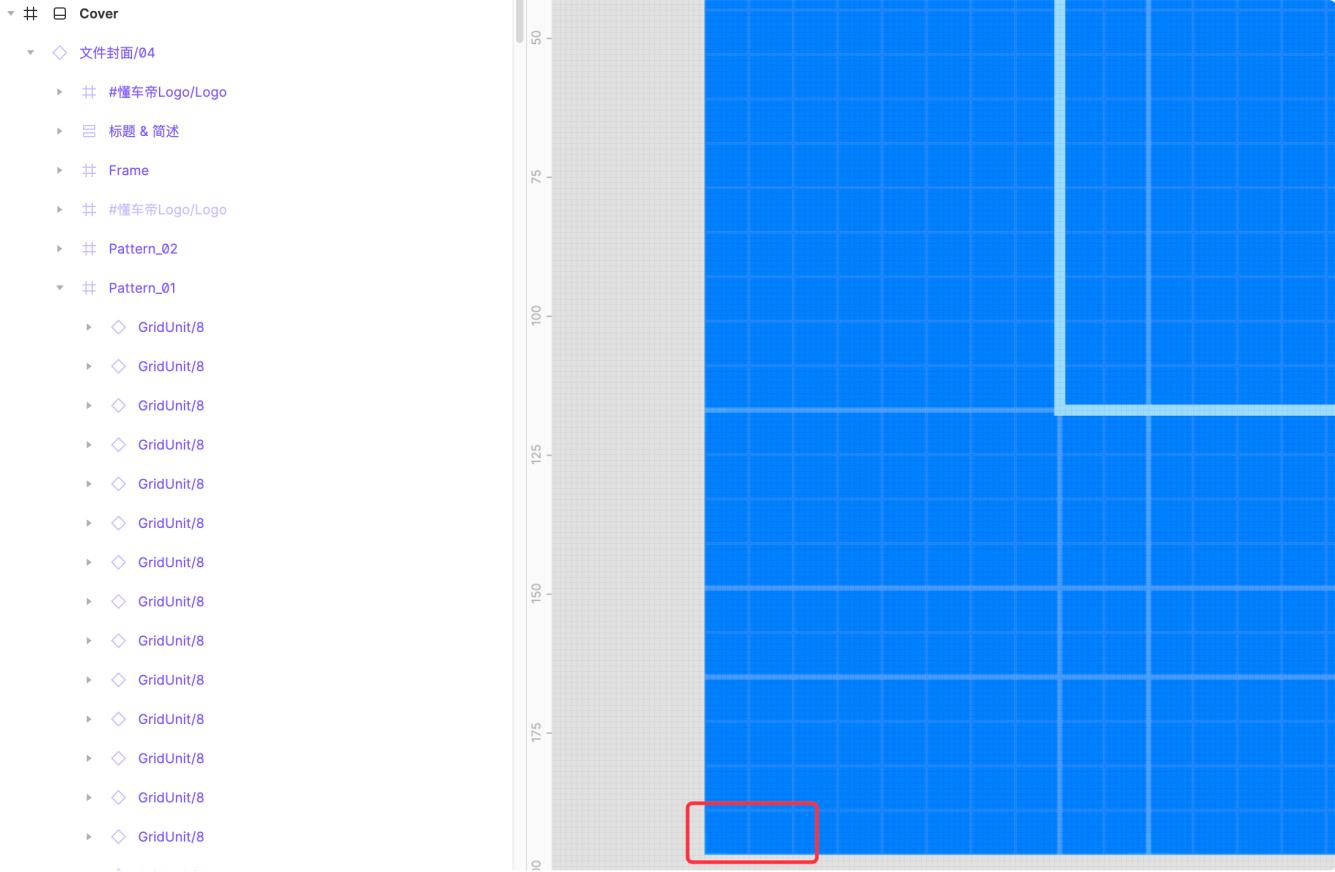
## 方法二：映射到中间结构，再进行转换代码

接着上面的思路，很容易得到第二个思路：完全不信任设计师的结构，而是自己重组结构，基于重组之后的树结构去做视觉还原和代码生成。但是这样的话又会遇到一些其他的问题 ...

如下，视觉上的节点数和实际上的节点数差距可能会很大 ... 类似这种会干扰到我们理解视觉意图的节点，统称「噪音节点」

右图每一个网格都是一个单独的节点 ...





基于上面的问题，以及方法一中的问题，衍生出来以下算法流程

- 预处理：删除噪音节点
- 元素组合：调整结构，主要会包一些容器节点
  - 因为研发需要结构优良的产物，不然理解成本 & 维护成本都会增加
- 布局算法：使产物具有相对定位 / 弹性布局的能力
  - 绝对定位的产物，维护成本会很高
- 响应式推测 / 翻译：使产物能在多机型尺寸下正常显示
  - 降低开发 / 沟通成本

暂时无法在文档外展示此内容

目前各家的方案都是先将设计稿转换成某个中间数据结构，基于该数据结构去进行上述算法处理，最终将处理后的结构转换成特定语言的代码片段。

## AI or 策略

目前的 D2C 领域，基本上存在两种模式的产品，一种更多使用机器学习、深度学习等 AI 领域的算法模型来解决问题，另一种则是通过写各种规则 / 算法来解决问题

我个人认为两种模式在技术上没有严格的好坏差异，只是有一些领域擅长与否的问题。

举个🌰：

- 语义化推测，通过写规则是基本做不了的，但是通过 AI 却能很好的解决(至少能给出可行解 ...)

- 布局算法，由于已经有明确的倾向性和规则，利用 AI 来解决的意义不大

需要明确的是，大多数情况下，AI 并不能给出所谓的「最优解」，只能给出「可行解」

理论上「监督学习」解决问题的模式是：通过大量标注数据学习经验 ...

但这里有一个比较相悖的点：以「元素组合」为例，其实存在很多种「组合方法」，并且多种方法都是「言之有理」，并不存在所谓的「理论最优」。这就导致每个人都可能有一套所谓自己的「标注结果」，最终导致模型无法有效收敛 ...

假设我们能有一套规则来指导每个标注人员去标注数据，提供给 AI 学习 ...

那我们为什么还需要 AI 呢？直接把这套规则写成代码不就好了吗？

目前 Pendah D2C 更多的还是策略层面的算法，还没有过多的 AI 模型参与

## Pendah D2C 解决方案

贴一波文档 -/-

 「Pendah 设计助手」使用说明

 Pendah D2C - 使用说明

 Pendah D2C 系列教程

现阶段 D2C 还比较早期，一定程度上还依赖于设计稿质量 ...

使用过程中有任何问题都可以随时找@杨健 ~

## D2C

主要包含插件和 web 平台。

在插件侧获取 设计稿信息 并解析上传， 或者在插件侧直接获取部分产物代码(jsx / css)

在平台侧能获取完整的产物代码，以及对导出的结构进行若干修改 / 增强 / 设计 props(现阶段平台侧还比较糙 ...)

<https://pendah/bytedance.net/design/project/61cc184efefccb005982a856>

<https://pendah/bytedance.net/design/project/61cc1bf7fefccb005982a88a>

无法复制加载中的内容

整体流程设计上而言其实和其他 D2C 而言没太多区别，只不过我们会在过程中更多的关注所谓的「设计规范」

如

[The Principle of Common Region: Containers Create Groupings](#)

[First Impressions Matter: How Designers Can Support Automatic Cognitive Processing](#)

[Information Scent: How Users Decide Where to Go Next](#)

## 预处理 -- 删除噪音节点

由于现阶段设计稿确实过于混乱 / 繁杂，导致需要有比较「重」的「预处理」流程，针对若干「噪音」进行清洗。将树结构清洗之后能有效降低后续算法的复杂度

### 无视觉意义的节点

有很多情况下，节点在数据层面存在，但是视觉上却「不存在」... 这种节点都需要删除，原因是，设计师最终交付的是「视觉稿」，核心在于「视觉」，而不是「结构」

- 被隐藏的节点，即设置「invisible」
- Opacity: 0
- 无边框 / 无填充 / 无阴影
  - 也有可能存在边框 / 填充，但是其填充本身 opacity: 0，或者 invisible...
  - 阴影依赖于边框 / 填充 ... 即如果某个节点没有边框 & 填充，设置阴影也不会生效 ...
- 宽高为 0
- ...

▼ # 组件树(正常状态) -240

□ Rectangle 1593 (Stroke)

◇ icon/cursor/pointinghand

| Vector 680

| Vector 679

| Vector 662

| Vector 44

▶ Group 151216

▶ # icon\_固定

T 组件树

— Vector 18

T 页面容器

▶ □ 标题

▶ □ Group 4568

▶ □ Group 3662

▶ ++ 组件树未展开-228



## 被覆盖的节点

指某些节点，实际存在，并且也有视觉意义，只不过在渲染层级上被其他节点「完全覆盖」，导致「视觉上」不可见了 ...

简单理解成「zIndex」层级关系问题，高级节点不透明导致低层级节点虽然渲染，但是不可见 ...

左图表示「组件树(正常状态) -240」把「组件列表展开 -228」完全遮盖了

右图为选中区域节点设置 opacity: 80% 之后的效果

▶ ## 设计素材库-飞机稿

▼ ## 画板

▶ ## 属性面板-属性面板

▶ ## icon/cursor/closedhand

▶ ## 基本/中/Default

▶ ## Group 1312315278

▶ ## 逻辑面板/normal

▶ ## Frame 31140

▼ ## 组件树(正常状态) -240

□ Rectangle 1593 (Stroke)

▶ ◇ icon/cursor/pointinghand

| Vector 680

| Vector 679

| Vector 662

| Vector 44

▶ ## Group 191216

▶ ## icon\_固定

T 组件树

— Vector 18

T 页面容器

▶ ## 标题

▶ ## Group 4568

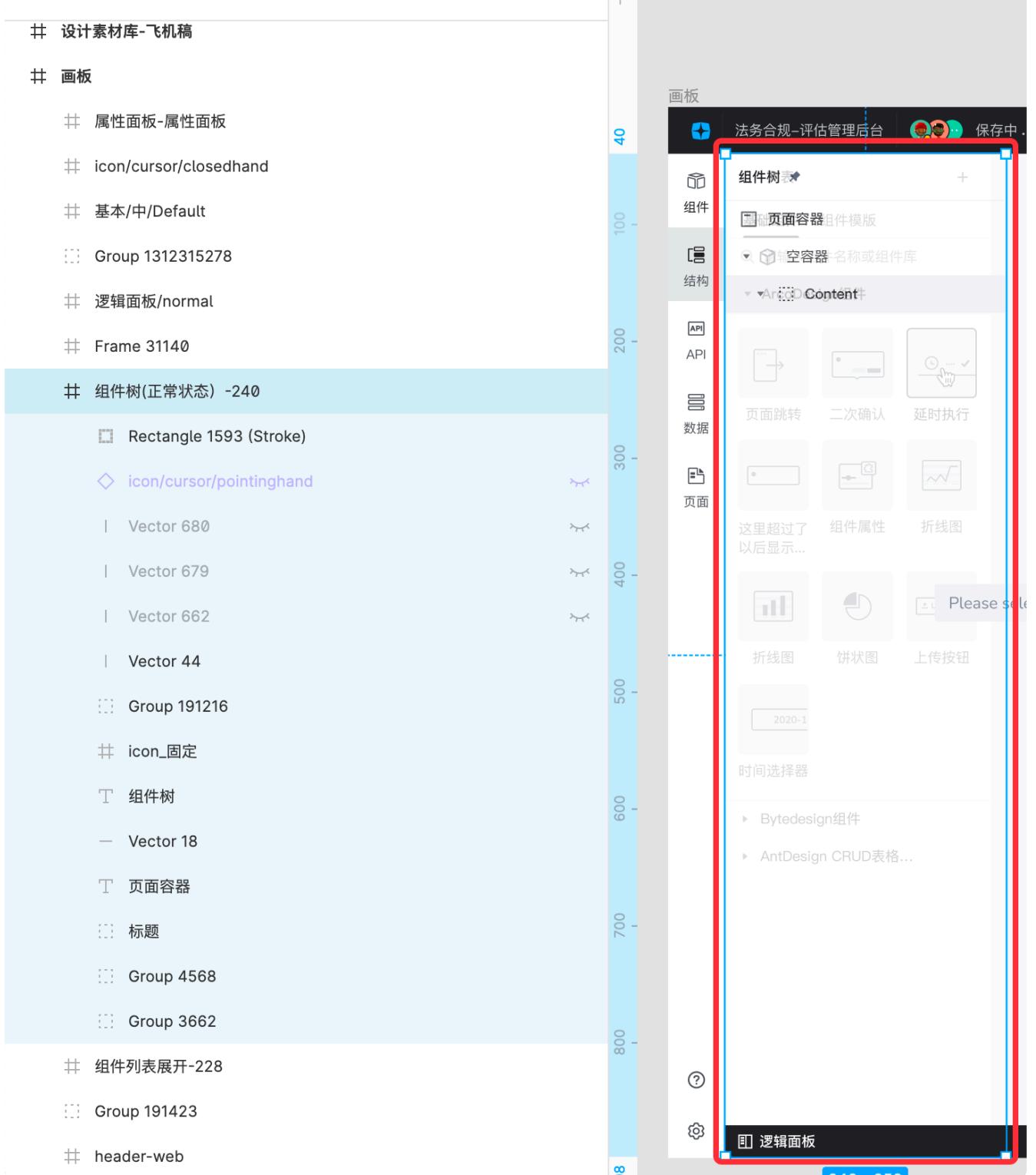
▶ ## Group 3662

▶ ## 组件列表展开-228

▶ ## Group 191423

▶ ## header-web





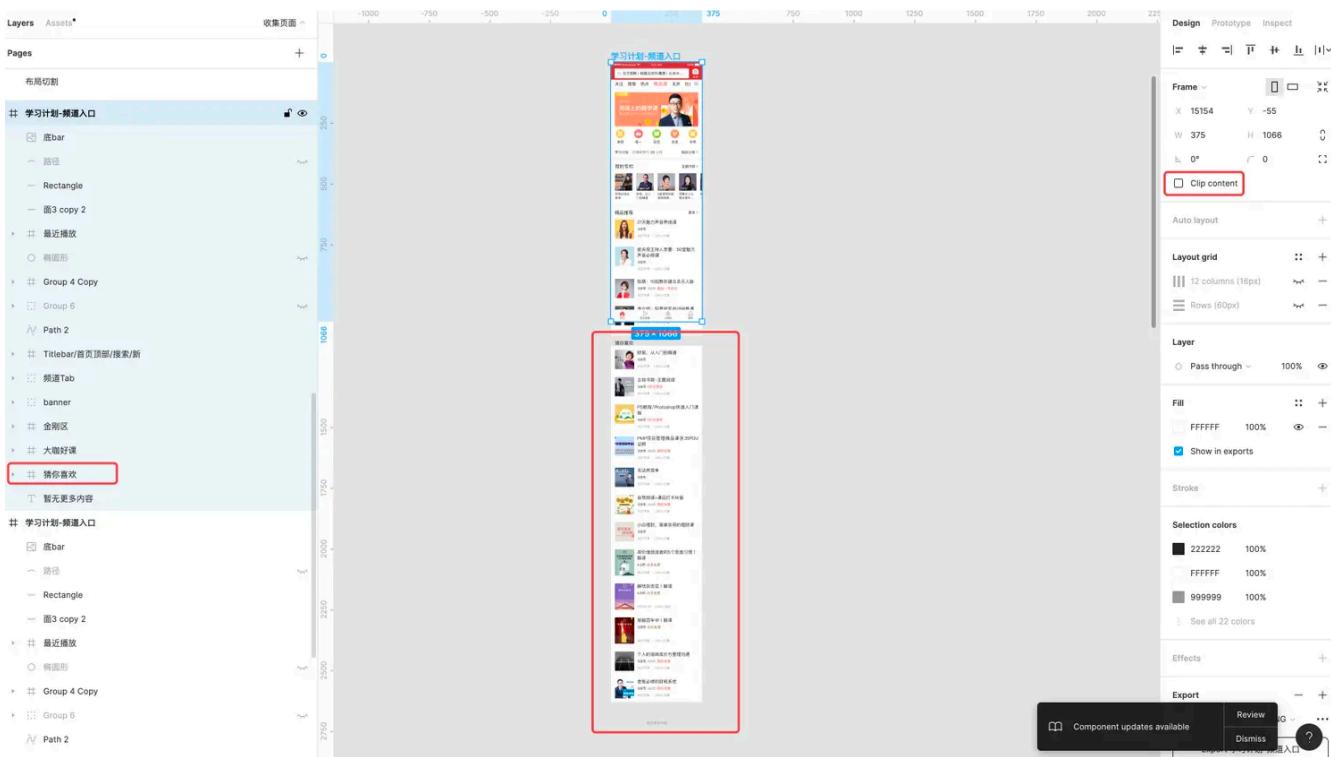
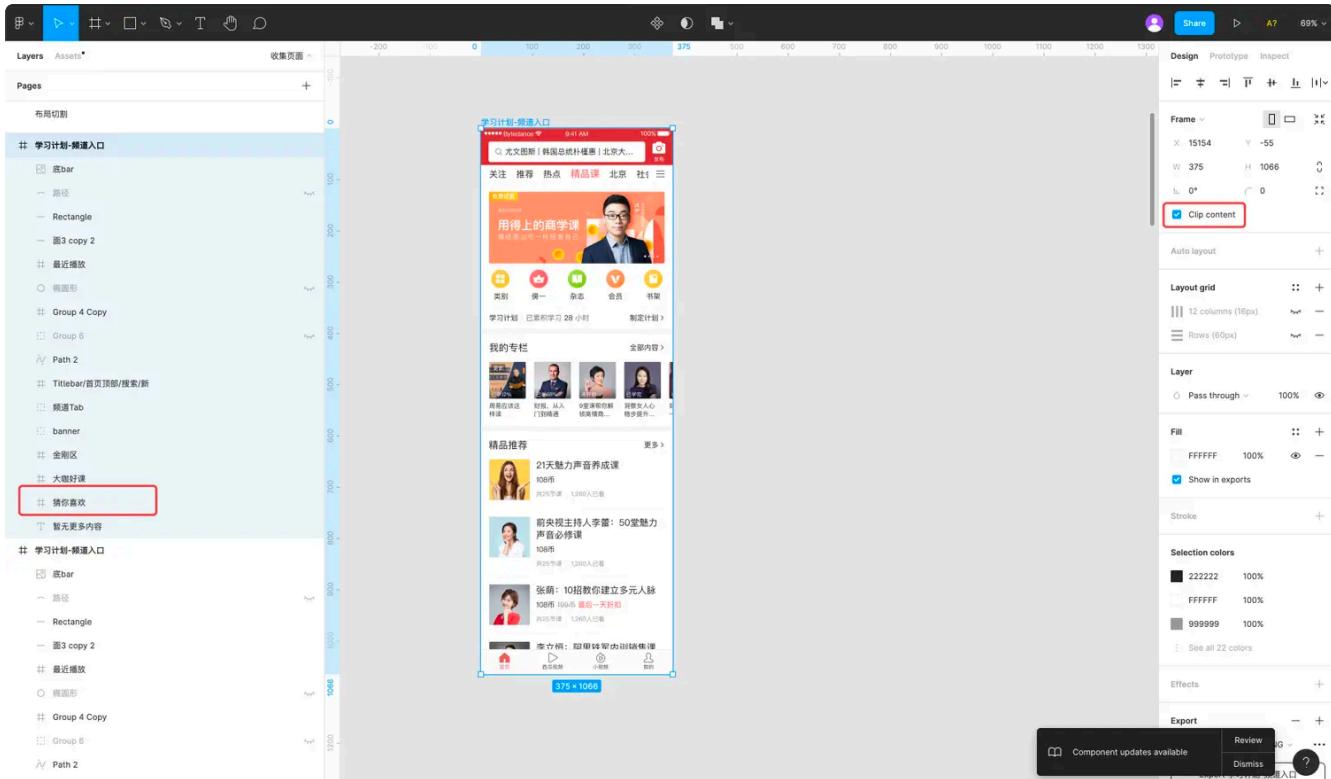
## 被裁切的节点

设计稿中存在「clipsContent」这样的选项，理解成「overflow: hidden」即可。这导致从设计稿中获取到的结构可能会包含「在视窗之外的结构」。

其实大概率是别的设计稿 ... 因为大多数设计师画稿都是 cv...

这部分结构肯定不应该出现在最终的产物代码中，所以需要删除 ...

其实在出代码的时候也同样给节点设置 overflow: hidden 的话，视觉上还原是没问题的 ... 就是产物代码质量会很低 ...



## 用作背景的 Rectangle(矩形) 节点

设计师会习惯用一个额外的「Rectangle(矩形)」节点来承载背景色，但是作为开发，应该没人会这么写代码(特殊需求场景除外)...

所以针对这种节点，我们需要把「Rectangle」的视觉信息(如背景色、边框)都抹到父节点上，再将 Rectangle 删除



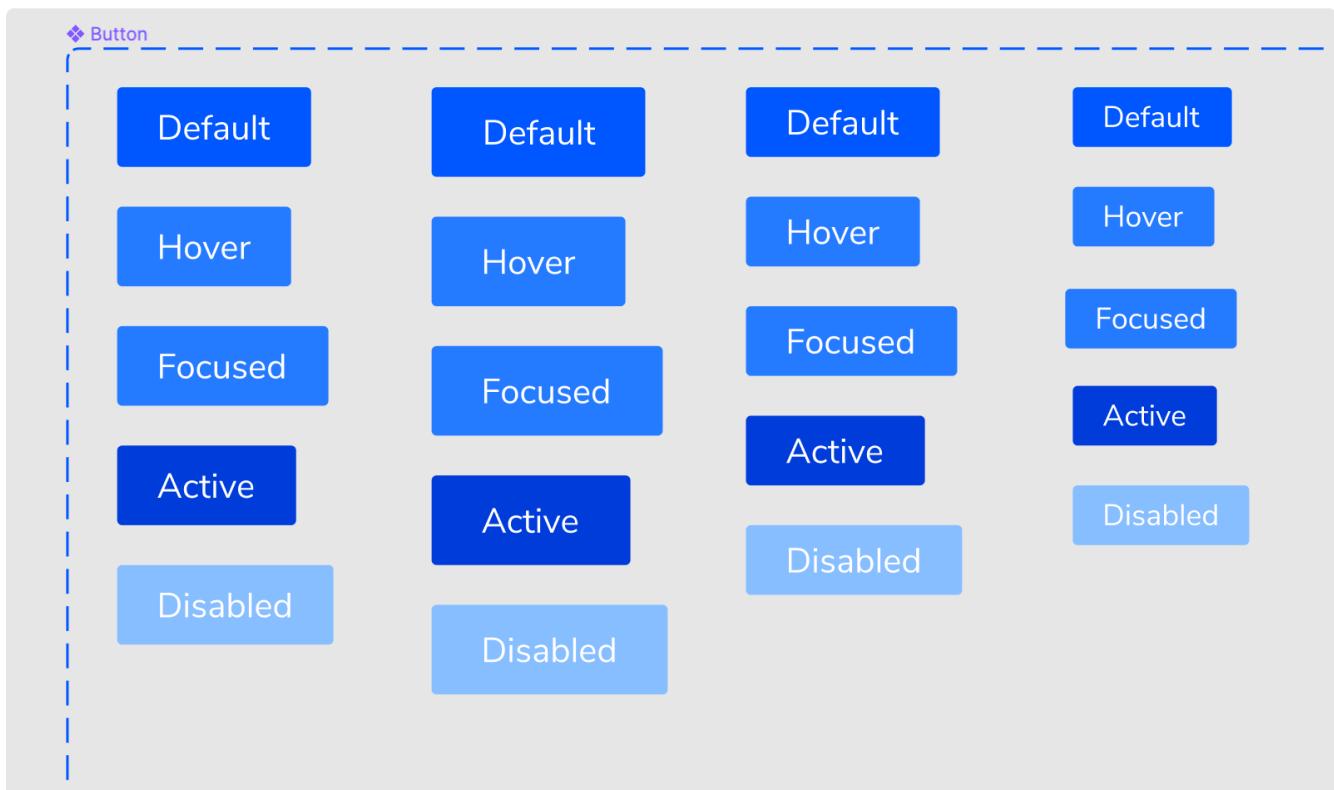
...

## C2D

C2D -- code to design，从代码资源生成设计资源。有些设计资源是需要沉淀的，就像代码一样，这种资源也称作「组件」。

但是目前，设计和研发都在各自维护自己的「组件资产」，两者之间仅有逻辑上的关联，且没有机制能确保两者的一致性。

一致性指的是，以下图「Button」为例，无法确保设计师使用的 Button 和 线上代码的 Button 具有相同的 颜色 /padding/border...

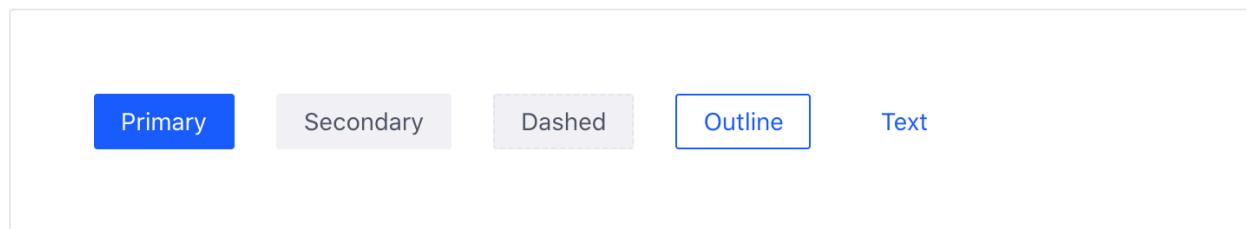


# 按钮 Button

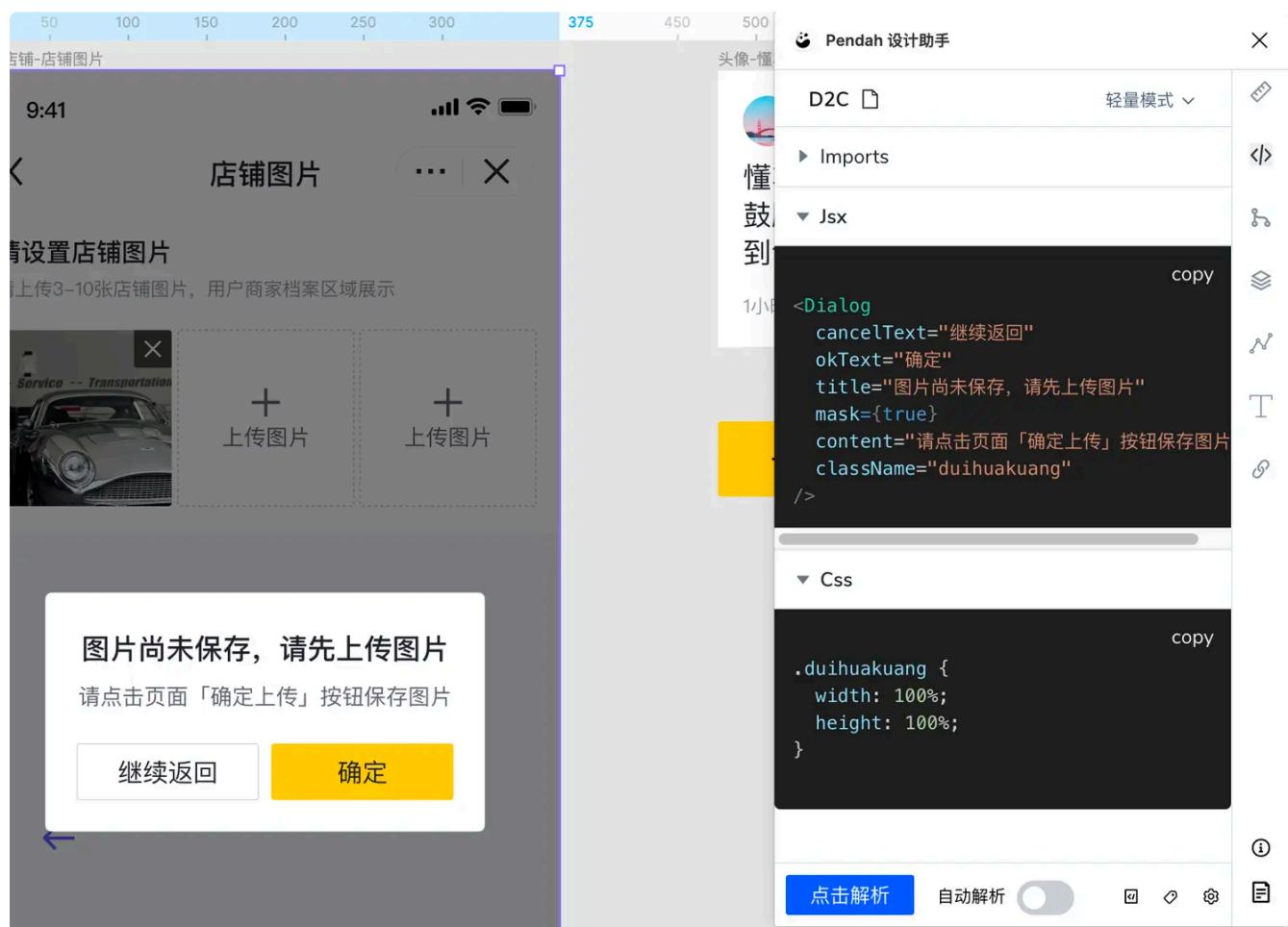
按钮是一种命令组件，可发起一个即时操作。

## 基本用法

按钮分为 主要按钮、次要按钮、虚线按钮、线形按钮和文本按钮五种。



结合到 D2C 的话，主要作用是使产物代码中能包含部分组件信息，如「Button / Tabs...」





目前和 懂车帝团队 关于组件解析的方案，主要是采用 自定义 parser 的形式来进行的，即当 D2C 解析到特定节点时，调用相关 parser 处理部分子树结构，并返回特定的 json 信息。D2C 再利用这些信息去进行后续的布局算法，生成代码

目前这一块已经写了部分 parser，也开始进行一些测试～

希望能得到大家的反馈，也希望这个插件能真正的在日常开发过程中给大家提效

感谢两个同学一直协助开发@余焕柱@魏昕明～

## Lint

上面预处理阶段说了很多由于设计稿「不规范」导致我们不得不做的算法 ... 还有很多是我们没办法做到的，需要依赖「Design Lint」来帮助设计师产出更「高质量」的设计稿，如「响应式是否出错」「图片导出是否合理」等 ...

这里的「不规范」「高质量」均针对协作而言，单从设计本身出发其实不存在这些问题 ... 毕竟设计的本质工作也不涵盖「用研发的思维去组织设计稿 ...」

而且两个角色的思维模式是有差异的，设计师在画稿的过程是「发散」的，而研发拿到设计稿之后却是一个「收敛」的过程。因此没理由要求设计师在画设计稿的时候就要保证怎样的结构 ...



## 后续规划

- 平台能力完善
  - ui 改版
  - 解析错误的修正
  - 在产物基础上进行增强，如 state, userActions(hover...)
  - ...
- 引入 AI 能力
  - classname 语义化推测 (
  - 背景图合并
  - ...
- 算法优化 & 能力完善
  - 减少冗余结构
  - 复用部分样式
  - 响应式推测
  - ...
- 更多的产物配置

- 如多端代码， lynx， ios...
  - Sass, less...
  - ...
- Lint
    - 更多的检测能力
    - ...