

PURSUIT工业化前端架构

PURSUIT工业化前端架构

网易蜂巢的前端架构实践

自我介绍

- ◎ 赵雨森
- ◎ 杭州研究院 - 云计算平台产品部
- ◎ 2014年入职，毕业于西安交通大学
- ◎ 闷骚男，爱摄影，组件控
- ◎ [rainfore@github](#)，[hzzhaoyusen@corp](#)
- ◎ [regular-ui](#)，[pursuit-cli](#)

- PURSUIT
- 工业化
- 前端架构

PURSUIT?

PURSUIT

PURSUIT

MEAN = MongoDB + Express + Angular + Node

PURSUIT

MEAN = MongoDB + Express + Angular + Node

NGBRESM = Node + Gulp + Browserify + Regular + Stateman + Mcss

PURSUIT

MEAN = MongoDB + Express + Angular + Node

NGBRESM = Node + Gulp + Browserify + Regular + Stateman + Mcss

es6 + babel + webpack + gulp + regular + mcsc + rgui + eslint
+ stateman + jsdoc + postmark + karma + mocha + expect.js +
spritesmith + fontcustom ...

PURSUIT

MEAN = MongoDB + Express + Angular + Node

NGBRESM = Node + Gulp + Browserify + Regular + Stateman + Mcss

es6 + babel + webpack + stateman + jsdoc + p + mcsc + rgu + eslin +
spritesmith + fontcusto + mocha + expect.js +



PURSUIT

MEAN = MongoDB + Express + Angular + Node

NGBRESM = Node + Gulp + Browserify + Regular + Stateman + Mcss

es6 + babel + webpack + stateman + jsdoc + p + mcscs + rgui + eslint + mocha + expect.js + spritesmith + fontcusto



PURSUIT = webPack + gUlp + Regular + mcsS + rgUI + eslinT

PURSUIT & Regular UI

PURSUIT & Regular UI

- 都是我写的。。

PURSUIT & Regular UI

- 都是我写的。。
- PURSUIT是组件化架构

PURSUIT & Regular UI

- 都是我写的。。
- PURSUIT是组件化架构
- pursuit-cli和rgui-tools同出一源

PURSUIT & Regular UI

- 都是我写的。。
- PURSUIT是组件化架构
- pursuit-cli和rgui-tools同出一源
- 好基友关系，使用起来很融洽

PURSUIT & Regular UI

- 都是我写的。。
- PURSUIT是组件化架构
- pursuit-cli和rgui-tools同出一源
- 好基友关系，使用起来很融洽

regular-ui v0.2-beta

PURSUIT == 追求

PURSUIT == 追求

不断追求前端技术，紧跟时代发展潮流。

[OVERVIEW](#)[INSTALL](#)[TUTORIAL](#)[SHOWCASE](#)[PACKAGES](#)[ABOUT](#)

A Fast, Advanced and Componentized

Front-end Framework for Your Next Web Application

[VIEW ON GITHUB](#)

工业化？

标题党

有逼格

炒概念

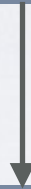
前端发展的几个阶段

前端发展的几个阶段

刀耕火种

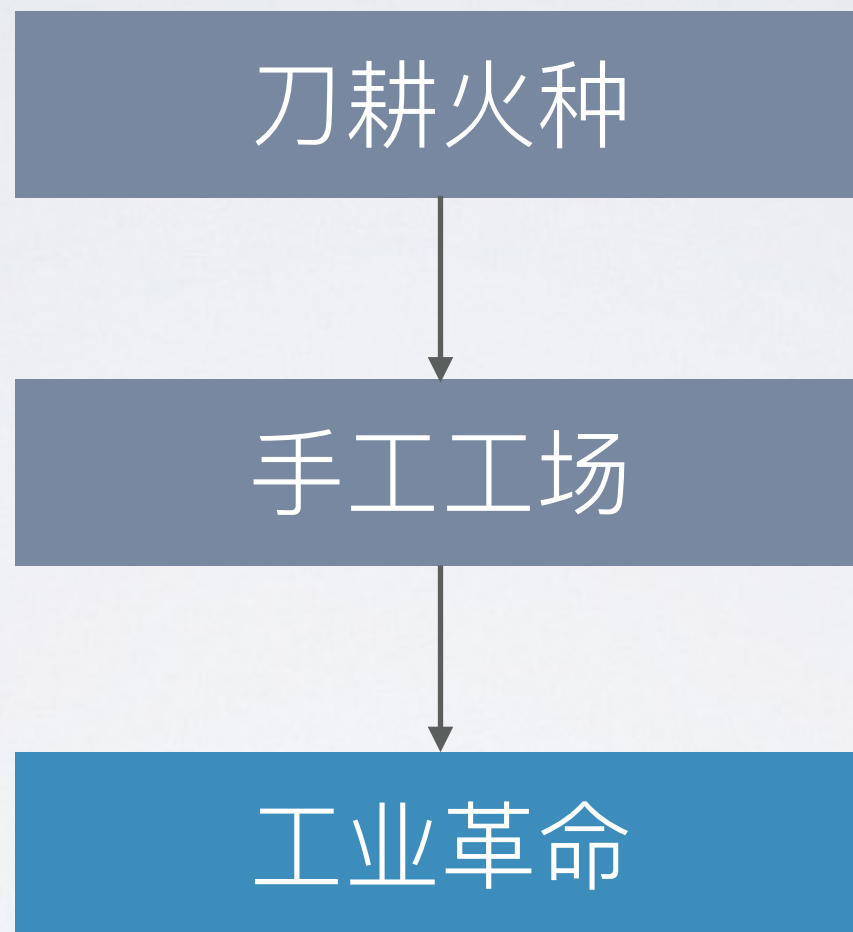
前端发展的几个阶段

刀耕火种

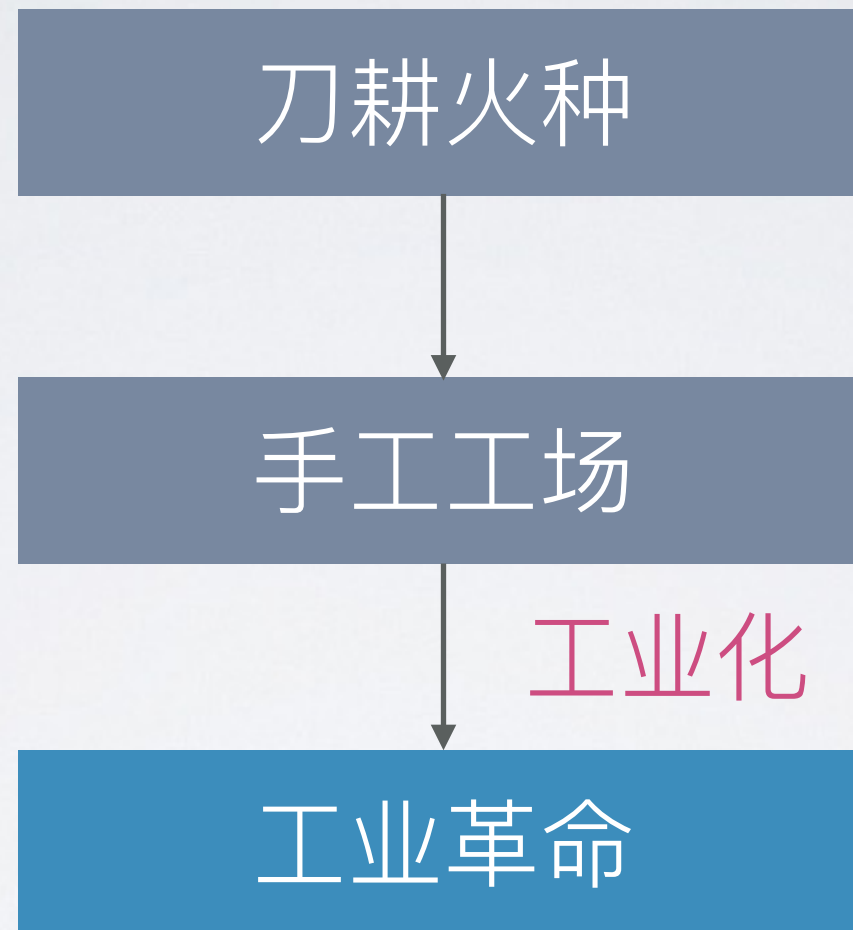


手工工场

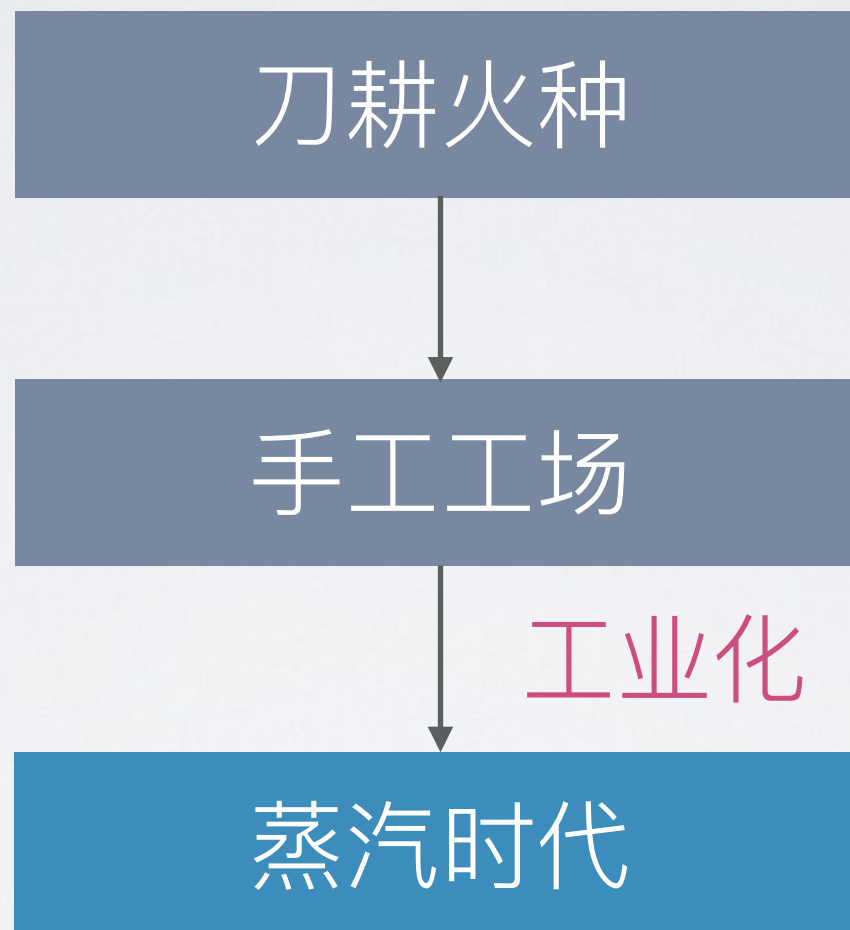
前端发展的几个阶段



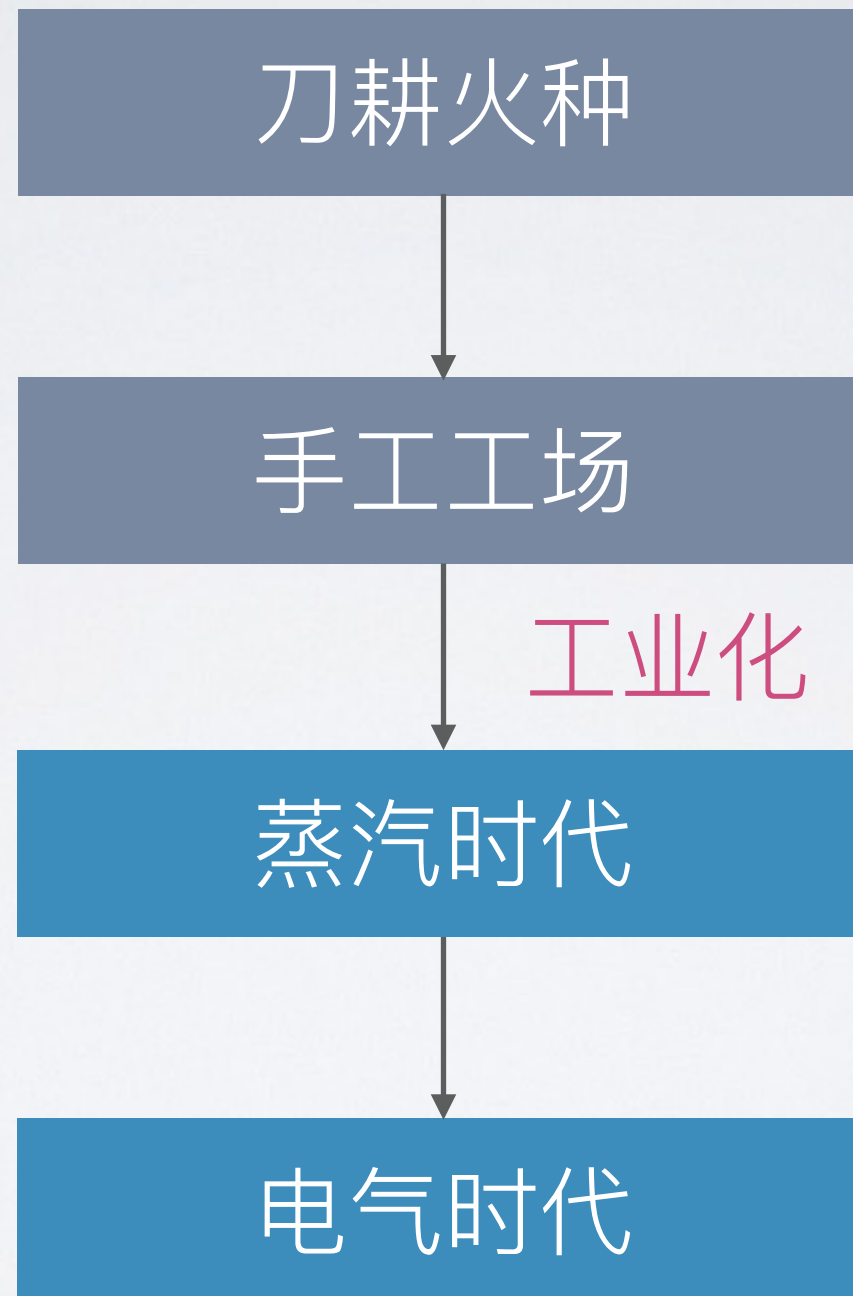
前端发展的几个阶段



前端发展的几个阶段



前端发展的几个阶段



工业化的四个方面

模块化

组件化

规范化

自动化

模块化

模块化的概念

将一个大文件拆分成相互依赖的小文件，再进行统一的拼装或加载。

JS的模块化

- ◎ Java有import
- ◎ C++有include
- ◎ Ruby有require
- ◎ CSS有@import
- ◎ JS没有

JS的模块化

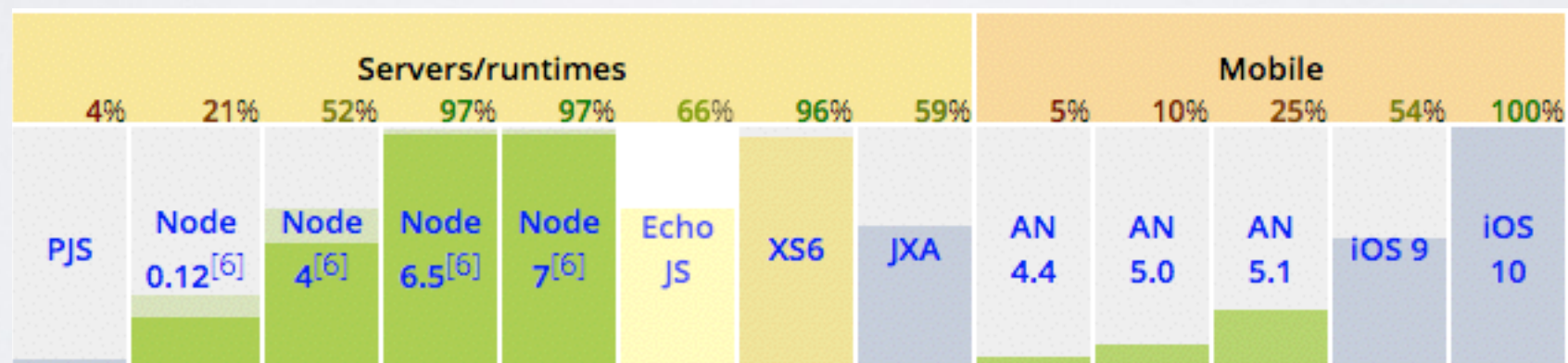
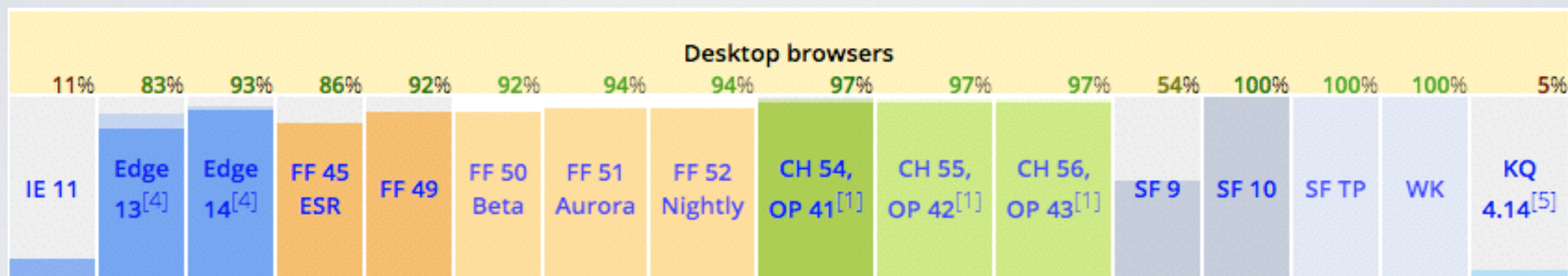
- CommonJS
- AMD
- CMD
- UMD

```
(function (root, factory) {  
    if (typeof exports === 'object')  
        module.exports = factory();  
    else if (typeof define === 'function' && define.amd)  
        define(factory);  
    else  
        root[library] = factory();  
})(this, function () {  
    //module ...  
});
```

ES6的模块体系

- import
- import ... as
- export
- export default

各平台对ES6的支持率



[combat-table](#)

兼容性处理

- 如果使用的是Babel：
 - IE6、7：就不用考虑了
 - IE8：
 - 使你的 React 应用兼容 IE8
 - export-all-loader(export * from 'xxx';)
 - IE9：只需吃个babel-polyfill即可

模块的打包与加载

- Webpack
- System.js

技术选型: Webpack + Babel + ES6



后续研究: System.js

CSS的模块化

- ◎ SASS
- ◎ LESS
- ◎ Stylus
- ◎ MCSS

痛点

选择器的私有化问题（全局污染问题）

各厂的命名风格

- ◎ BEM风格；
- ◎ Bootstrap风格；
- ◎ Semantic UI风格；
- ◎ 我们公司的NEC风格；
- ◎ ...

各厂的命名风格

- ◎ BEM风格；
- ◎ Bootstrap风格；
- ◎ Semantic UI风格；
- ◎ 我们公司的NEC风格；
- ◎ ...

弱约束

“与其费尽心思地告诉别人要遵守某种规则，以规避某种痛苦，倒不如从工具层面就消灭这种痛苦。”

——知乎段子手

工具层面的三种解决方案

- Shadow DOM
- CSS in JS
- CSS Modules

技术选型: MCSS -> PostCSS
后续研究: CSS Modules

组件化

1. 组件化 \neq 模块化

- 模块化是语言层面的
- 组件化是设计层面的

组件

每个包含模板(HTML)+样式(CSS)+逻辑(JS)功能完备的结构单元，我们称之为**组件**。

组件化要解决的问题

- 组件封装
- 逻辑 (JS) 继承
- 样式 (CSS) 扩展
- 模板 (嵌套) 嵌套

组件化要解决的问题

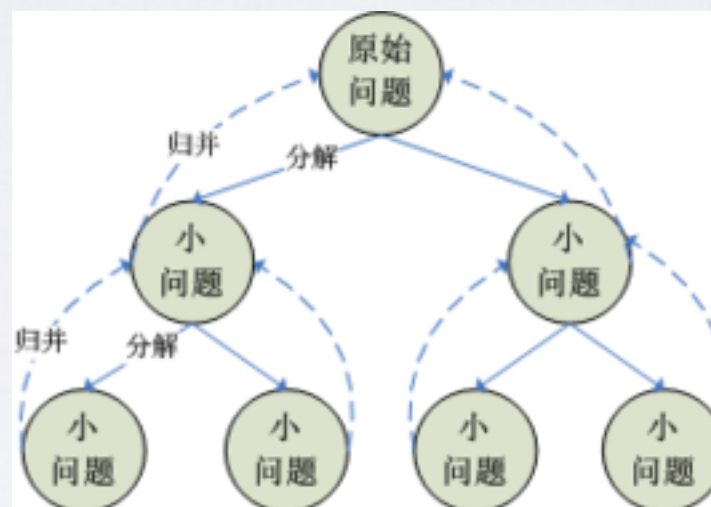
- 组件封装
- 逻辑 (JS) 继承
- 样式 (CSS) 扩展
- 模板 (嵌套) 嵌套

2. 组件化是对面向对象的更高级的抽象

“Keep Simple. Everything can be a component.”

——React

3. 分治（分而治之）思想



思想上的区别

- 传统框架/类库是DOM优先
- 组件化框架是组件优先

标签化

- 一个标签可以代表一个组件
- 解析模板即可知道页面全貌
- 为可视化前端开发提供了可能

```
<pager current="1" total="8" />
```


目录结构的改进



```
component/  
  demo/  
    index.md      # 组件文档  
  test/  
    spec.js       # 单元测试  
  index.mcss      # 组件样式  
  index.js        # 组件逻辑  
  index.rgl       # 组件模板  
  index.json      # 组件信息
```


目录结构的改进

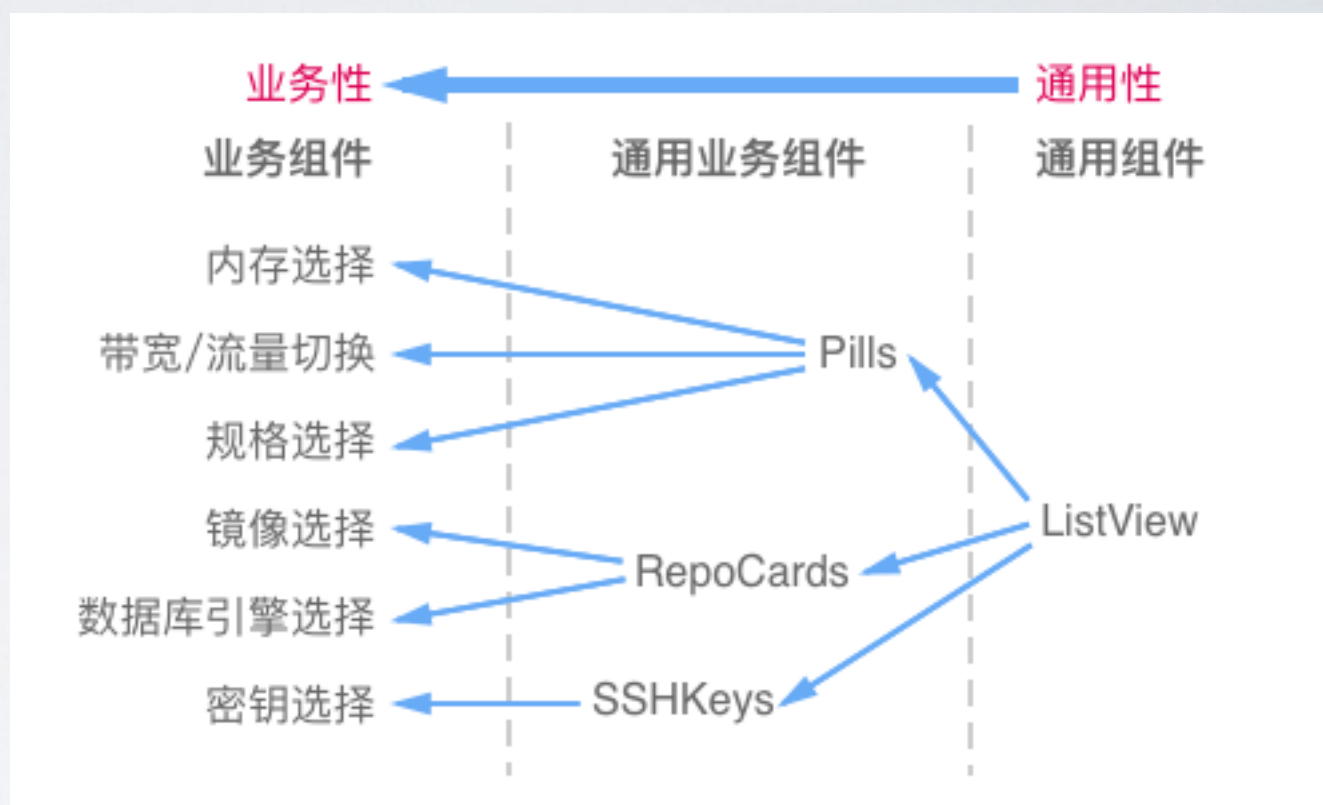


```
component/  
  demo/  
    index.md      # 组件文档  
  test/  
    spec.js       # 单元测试  
  index.mcss      # 组件样式  
  index.js        # 组件逻辑  
  index.rgl       # 组件模板  
  index.json      # 组件信息
```

原则：一个组件一个目录

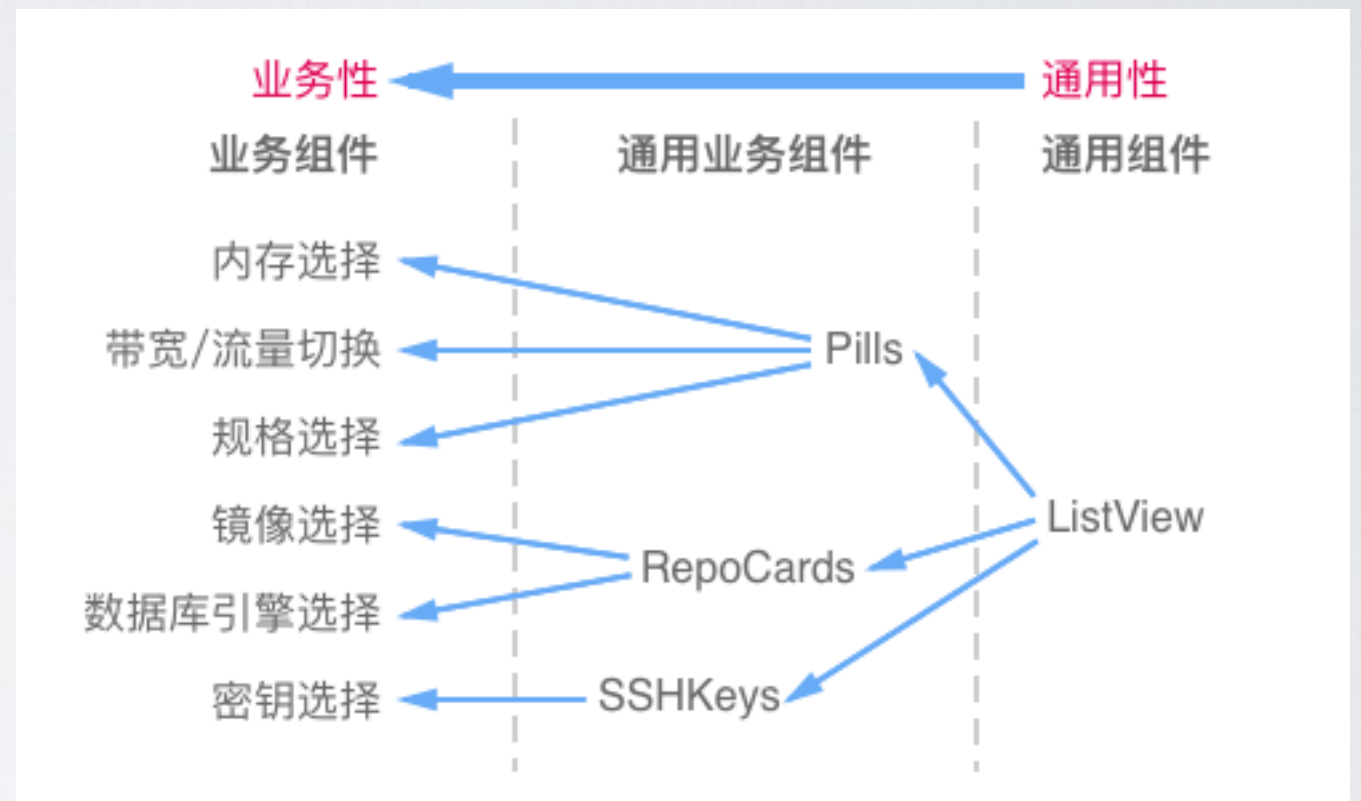
组件的分类

- 通用组件
- 通用业务组件
- 业务组件



组件的分类

- 通用组件
- 通用业务组件
- 业务组件



原则：在不增加组件配置复杂度的情况下，尽可能提高组件的通用性。


目录结构的改进

src/	# 源文件
base/	# 基础类库
common/	# 通用组件和通用业务组件
specific/	# 可复用的业务组件
module/	# 模块组件或者一次性的业务组件
page/	# 页面及入口css和js
service/	# 数据服务
icons/	# 图标源文件
png/	# 雪碧图标源文件
svg/	# 字体图标源文件
assets/	# 静态文件
dest/	# 生成文件
css/	
js/	
img/	
fonts/	# 图标字体和专用字体
vendor/	# 第三方库

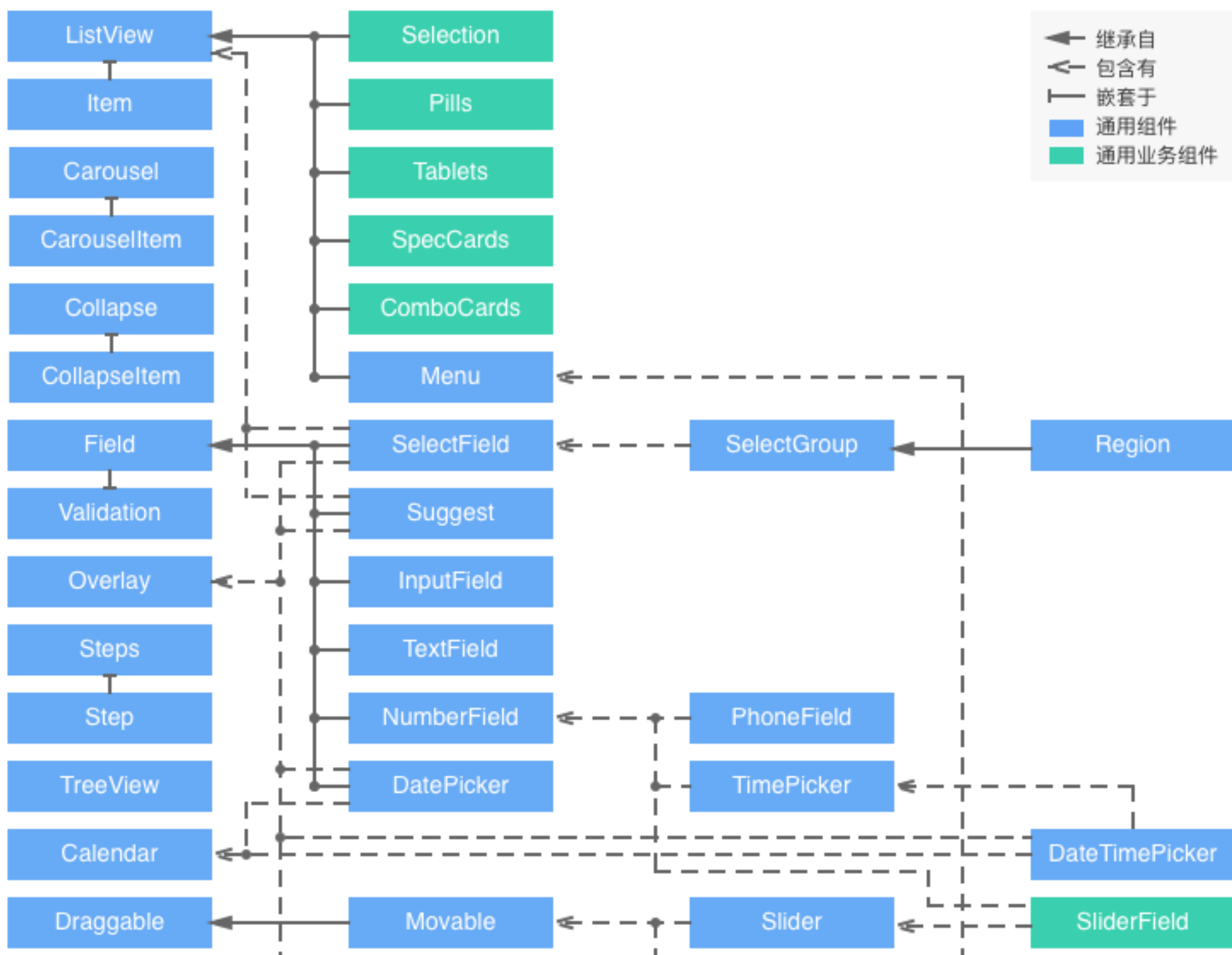
组件之间的关系

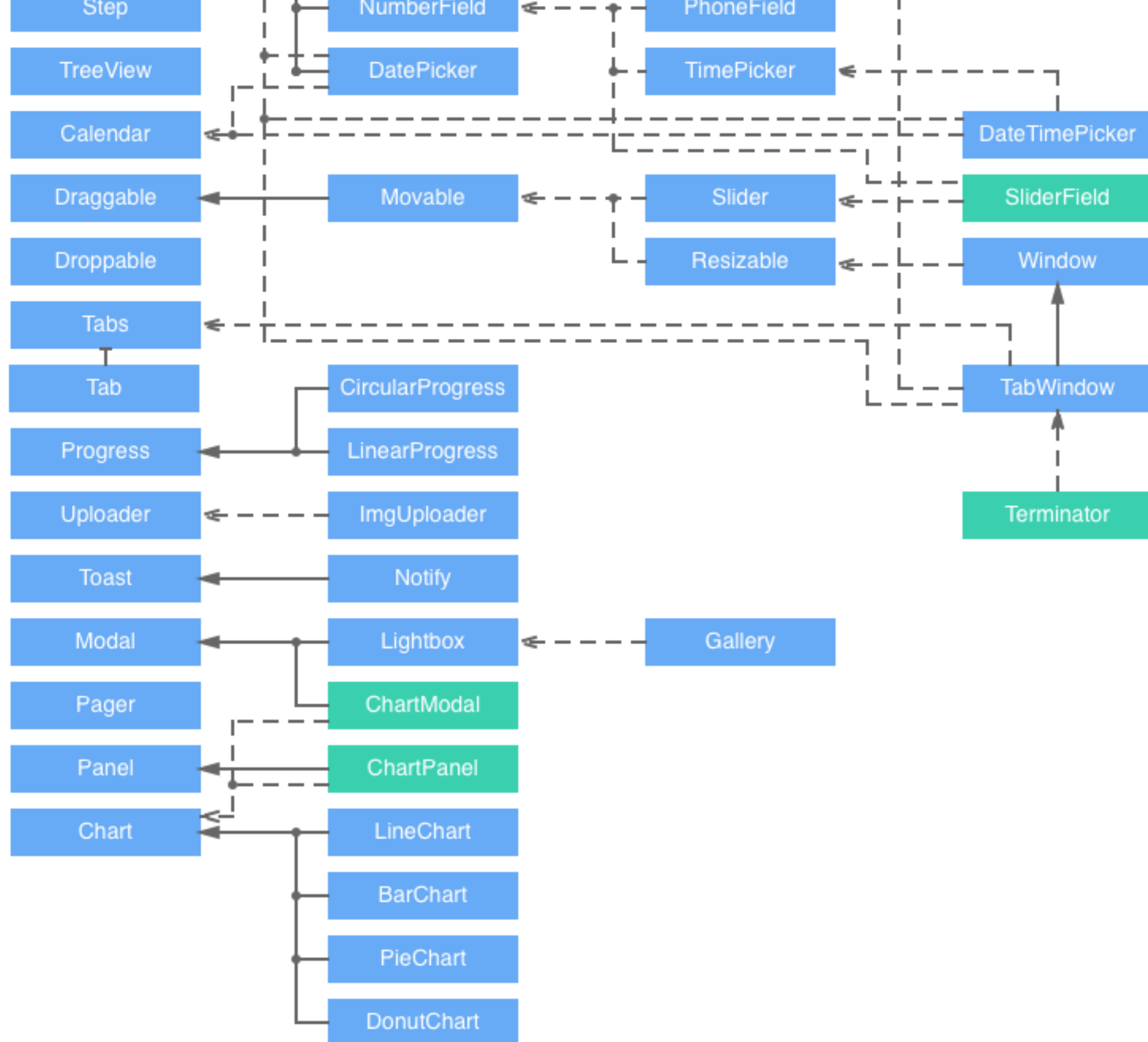
- 继承
- 扩展
- 嵌套
- 包含

组件之间的关系

- 继承
 - 扩展
 - 嵌套
 - 包含
- 
- 依赖

蜂巢组件（部分） 依赖关系图





组件库

组件库是一系列组件的集合。



技术选型: RegularJS + Regular UI + Cloud UI

规范化

编码规范

- 命名规范
- CSS/MCSSS编码规范
- JavaScript编码规范
- Regular组件设计规范

ESLint

- ⦿ 花一天时间先遍历ESLint的437条规则；
- ⦿ 筛选需要的规则，并在组内讨论；
- ⦿ 先将确定的规则全部配error，然后再根据情况降级；
- ⦿ lint存在error时禁止提交代码。

技术选型: ESLint

后续研究: StyleLint



自动化

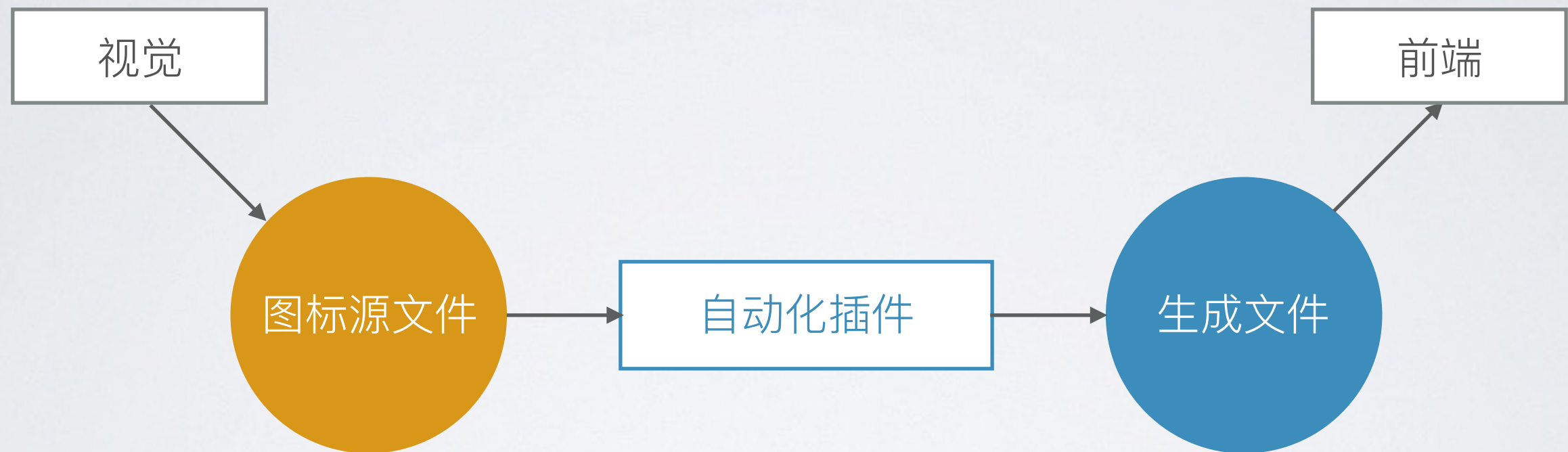
“任何简单机械的重复劳动都应该让机器去完成。”

——Yusen Zhao

手工合并图标

- 雪碧图用PS手动拼接
- 字体图标用Icomoon管理

自动化合并图标



技术选型: SpriteSmith + FontCustom



可视化组件文档

- PostMark
- JSDoc

技术选型：PostMark + JSDoc



前端自动化测试

- 单元测试
- UI测试

维护自动化测试成本最低原则

- ◎ 基础类库的单元测试；
- ◎ 通用组件和通用业务组件的单元测试；
- ◎ 通用组件和通用业务组件简单的UI测试；
- ◎ 主流程的UI测试。

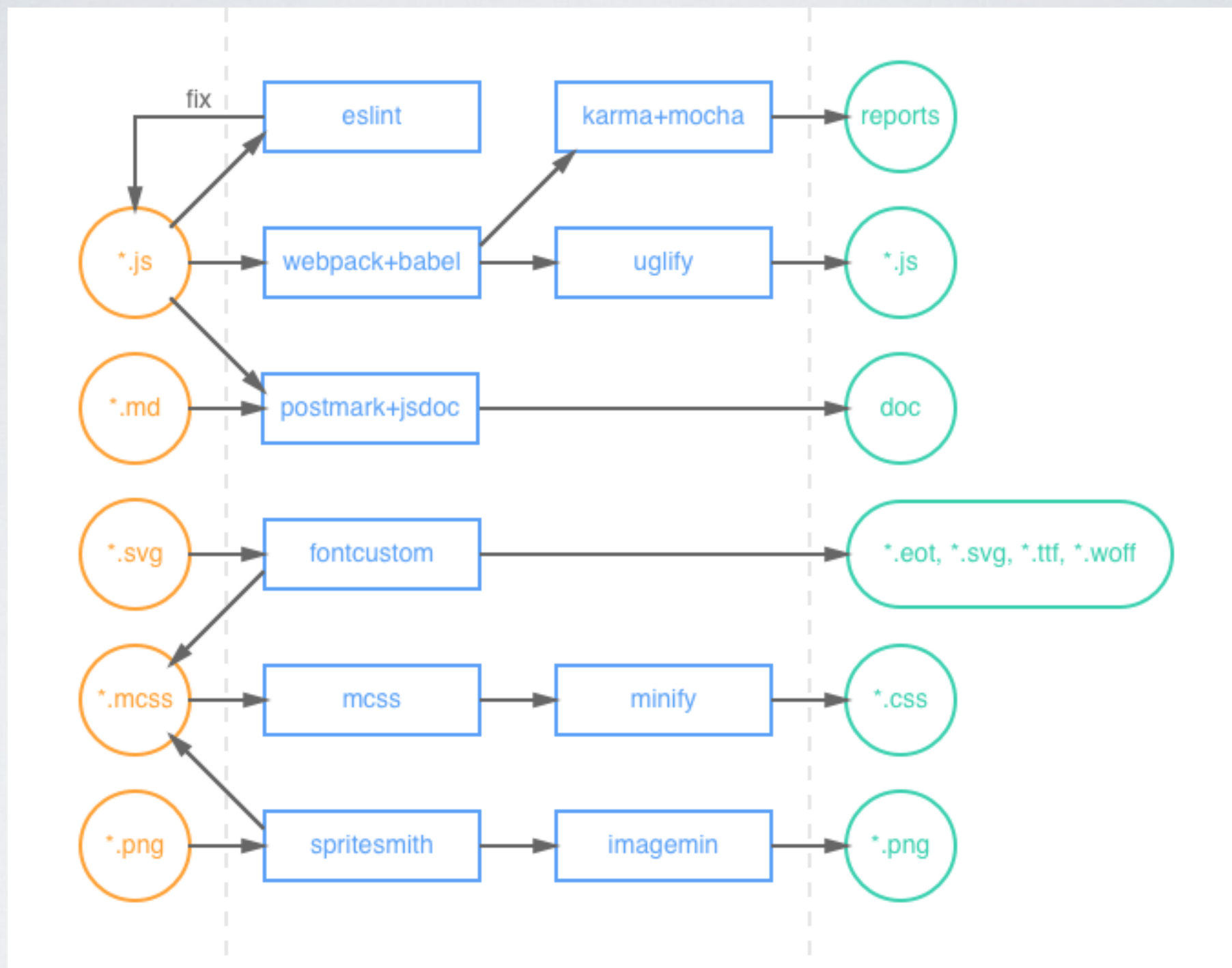
技术选型: Karma + Mocha + Expect.js



构建工具

pursuit-cli

技术选型：Gulp



- ◎ `pursuit dev`: 构建项目+生成文档+验证代码风格
- ◎ `pursuit online`: 生成图标->构建项目->运行测试->验证代码风格



前端架构

需要关注的问题

- ◎ 生产效率
- ◎ 性能优化
- ◎ 前端统计
- ◎ 健壮性
- ◎ 伸缩性
- ◎ 安全性
- ◎ ...

生产效率

- ◎ 模块化
- ◎ 组件化
- ◎ 规范化
- ◎ 自动化

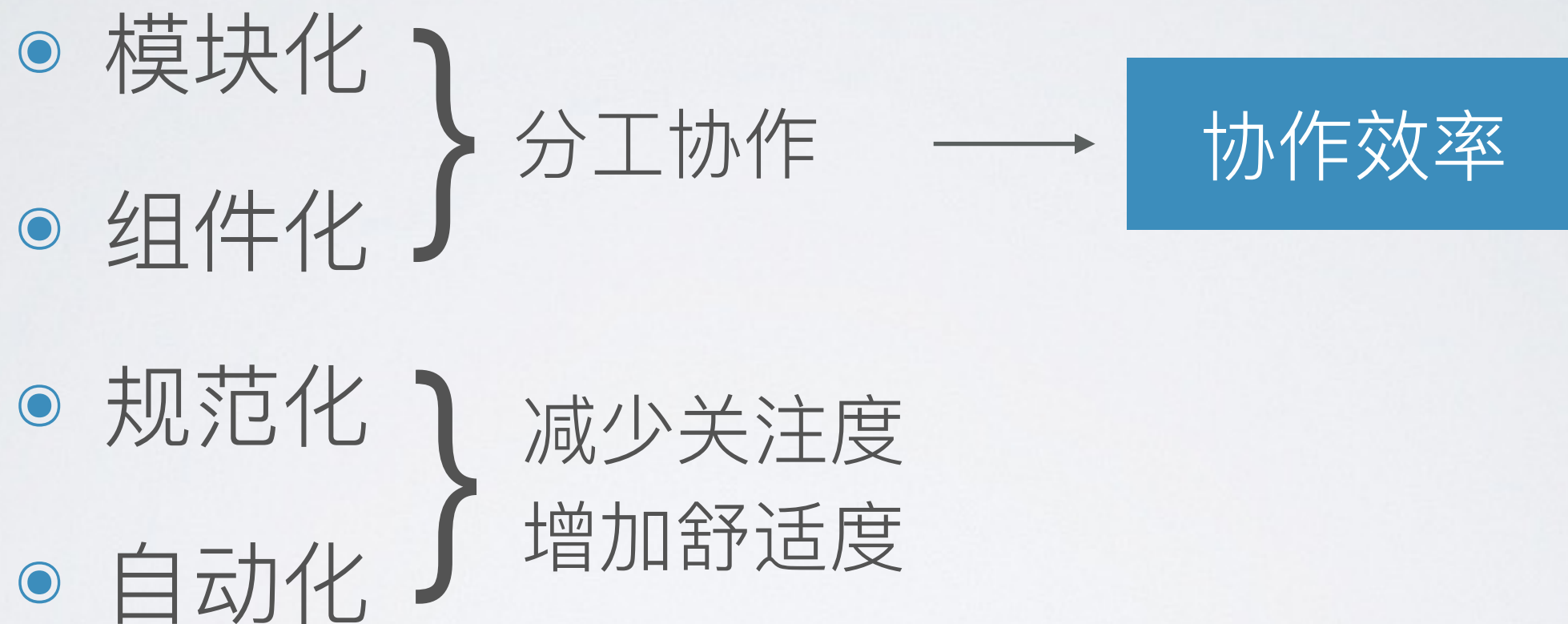
生产效率

- ④ 模块化
 - ④ 组件化
 - ④ 规范化
 - ④ 自动化
- } 分工协作

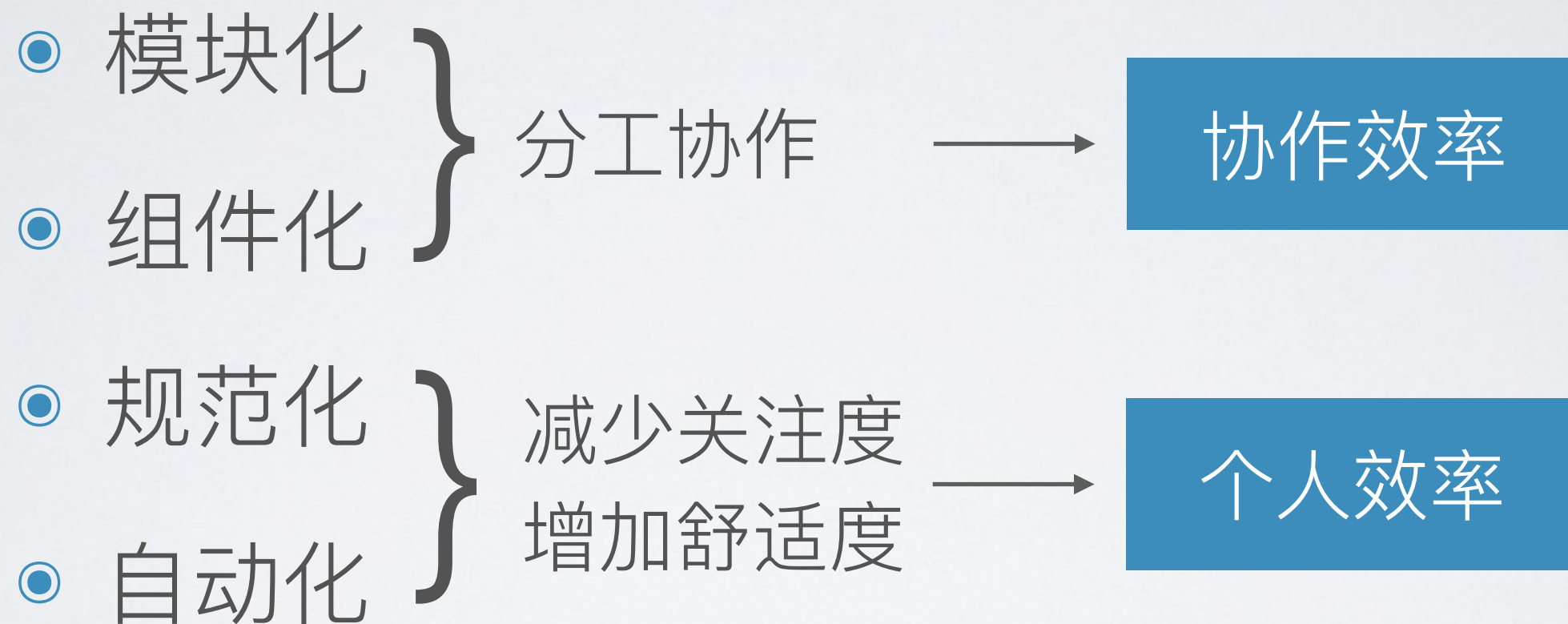
生产效率



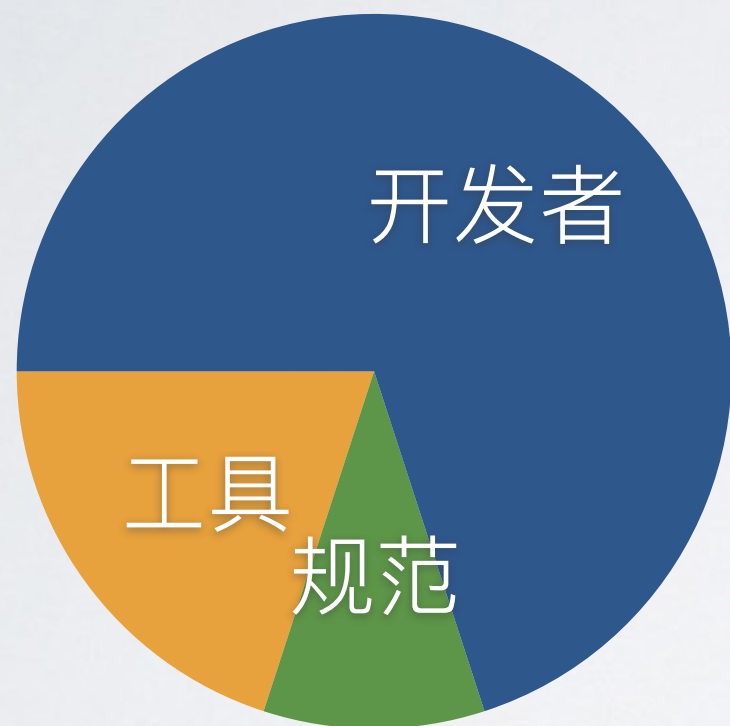
生产效率



生产效率



关注度



关注度



QUESTIONS



THANKS