D2 前端技术论坛
D2 FRONTEND TECHNOLOGY FORUM
精心

第十六届D2前端技术论坛

Formily核心设计思路

白玄

自我介绍

花名　　姓名

白玄　王智力

来自阿里数字供应链事业部

业务侧

数字供应链合同域相关的业务功能

横向团队

横向基础前端架构，包括River Builder低代码产品的前端架构

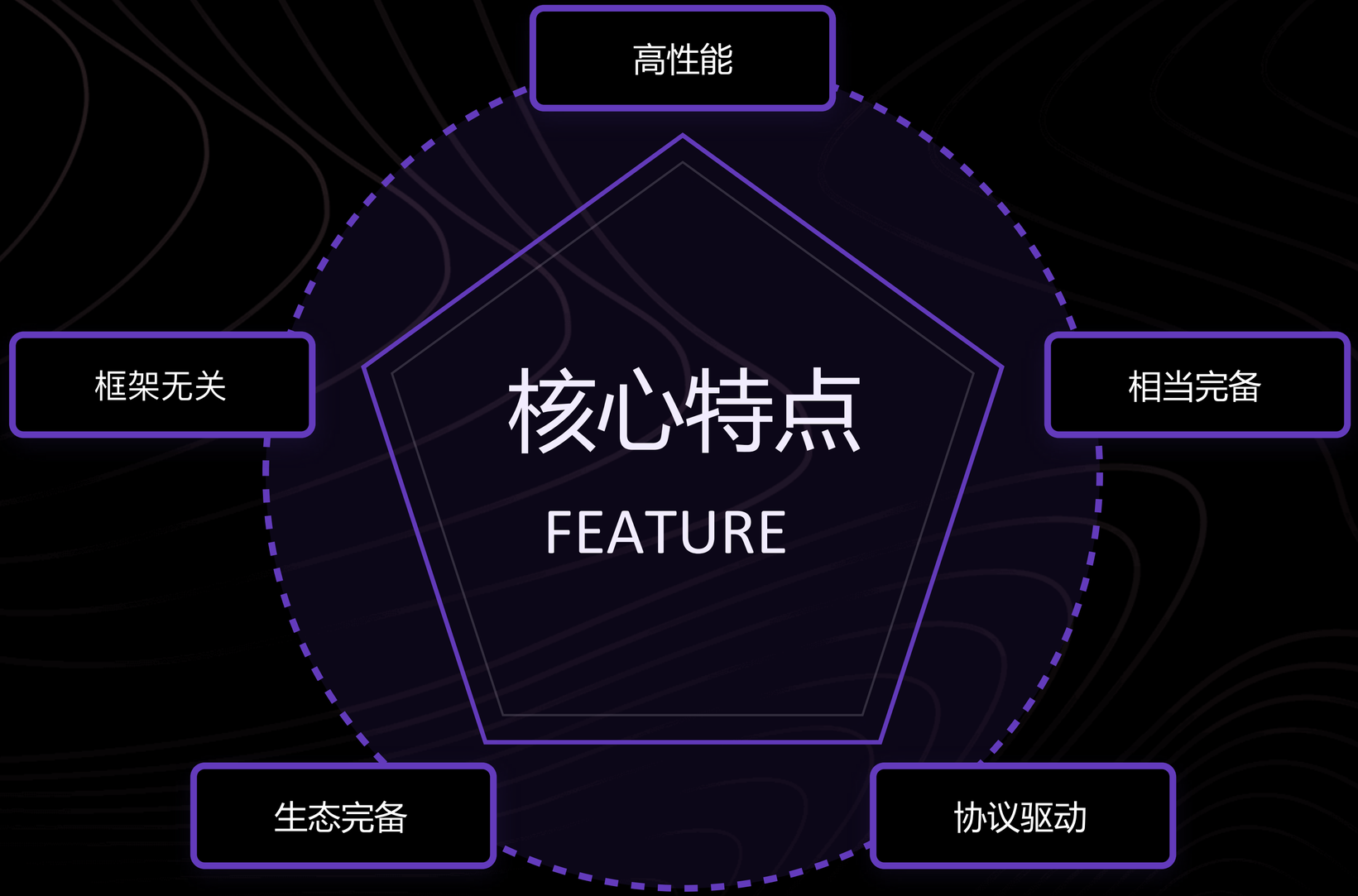
SELF-INTRODUCTION

Contents
目录

什么是Formily?

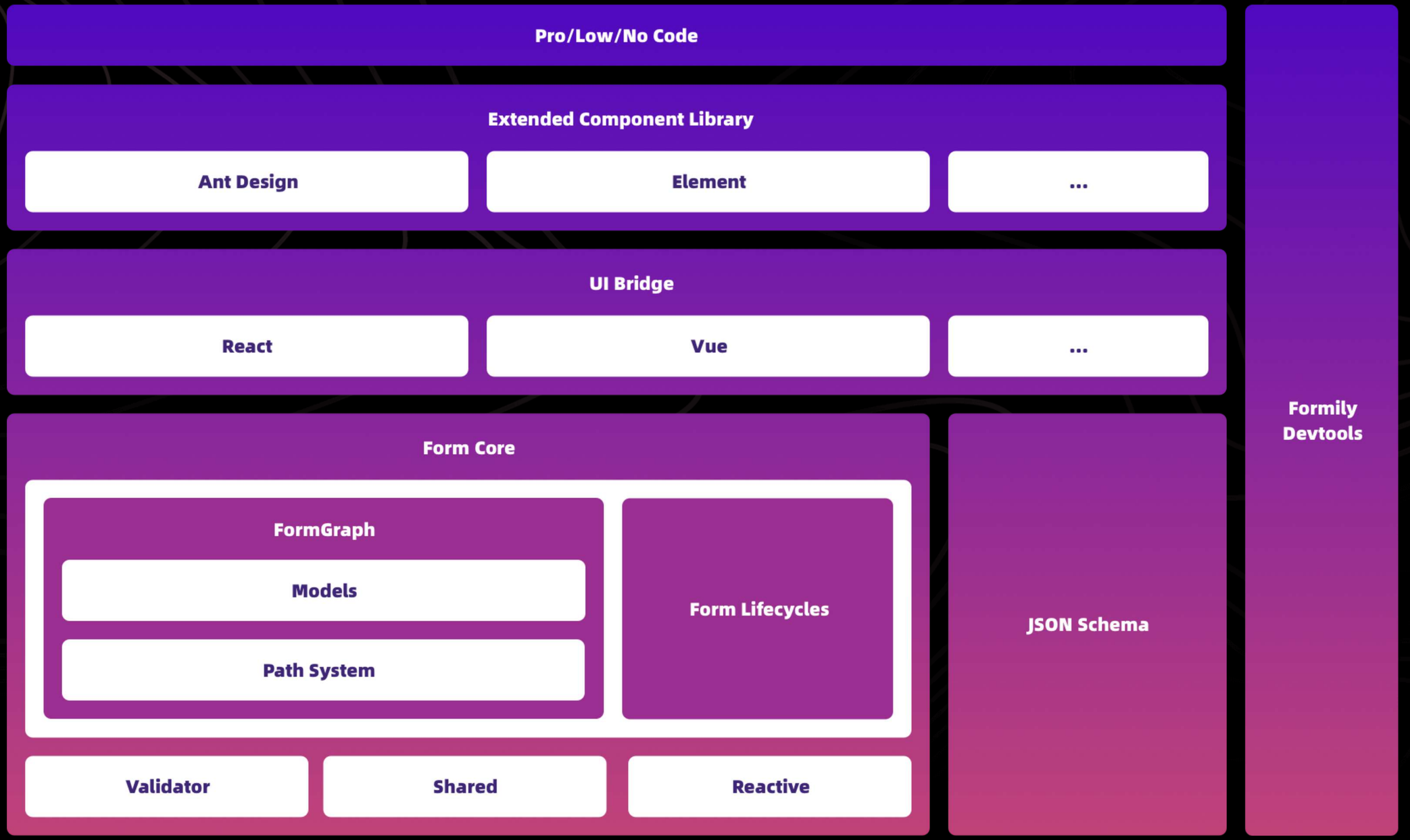
是一款面向中后台复杂场景的 数据+协议 驱动的表单框架
它也是阿里巴巴集团统一表单解决方案

域名 https://formilyjs.org


高性能
框架无关
相当完备
核心特点
FEATURE
生态完备
协议驱动


单一
职责
设计
原则
优雅命名
简单
直观
一致
DESIGN PHILOSOPHY

核心架构


Pro/Low/No Code
Extended Component Library
Ant Design
Element
...
UI Bridge
React
Vue
...
Form Core
FormGraph
Models
Path System
Form Lifecycles
Validator
Shared
Reactive
JSON Schema
Formily
Devtools

02

设计思路

高性能思路

设计思路

问题

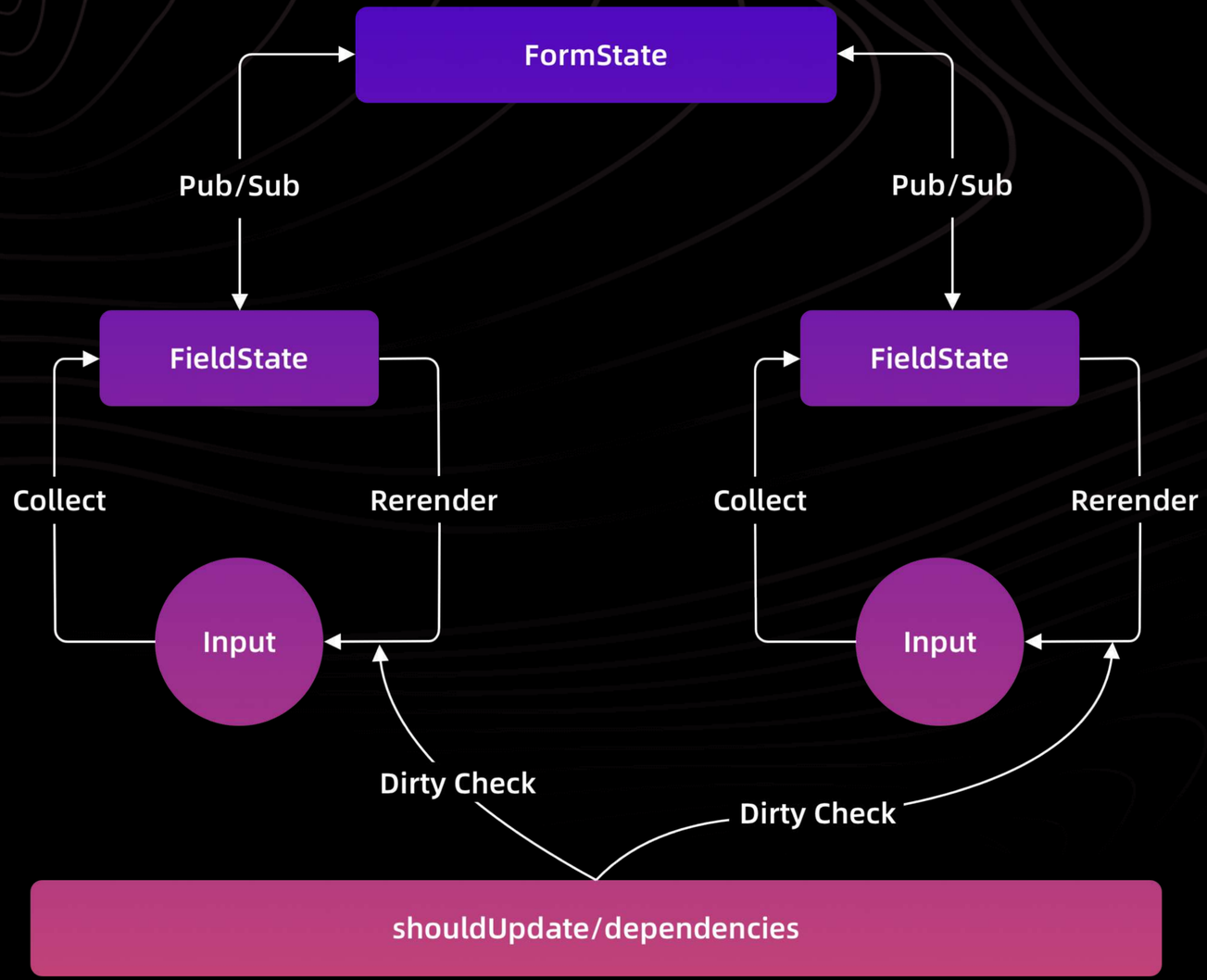
普通React用法，想要实现数据收集和同步，需要强依赖虚拟DOM的整树重绘才能达到目的。

目标

不管字段数量多少，高频输入，永远都是O(1)复杂度

方案一 —— ReactFinalForm/Antd4.0方案

抽象FormState/FieldState，内部使用pub/sub模式进行通讯，脏检查控制联动时的渲染重绘



时间复杂度

输入时：Ant Design O(1)/ReactFinalForm O(1)
联动时：Ant Design O(1 or n)/ReactFinalForm O(1 or n)

优点

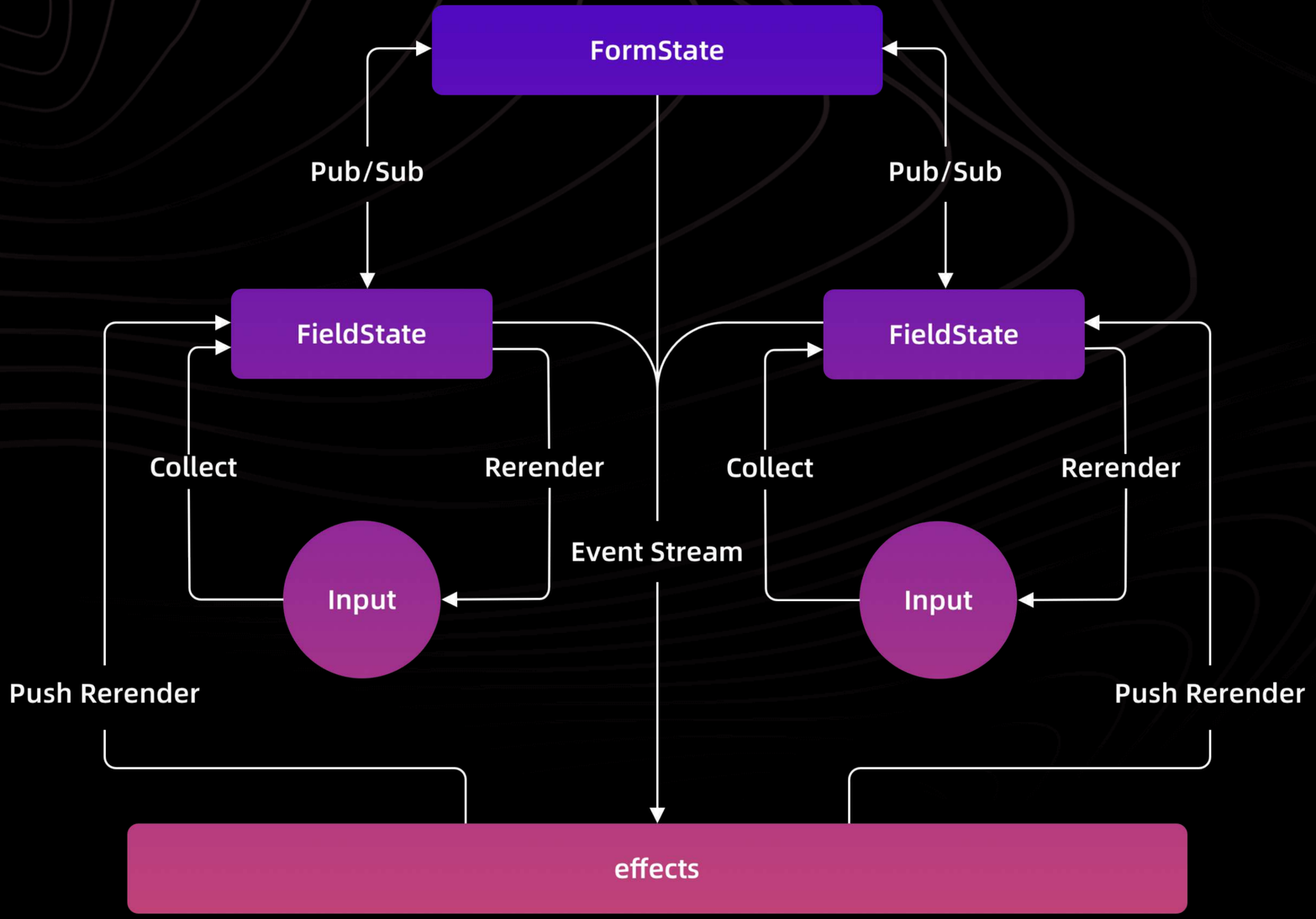
简单，易上手

问题

- 脏检查成本高
- 外部联动无法做到O(1)

方案二 —— UForm/Formily1.x方案

抽象FormState/FieldState，内部使用pub/sub模式通讯，抽象主动更新模型



时间复杂度

输入时：O(1)
联动时：O(1)

优点

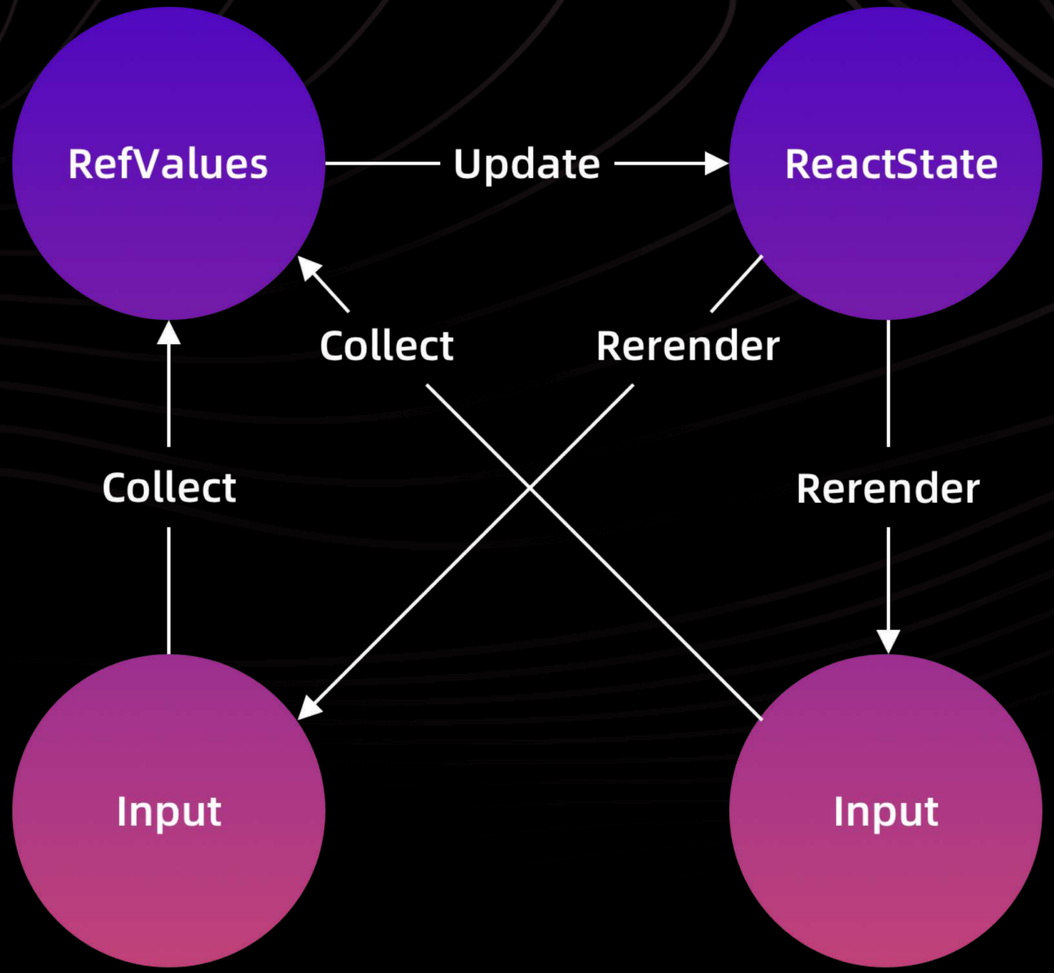
高性能

问题

- 学习成本高
- 外部联动需要转换成主动模型

方案三 —— ReactHookForm方案

基于DOM机制收集数据，绕过React受控更新机制，保证高频输入无卡顿



时间复杂度

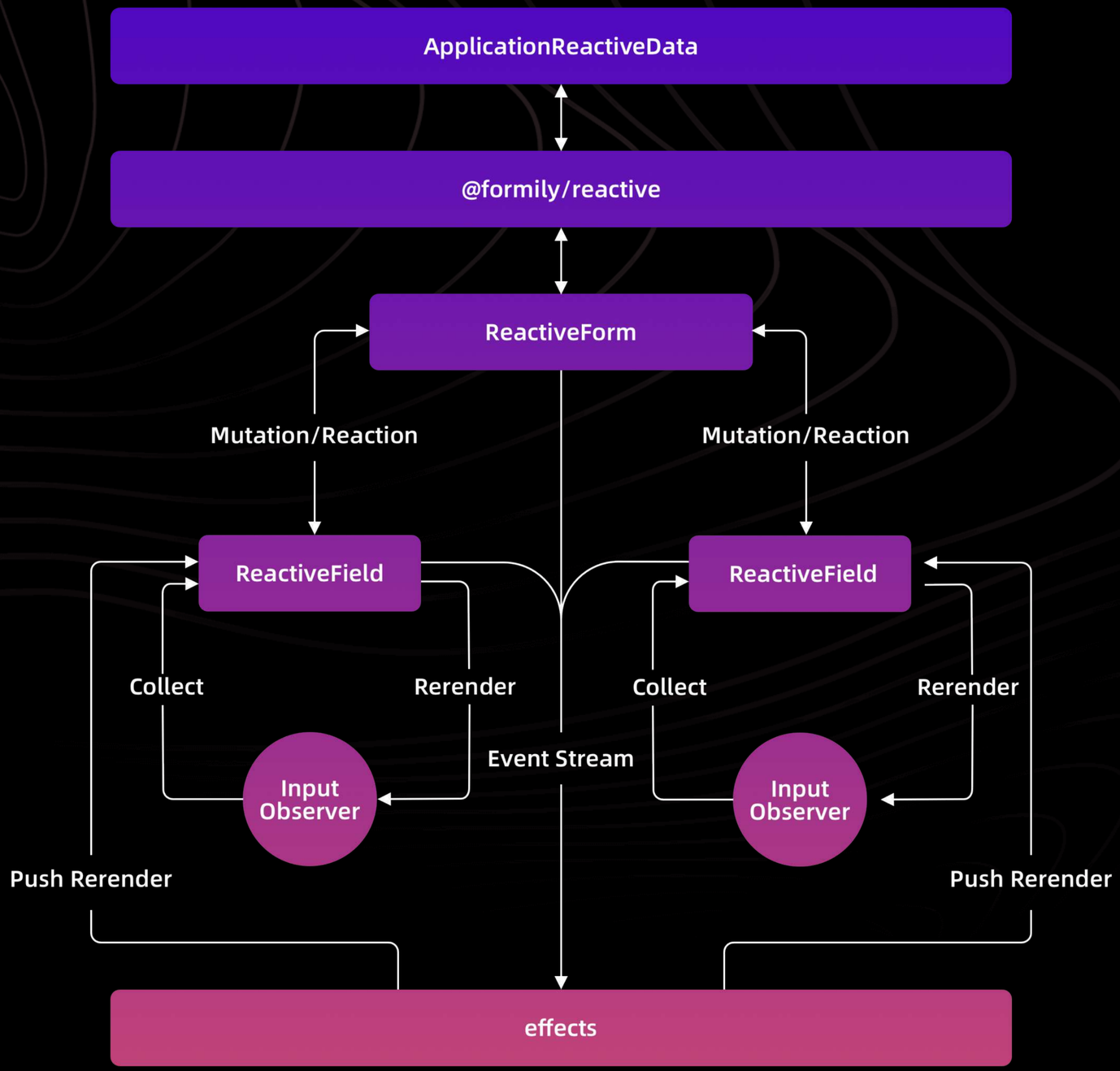
输入时：O(1)
联动时：O(n)

优点

足够轻量，简单

问题

一旦涉及数据联动，还是需要重绘整树



方案四 —— Formily2.0方案

依托Reactive响应式解决方案，构建整个响应式表单领域模型

时间复杂度

输入时：O(1)
联动时：O(1)

优点

- 不再需要大量脏检查成本，一切都是基于数据变化精确渲染
- 既支持主动更新模型，也支持被动响应模型，在多对一被动联动场景，代码量大大减少
- 外部数据与表单内部字段联动时，只需要定义ApplicationReactiveData即可，代码编写成本很低

问题

学习成本高

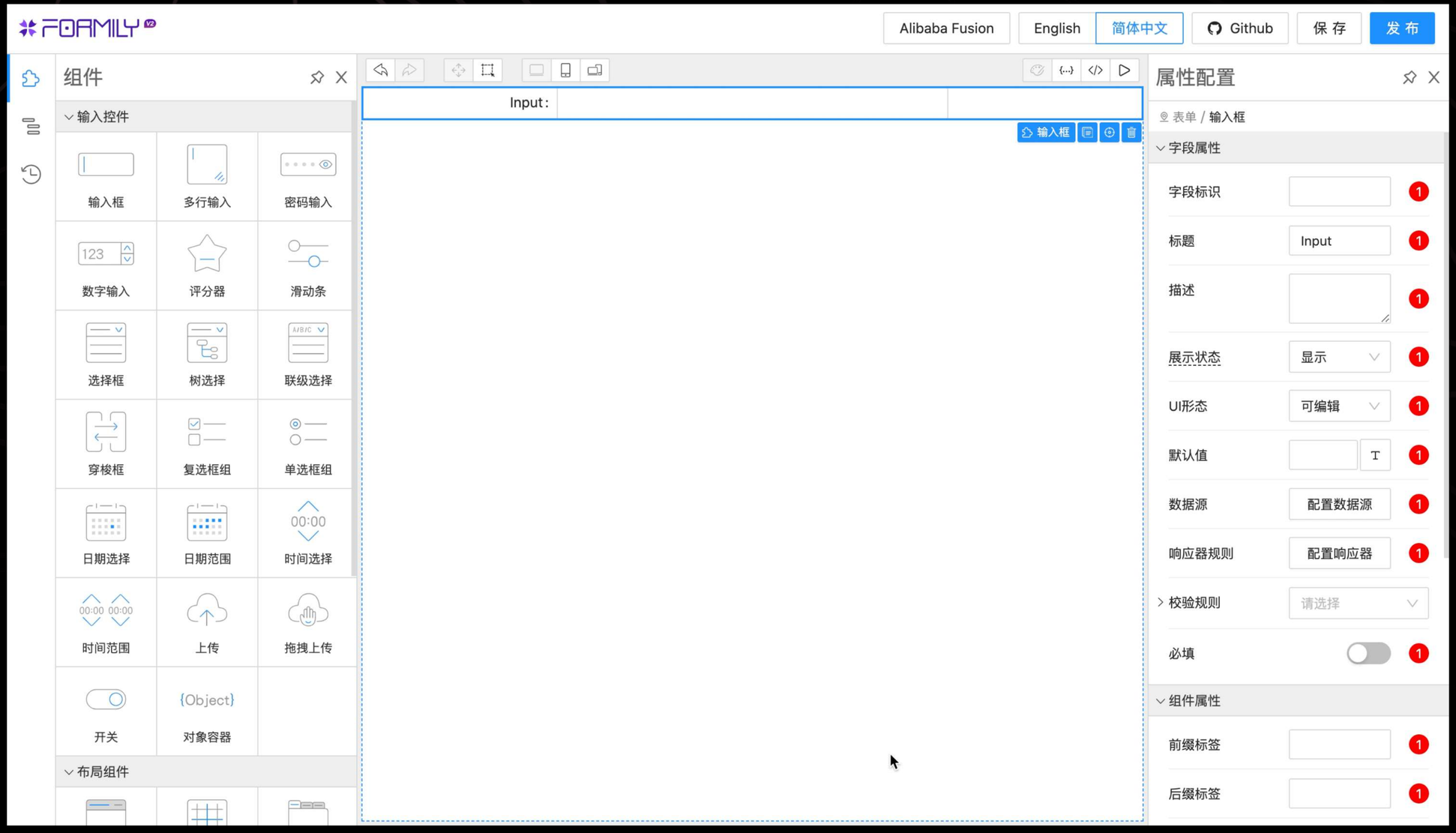
高频输入性能

时间复杂度 O(1) 复杂度

2000个字段，针对单个字段的高频输入，完全无卡顿

* name 1988: Please Input 1
* name 1989: Please Input 1
* name 1990: Please Input 1
* name 1991: Please Input 1
* name 1992: Please Input 1
* name 1993: Please Input 1
* name 1994: Please Input 1
* name 1995: Please Input 1
* name 1996: Please Input 1
* name 1997: Please Input 1

外部共享联动


FOAMILY V2
Alibaba Fusion
English
简体中文
Github
保 存
发 布
组件
输入控件
输入框
多行输入
密码输入
数字输入
评分器
滑动条
选择框
树选择
联级选择
穿梭框
复选框组
单选框组
日期选择
日期范围
时间选择
时间范围
上传
拖拽上传
开关
对象容器
布局组件
Input:
输入框
属性配置
表单 / 输入框
字段属性
字段标识
标题
Input
描述
展示状态
显示
UI形态
可编辑
默认值
数据源
配置数据源
响应器规则
配置响应器
校验规则
请选择
必填
组件属性
前缀标签
后缀标签

协议驱动思路

设计思路

问题

如何更高效的开发表单？

目标

- 定义一套通用协议，简单高效的描述表单UI/逻辑
- 对低代码搭建是友好的

方案一　纯UITree协议方案，类似于JSON版的JSX

```
{
  componentName:"Form",
  props:{},
  children:[
    {
      componentName:"Field",
      props:{
        name:"username"
      },
      children:[
        {
          componentName:"Input",
          props:{
            placeholder:"Please Input"
          }
        }
      ]
    }
  ]
}
```

优点

- 足够简单，componentName/props/children完全可以把整个UI结构给描述清楚
- 对搭建非常友好

问题

只能描述视图结构，无法描述状态模型

方案二 标准JSON Schema方案，参考ReactJSONSchemaForm

```
{
  type:"object",
  properties:{
    array:{
      "title": "Custom array of strings",
      "type": "array",
      "items": {
        "type": "string"
      }
    }
  }
}
```

优点

以数据描述视角来驱动UI渲染，基于这样的数据描述，我们可以将很多状态处理逻辑固化到渲染引擎或者组件内部。从而使得我们的协议层变得非常干净

问题

无法解决UI容器包裹问题，对搭建不够友好

方案三 Formily2.0方案，针对JSONSchema的扩展协议

```
{
  "type":"object",
  "properties":{
    "card":{
      "type":"void",
      "x-component":"Card",
      "x-index":0,
      "properties":{
         "categoryId": {
            "type":"string",
            "x-component":"Input",
            "x-index": 0,
         },
         "brandId":{
           "type":"string",
           "x-component":"Text",
           "x-component-props":{
              "style":{
                "color":"red"
              }
           },
           "x-visible":"{{$values.categoryId === 2}}"
         },
      }
    }
  }
}
```

核心思路

- 扩展Void类型描述数据无关布局容器或控件
- 扩展 x-* 属性描述UI控件与控件属性

优点

- 完全表达UI，无需第二套协议，学习成本很低
- 可以将临时状态封装在渲染引擎或者组件内部
- 对搭建更加友好，因为它已经可以把UI层级给表达出来了

扩展组件思路

设计思路

问题

如何开发协议驱动的自定义组件

目标

- 足够简单
- 符合标准组件开发最佳实践

基础扩展方案

属性约定，比如 value/onChange 组件属性约定，保证所有输入型组件可以快速接入

```
const CustomComponent = ({value,onChange})⇒{
  //渲染引擎会给自定义组件传字段值 value
  //自定义组件内部数据发生变动，可以通过 onChange 告诉渲染引擎字段值发生变化
}
```

优点

符合统一 React 属性命名规范标准，对于第三方组件接入成本很低

缺点

只能约定标准属性，value/onChange/disabled/readOnly 之类的，无法约定渲染引擎内部与扩展组件间其他数据交互

白盒进阶扩展方案

标准Hooks方案，借助ReactContext，就近寻找渲染引擎内部状态模型

```
const CustomComponent = ({value,onChange})⇒{
  const field = useField() //就近寻找当前字段模型
  const schema = useFieldSchema() //就近寻找当前字段Schema描述
}
```

优点

完全隔离了渲染引擎与组件自身属性，做到了类型更友好，写法更自然的通信模式

缺点

对接三方组件，每次都需要单独写一个包装组件，不方便快速桥接

黑盒进阶扩展方案

连接器扩展

```
import { connect, mapProps } from '@formily/react'
import { Select as AntdSelect } from 'antd'

export const Select = connect(
  AntdInput,
  mapProps(
    {
      //简单属性映射
      dataSource: 'options',
      loading: true,
    },
    (props, field) => {
      return {
        //复杂属性映射
        ...props
      }
  })
)
```

优点

- 无需重新封装组件，只需要关注属性与字段模型的映射
- 映射转换逻辑方便复用

缺点

学习理解成本较高

高级场景化扩展方案

渲染权限转移模板化方案

```
//过去：渲染引擎负责渲染子树
const CustomComponent = ({children})⇒{
  return (
    <Other>
      {children}
    </Other>
  )
}

//现在：组件负责渲染子树
const CustomComponent = ()⇒{
  const schema = useFieldSchema() //就近寻找当前字段Schema描述
  const { positionA, positionB, positionC } = parseSchema(schema) //人工解析schema
  //插槽化渲染，布局，交互，逻辑固化
  return (
    <Other>
      <RecursionField name="positionA" schema={positionA} />
      <RecursionField name="positionB" schema={positionB} />
      <RecursionField name="positionC" schema={positionC} />
    </Other>
  )
}
```

优点

可以封装出大量模板组件，模板组件内部可以将布局，交互，布局，逻辑全部固化，在实际使用过程中提效很高，比如ArrayTable/ArrayCard这种自增组件

缺点

学习理解成本较高

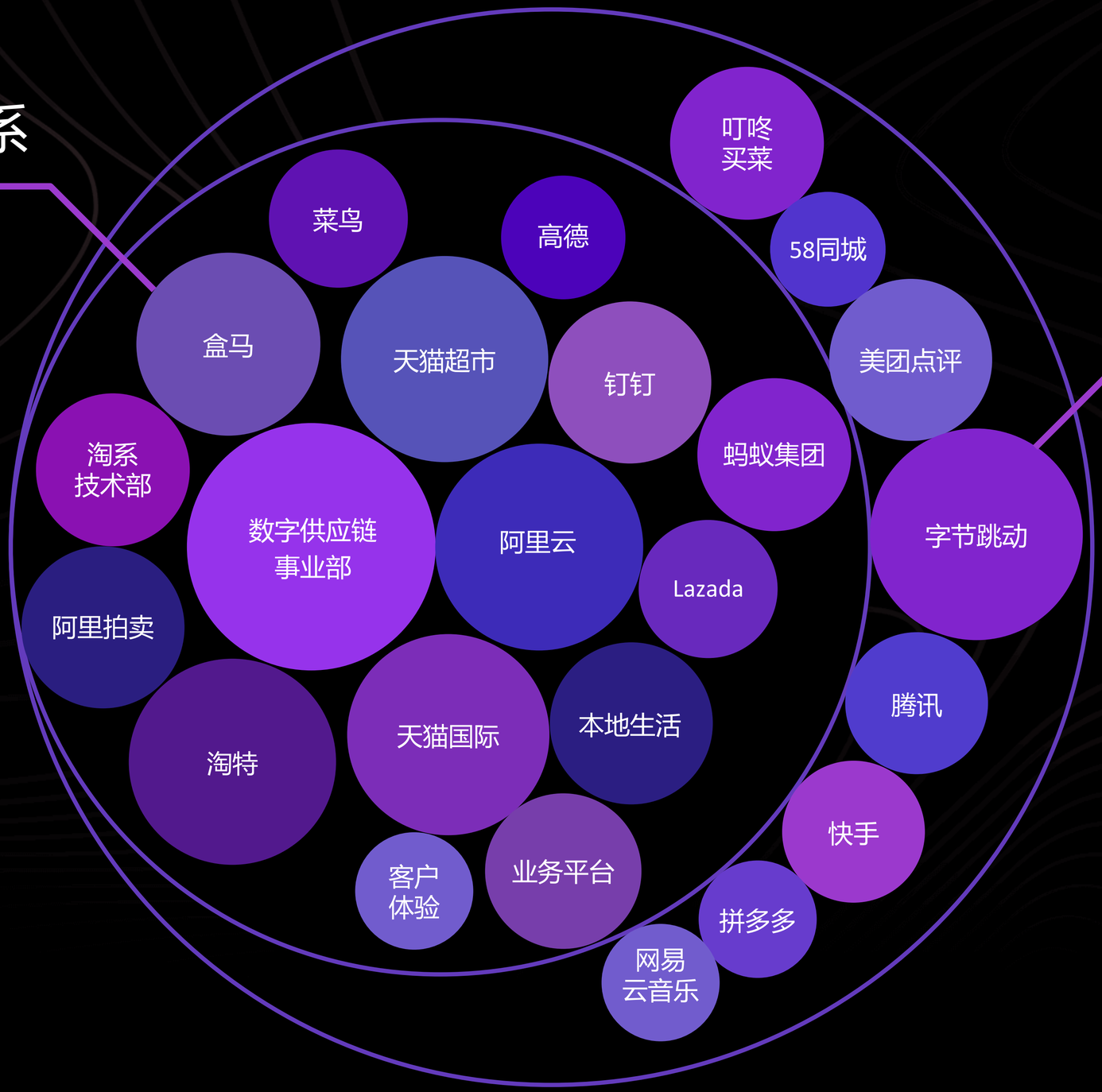
总结

Formily的扩展能力，是建立在标准组件开发最佳实践基础之上的，它是没有任何黑魔法的

03

使用数据

使用数据


阿里系
非阿里系
菜鸟
高德
叮咚买菜
58同城
盒马
天猫超市
钉钉
美团点评
淘系技术部
蚂蚁集团
数字供应链事业部
阿里云
字节跳动
Lazada
阿里拍卖
腾讯
本地生活
天猫国际
淘特
快手
客户体验
业务平台
拼多多
网易云音乐

社区运营数据

总star数

6.5k

总关闭issue数

700+

chrome插件安装量

1.6k

总fork数

762

总贡献人数

131

文档独立用户访问量

12k/月

总PR数

900+

总下载量

152k

未来规划

规划

- 进一步优化性能
- 构建更多的扩展生态
- 支持React18
- 思考如何借助Formily一次性解决CRUD页面

还有哪些不足？

- 不兼容IE，无法解决，也不想解决
- 整体体积较大，压缩前 130k左右，压缩后80k左右，包含桥接组件库
- 学习成本还是比较高，打算逐步完善视频教程

Formily的价值到底在哪里？

在 高性能 的加持下

1套底层

适配多个框架

1份Schema

适配多端

1个生态

全业务场景覆盖
(Pro/Low/No Code)

联系方式

个人微信

Formily技术交流钉钉群

数字供应链团队

数字供应链校招交流群

River Design

River Code

River Builder

River Analytics

Thanks