

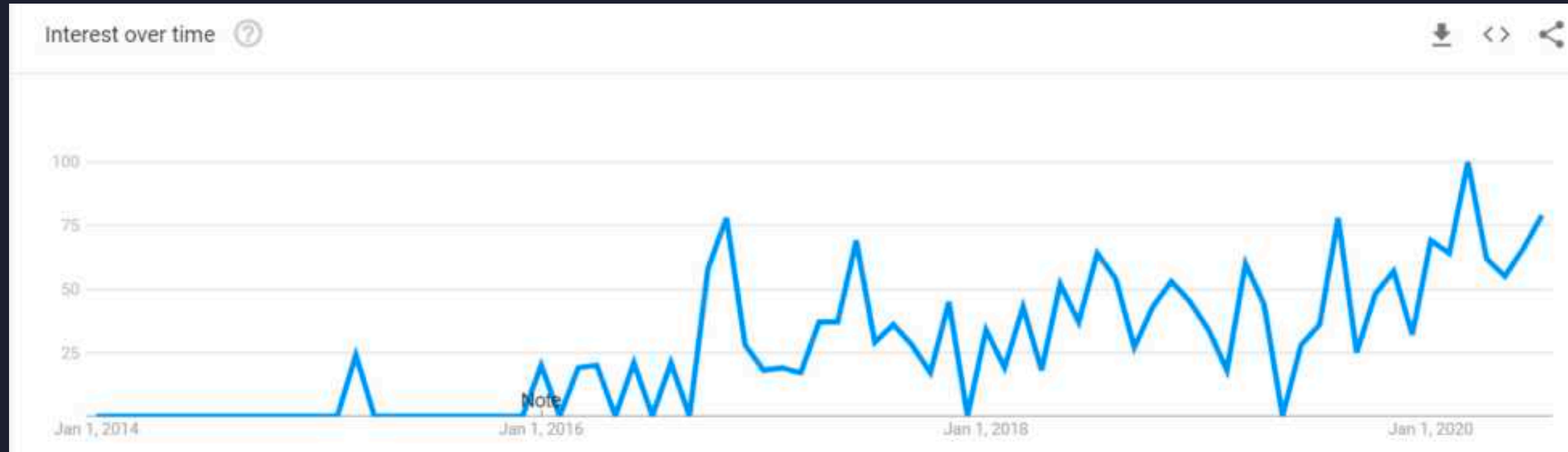
中后台低代码的思考和实践

网易云音乐前端专家 / 葛星

目录

- 1 为什么需要低代码平台?
- 2 低代码平台的核心 – UI 可视化
- 3 低代码平台的核心 – 模型驱动
- 4 低代码平台后续规划

为什么需要低代码平台- 业界的趋势



Google的搜索趋势

Gartner评估超过18家 LowCode 平台中
总规模超过8000亿美金，但中国市场刚刚起步



为什么需要低代码平台?

生产力低下与业务高速发展的矛盾

40+的工作台
随着业务的发展正不断的扩大

数据应用&公共技术

激励管理

增长引擎 (Bamboo)

...

洞察系统 (Insight)

版权评估

用户画像 (Sniper)

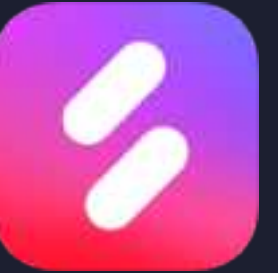
推歌系统 (Sniper)



声波



云音乐



音街



Look直播

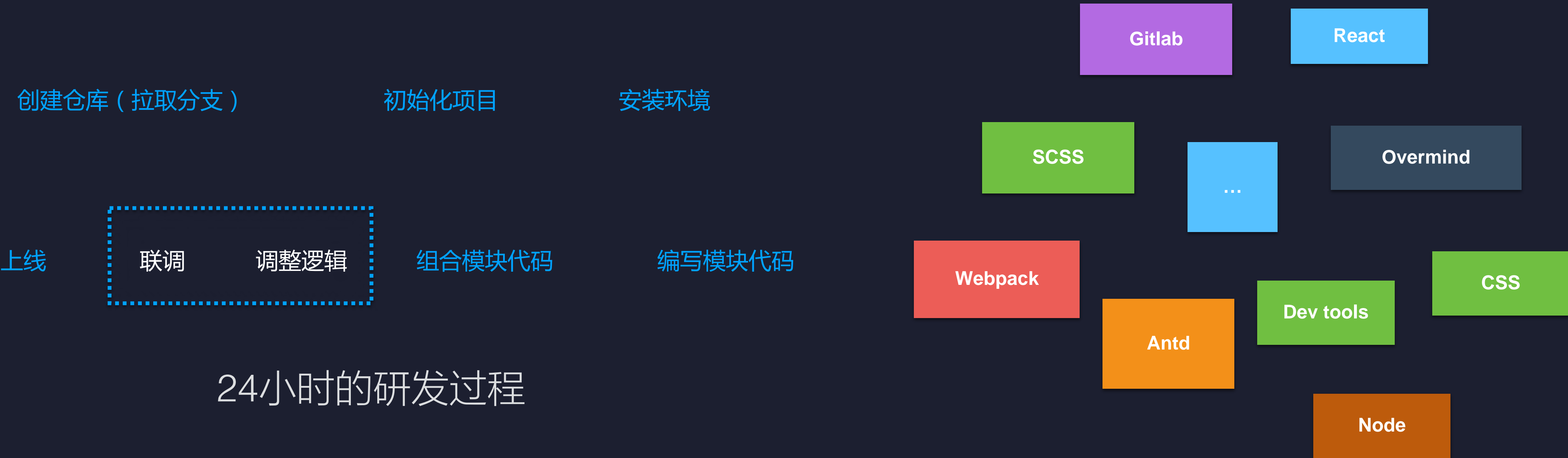
大量场景模式化，
搜索列表页，详情，数据看板等 Web侧65%左右，
中长尾页面较多

应用覆盖率

应用名称	域名	代码分支	对比分支	创建时间	当前覆盖率	期望覆盖率	操作
music-gateway-authorization	qa-actd.lgame.163.com	feature/monitor	master	1610364143000	100%	100%	编辑 查看
music-vip-activity	qa-amrmon.lgame.163.com	mapper			100%	100%	编辑
music-vip-center	qa-lgame.163.com	master	feature/benefit-pondant		100%	100%	编辑 查看
music-vip-gift	qa-best.lgame.163.com	master	master	1610356222000	100%	100%	编辑 查看
music-test-adv					100%	100%	编辑
music-vip-product	qa-ask.lgame.163.com	feature/bpc_together			100%	100%	编辑
music-vip-center	qa-actd.lgame.163.com	feature/benefit-pondant	master		100%	100%	编辑 查看

为什么需要低代码平台 - 核心问题域

- 如何加速软件的研发过程? —————> 模型驱动
- 如何降低软件的研发门槛? —————> UI可视化编程



低代码平台的核心 - UI可视化编程



环境: 域名: 应用: [查询](#)

应用名称	域名	代码分支	全量覆盖率	增量覆盖率	操作
music-mlog	http://qa-note.igame.163.com	feature/monitor	<div><div></div></div> 0%	<div><div></div></div> 0%	编辑 查看
music-coverage	http://qa-lishui.igame.163.com		<div><div></div></div> 0%	<div><div></div></div> 0%	编辑 查看
music-mlog	http://qa-note.igame.163.com		<div><div></div></div> 0%	<div><div></div></div> 0%	编辑 查看
music-mlog	http://qa-note.igame.163.com		<div><div></div></div> 0%	<div><div></div></div> 0%	编辑 查看

物料



可视化设计器



UI

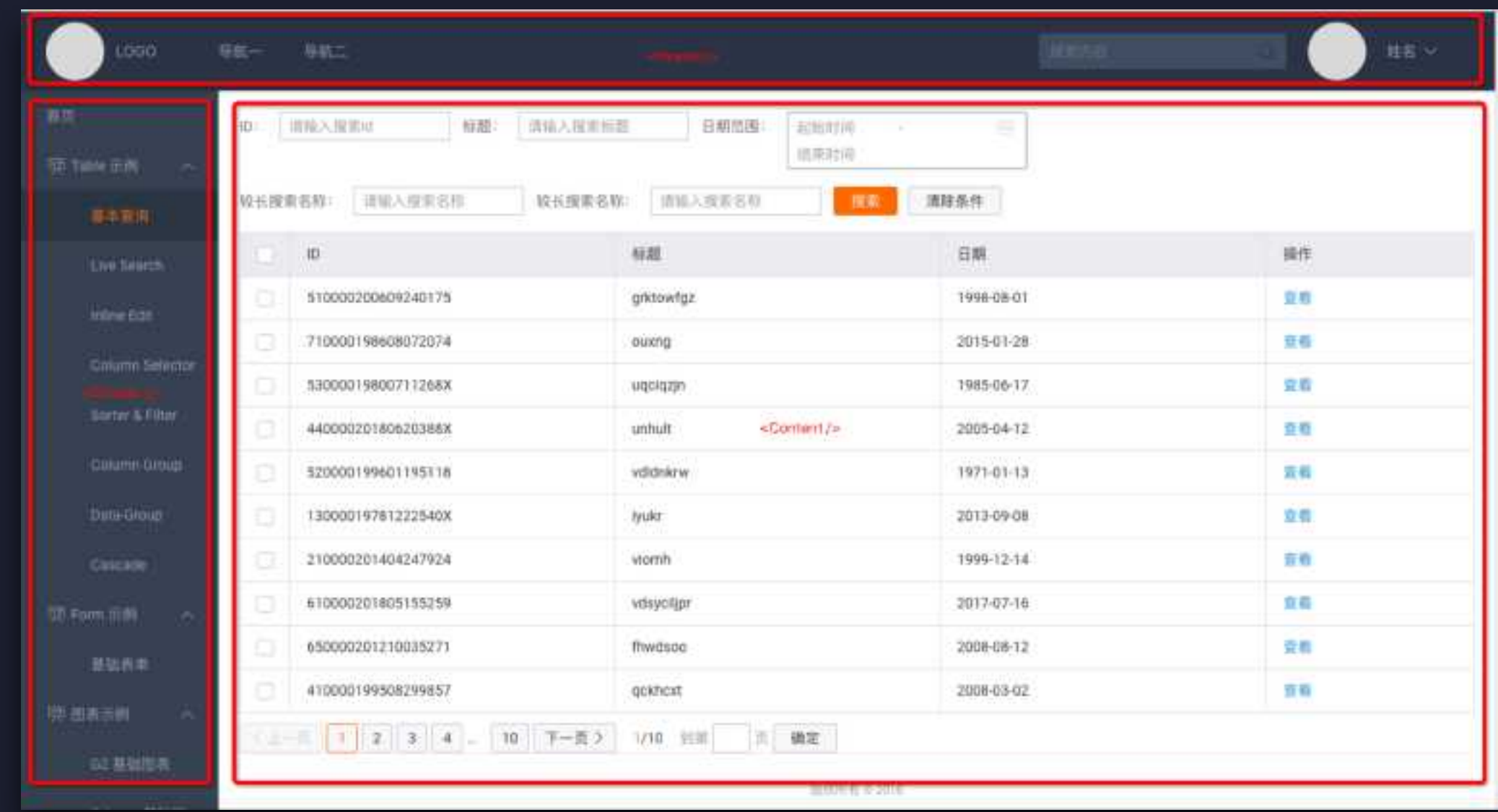
编辑时 点选，拖拽，配置能力

预览时 所见即所得的能力



UI可视化编程的模型就是将物料编排出产物，通过产物渲染UI的过程

低代码平台的核心 - 传统UI可视化的本质



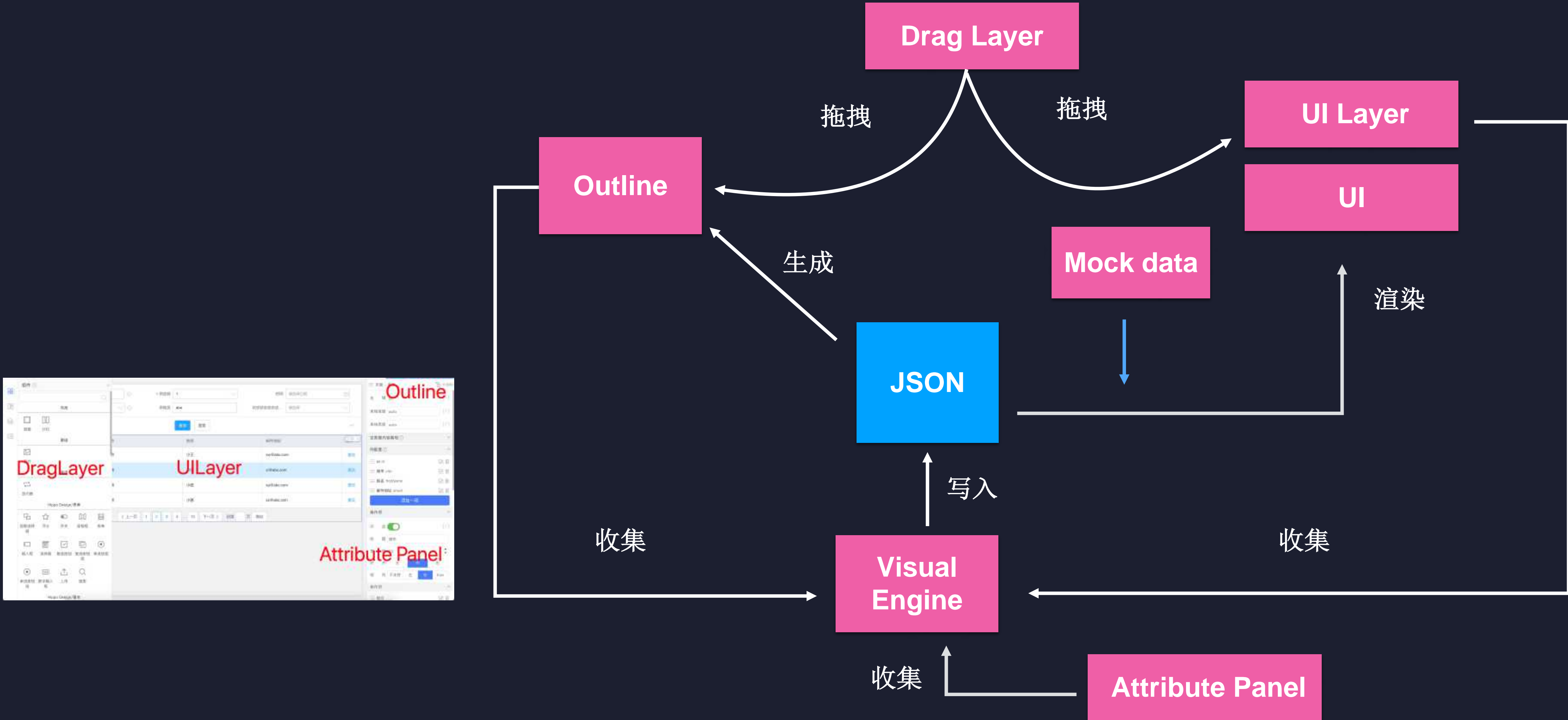
→ `<Header></Header>`
`<Sidebar></Sidebar>`
`<Content></Content>`



```
[{  
  type: 'Header',  
  attributes: {},  
  children: []  
}, {  
  type: 'Sidebar',  
  attributes: {},  
  children: []  
}, {  
  type: 'Content',  
  attributes: {},  
  children: []  
}]
```

产物就是JSON数据

低代码平台的核心 - 传统UI可视化的整体流程



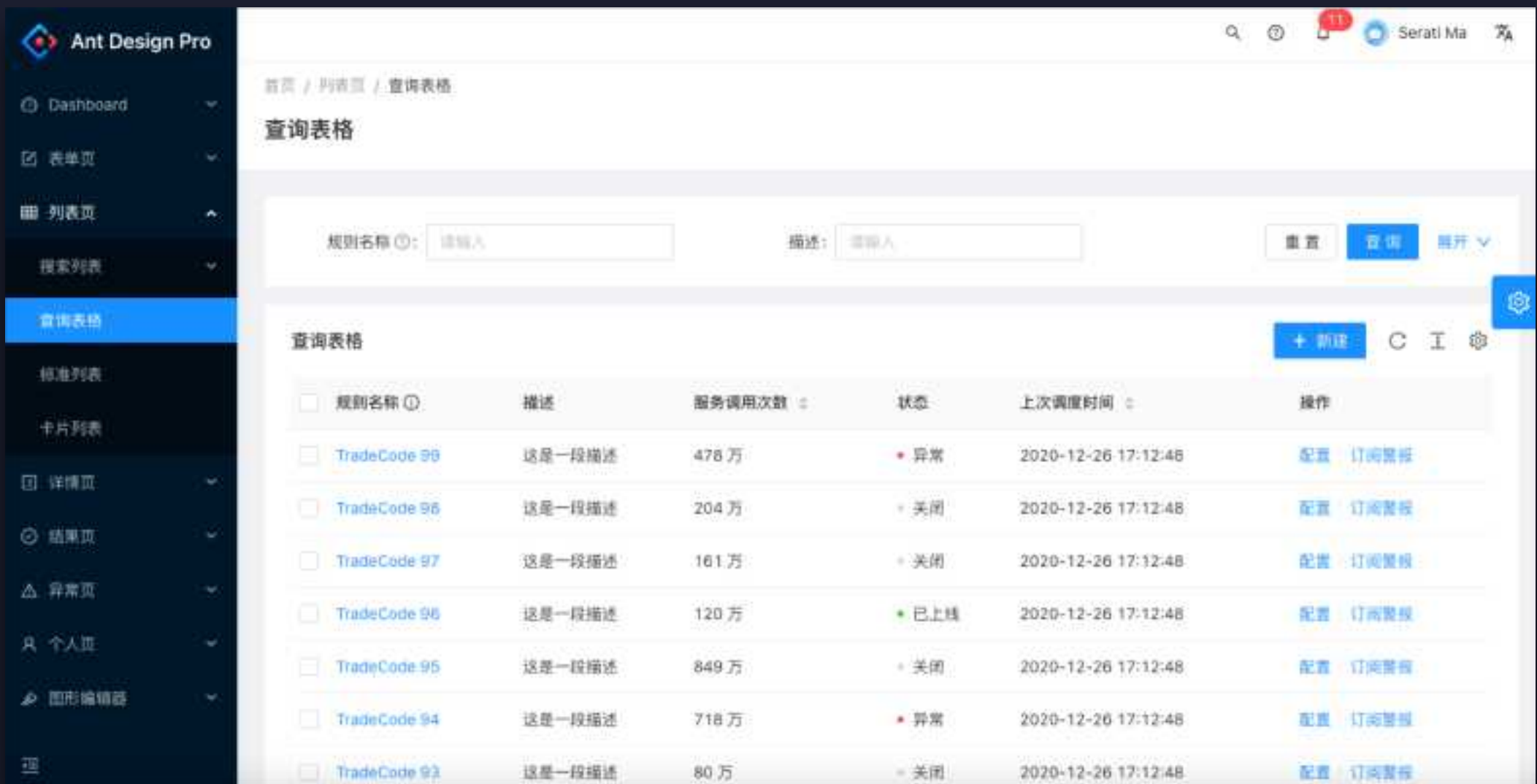
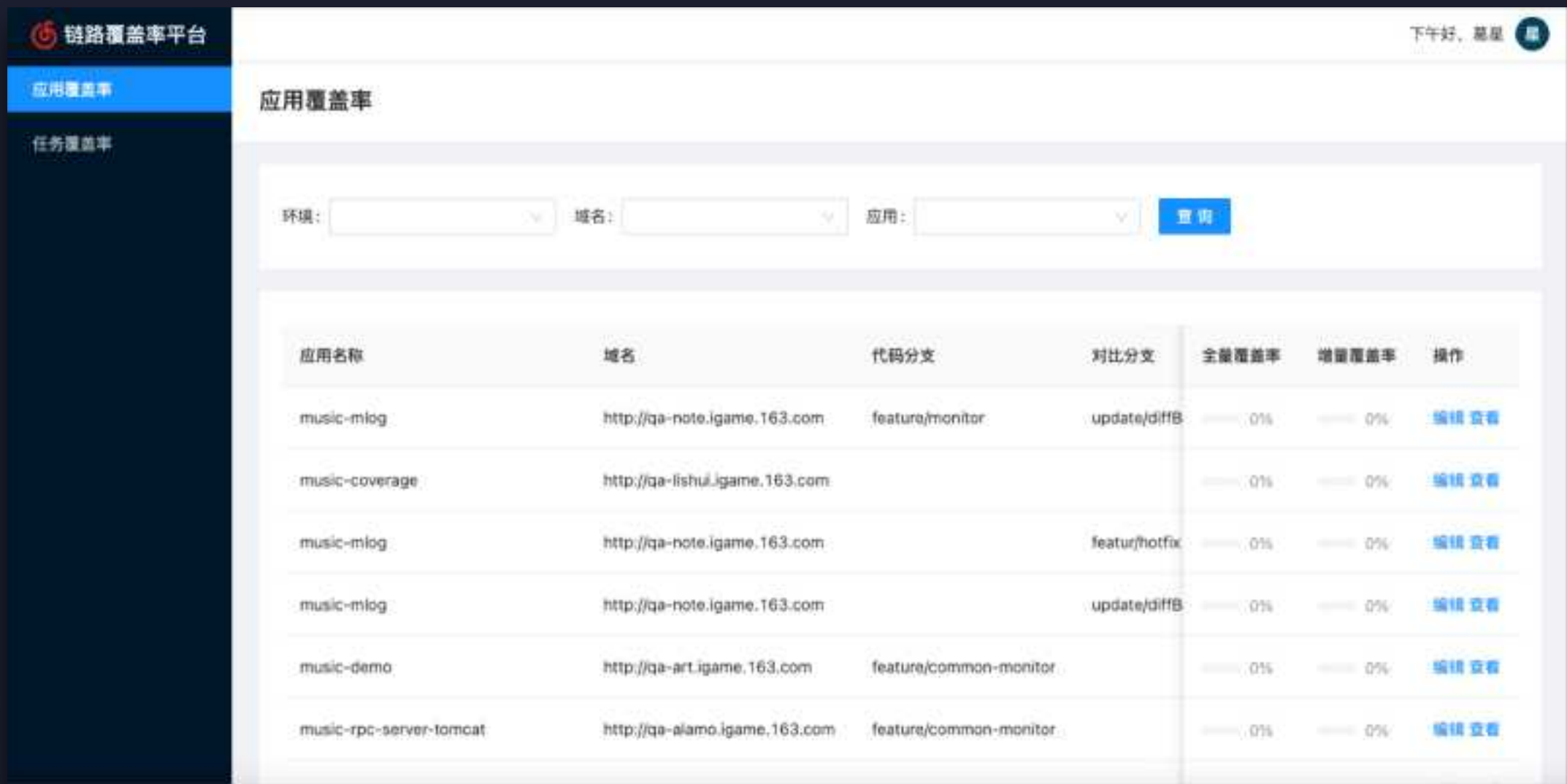
低代码平台的核心 - UI可视化编程

中后台系统的可视化搭建和**C**端搭建的不同点

必然涉及到不同程度的逻辑编写

低代码平台的核心 - UI可视化编程

想象下页面越来越复杂的情况



一条路走到黑，没有回头路
稍微复杂的场景下效率不如写代码

低代码平台的核心 - UI可视化编程

传统UI可视化编程之殇

可维护性

- 不断膨胀的JSON协议规范
- 看不到边界的JSON描述

```
"attributes": {  
  "prop1": 1234, // 简单 json 数据  
  "prop2": [{ // 简单 json 数据  
    "label": "选项1",  
    "value": 1  
  }, {  
    "label": "选项2",  
    "value": 2  
  }],  
  "prop3": [{  
    "name": "myName",  
    "rule": {  
      "type": "JSExpression",  
      "value": "/\\w+/i"  
    }  
  }],  
  "valueBind": { // 变量绑定  
    "type": "JSExpression",  
    "value": "this.state.user.name"  
  },  
  "onClick": { // 动作绑定  
    "type": "JSExpression",  
    "value": "function(e) { console.log(e.target.innerText) }",  
  },  
  "onClick2": { // 动作绑定2  
    "type": "JSExpression",  
    "value": "this.submit"
```

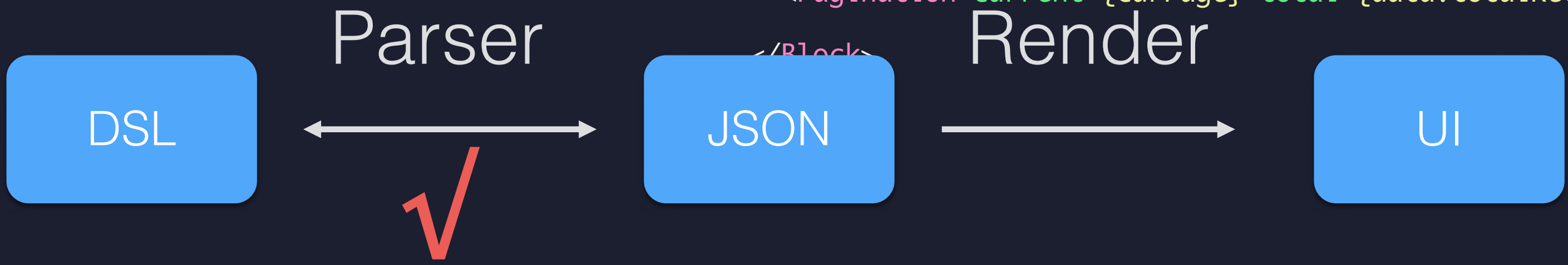
低代码平台的核心 - UI可视化编程

可维护性的解法

DSL - Domain Specific Language

- 可读性
- 代码属性
- 受限 - 双向工程（Round-trip engineering）

```
div
<Panel title="采购合同列表">
  <InlineView marginY="12">
    <Action confirm="确定要删除选中的数据吗? " data={{ids: selectedRowIds}}>
      <Button>
        批量删除
      </Button>
    </Action>
  </InlineView>
  <Table dataSource={data.data} primaryKey="purOrderId" hasBorder={false} isZebra={true}>
    <Table.Column title="订单id" dataIndex="purOrderId"/>
    <Table.Column title="备注" dataIndex="purRemark"/>
    <Table.Column title="状态" dataIndex="statusDesc"/>
    <Table.Column title="操作" dataName="record" cell={(value, index, record) => {
      return lastColumn()({ record: record });
    }} width="150"/>
  </Table>
  <Block marginTop="12">
    <Pagination current={curPage} total={data.totalRecord} pageSize="5" />
  </Block>
</div>
```



低代码平台的核心 - UI可视化编程

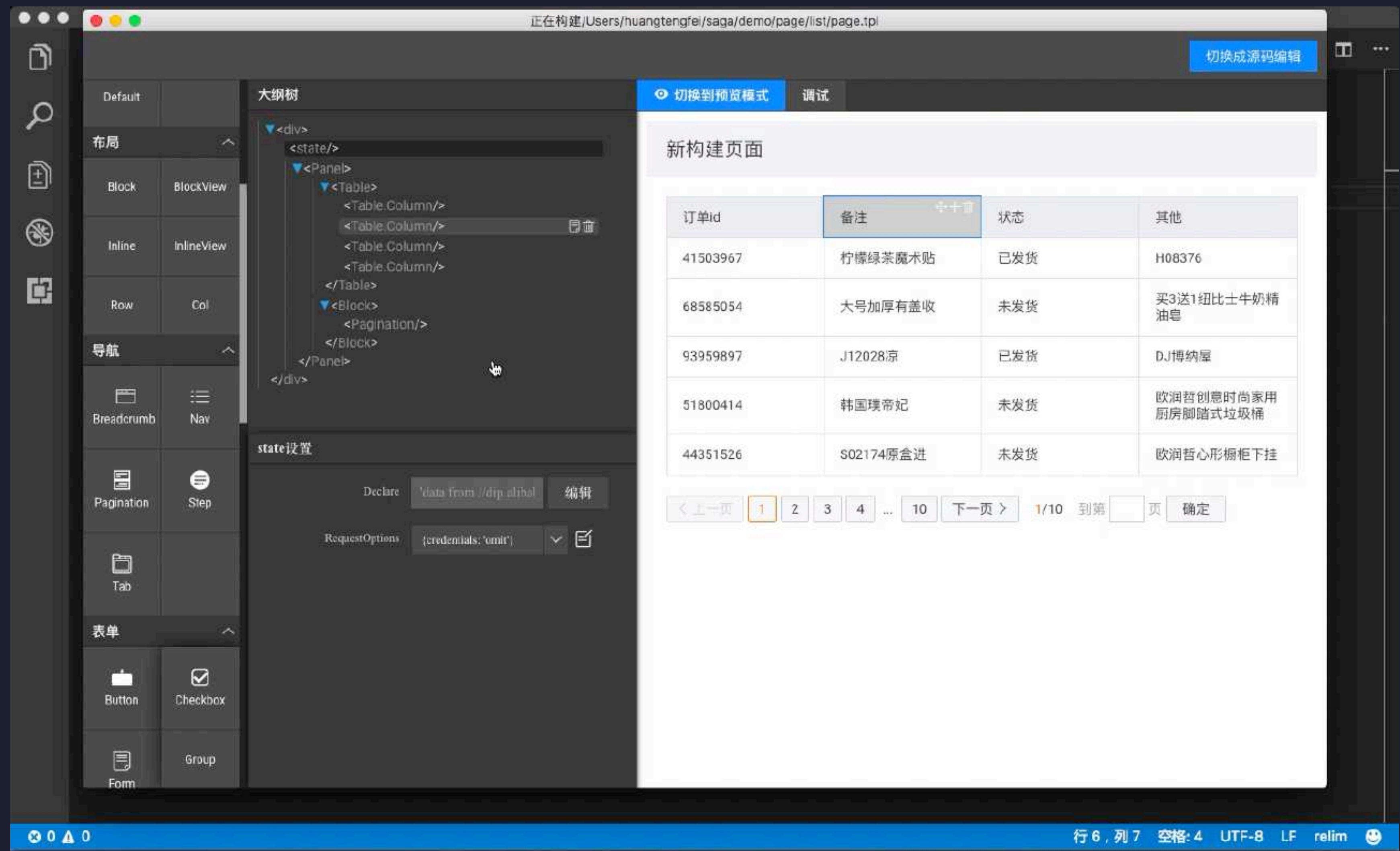
可维护性的解法

DSL - Domain Specific Language

几乎所有的**DSL**都可以通过可视化方式来构建

如果要实现可视化的双向互转，必须基于特定的**DSL**来实现

低代码平台的核心 - UI可视化编程



理想是美好的，现实是骨感的

低代码平台的核心 - UI可视化编程

DSL是否是完美的？

- 学习成本
- 用户心智 - 人群的差异性

你们快来用啊！ 你们自己在用吗？

我只想用React！

推广起来成本很高

低代码平台的核心 - UI可视化编程

我们需要重新思考UI可视化编程的中间产物



VS



<div />

HTMLParser

```
{
  type: "div",
  attributes: {}
  children: []
}
```

Babel

<Table />

```
"end": 25,
"loc": {},
...
},
"closingElement": {
  "type": "JSXClosingElement",
  "start": 88,
  "end": 96,
  "loc": {
    "start": {
      "line": 3,
      "column": 10
    },
    "end": {
      "line": 3,
      "column": 18
    }
  },
},
"name": {
  "type": "JSXIdentifier",
  "start": 90,
  "end": 95,
  "loc": {
    "start": {
      "line": 3,
      "column": 12
    },
    "end": {
```

低代码平台的核心 - UI可视化编程

所以我们只要将原先的可视化对**JSON**的映射

转换成对**AST**的映射就能实现**React**的中后台可视化研发了？

低代码平台的核心 - UI可视化编程

基于ESTree Spec AST的UI可视化编程

难点一

如何关联**DOM** 和 **AST**?

```
traverse(ast, {
  JSXElement(path /* , data */) {
    const { node } = path;
    const { openingElement } = node;

    // 插入 data-hippo data-id
    let { uuid } = node;

    const { attributes } = openingElement;
    const componentName = getNodeName(openingElement);

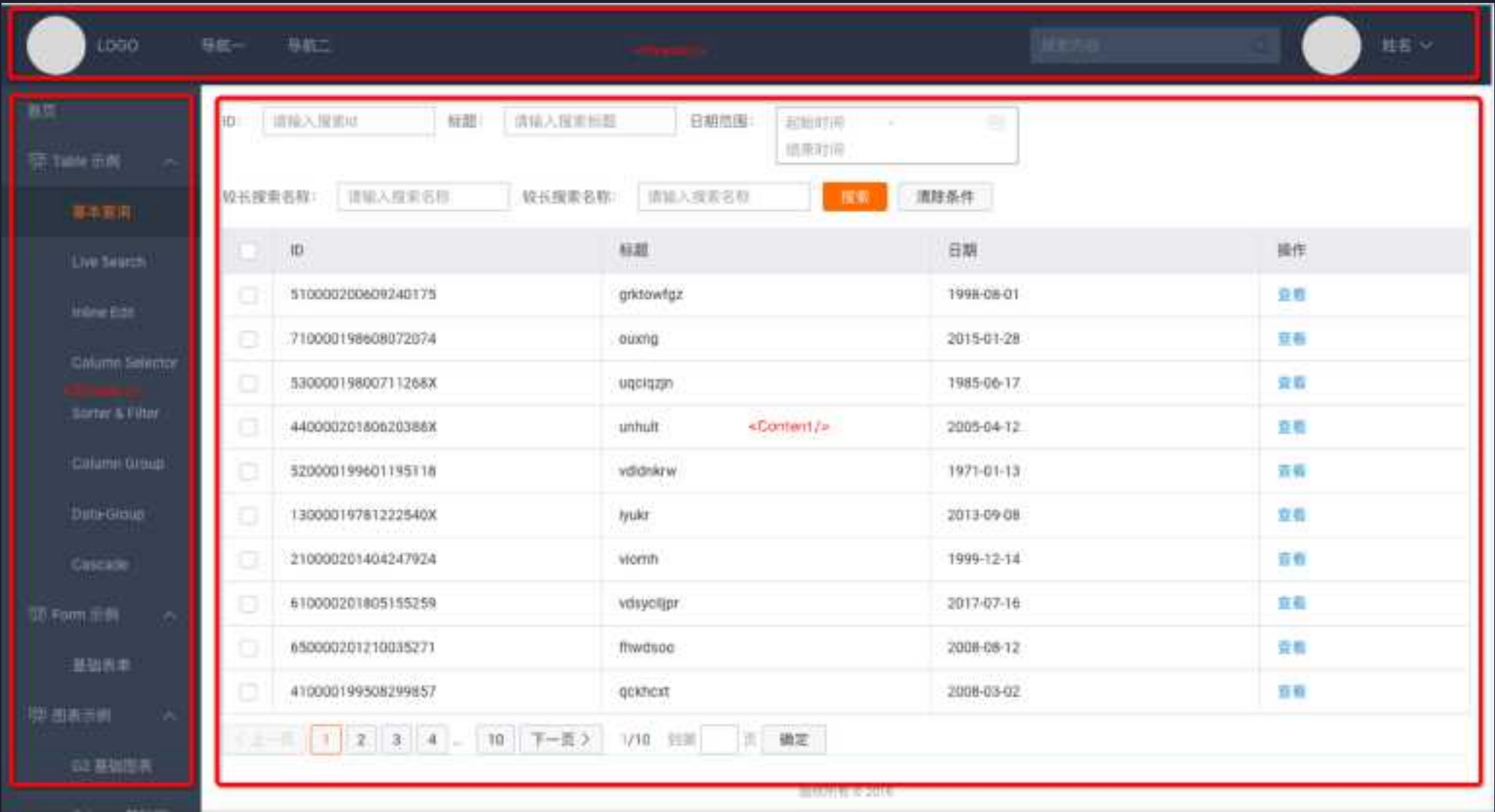
    if (!uuid) {
      uuid = `U_${componentName}_${Math.random()
        .toFixed(5)
        .replace('0.', '')}`;
      node.uuid = uuid;
      fillTime(attributes, SLOT.id, uuid);
      fillTime(attributes, SLOT.hippo, "");
    }
  }
});
```


低代码平台的核心 - UI可视化编程

基于ESTree Spec AST的UI可视化编程

难点二

源码与可视化不对等的问题？



<Header/>
<Sidebar/>
<Content/>

```
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

低代码平台的核心 - UI可视化编程

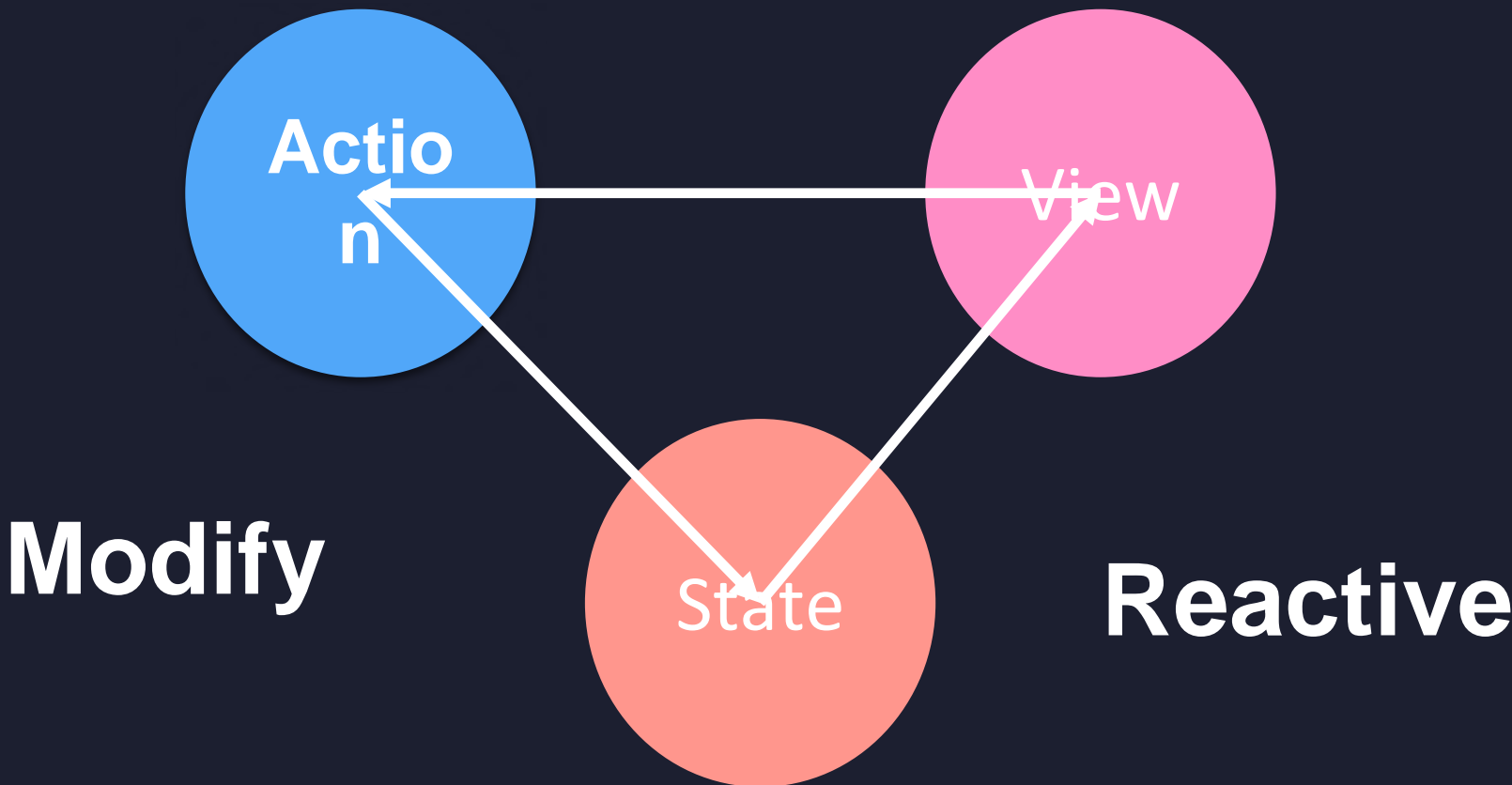
基于ESTree Spec AST的UI可视化编程

难点二

源码与可视化不对等的问题？

UI与逻辑分离

dispatch



```
export default ({
  state: {
    count: 0,
    list: [],
  },
  actions: {
    increment(state) {
      const { count } = state;
      count++;
    },
    async fetch(state) {
      const ret = await request('/list');
      this.transaction(() => {
        state.data = ret.data;
        state.total = ret.total;
      });
    },
  },
});
```

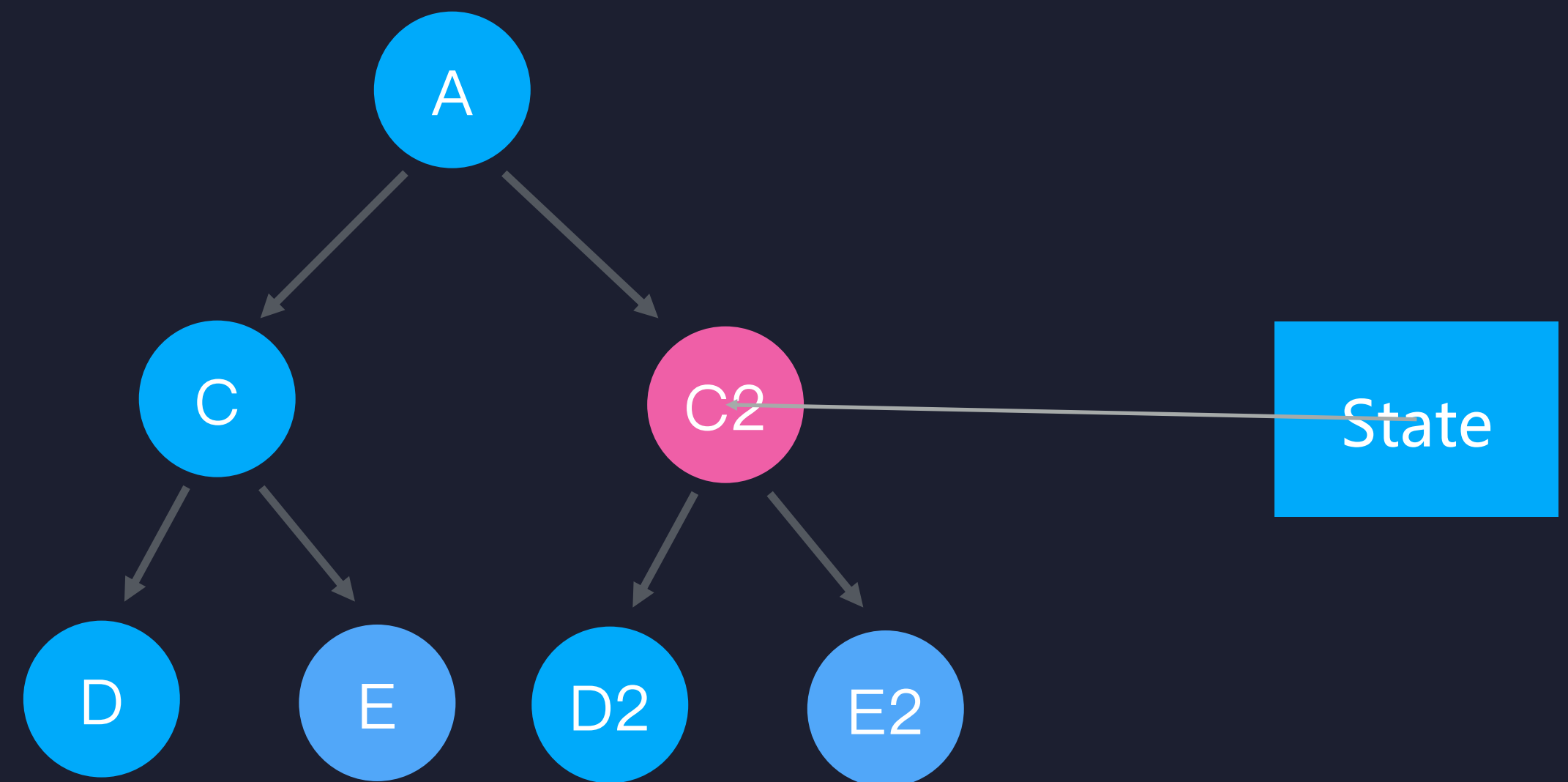
```
@connect(state => state);
class App extends React.Component {
  fetch = () => {
    this.props.dispatch('fetch');
  }
  render() {
    return <span onClick={this.fetch}>
      {this.props.state.count}
    </span>
  }
}
```

SA架构额外带来的好处— 高性能

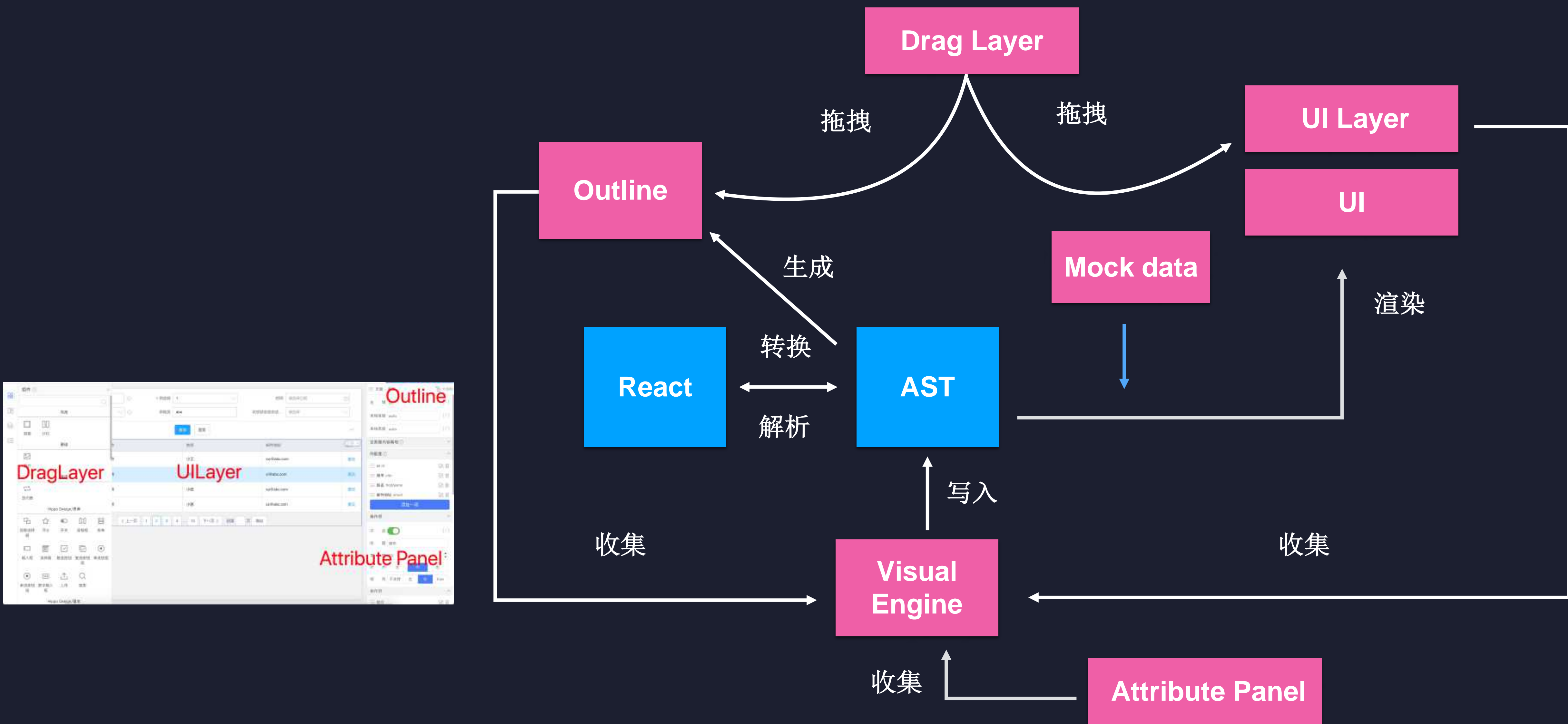
1. 依赖追踪保证只更新对有依赖数据变更的更新
2. 一个事务周期内保证多次状态变更只会有一次更新
3. 子组件无状态更新的时候不渲染

```
export default {  
  state: {  
    count: 0,  
    list: [],  
  },  
  actions: {  
    increment(state) {  
      const { count } = state;  
      count++;  
    },  
    async fetch(state) {  
      const ret = await request('/list');  
      this.transaction(() => {  
        state.data = ret.data;  
        state.total = ret.total;  
      });  
    }  
  }  
}
```

```
@connect(state => state);  
class App extends React.Component {  
  render() {  
    return <span>{this.props.count}</span>  
  }  
}
```



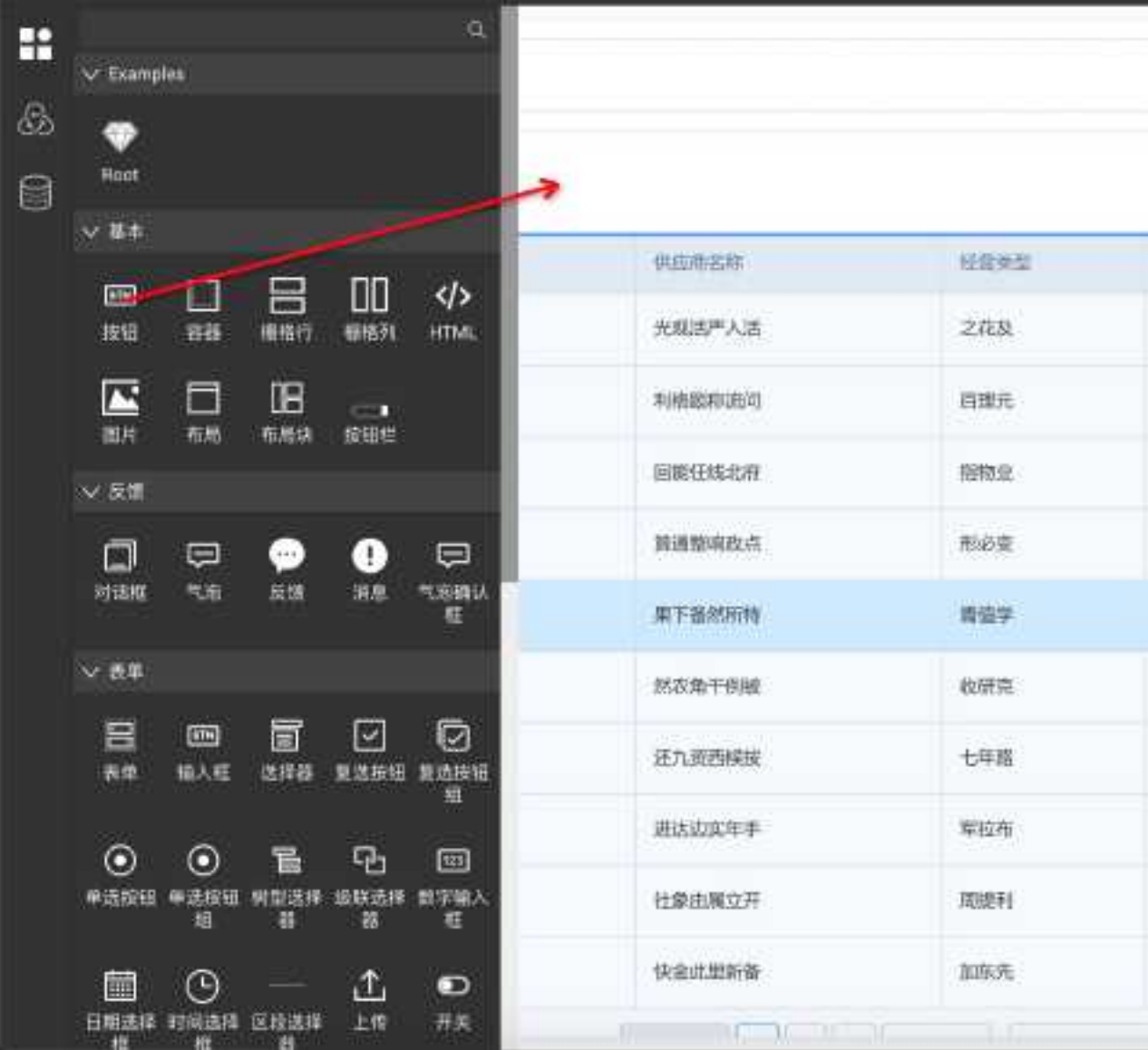
低代码平台的核心 - AST UI可视化的整体流程



低代码平台的核心 - UI可视化编程

基于AST的UI可视化编程本质

把人的意图变成机器可以识别的代码



1. 判断Button这个变量是否在View里面导入过
2. 导入过，则返回
3. 没有导入，则导入Button
4. 导入Button，判断包含Button的大包是否存在
5. 有，使用之前的大包
6. 没有，重新写入
7. 最后写入Button的React Element

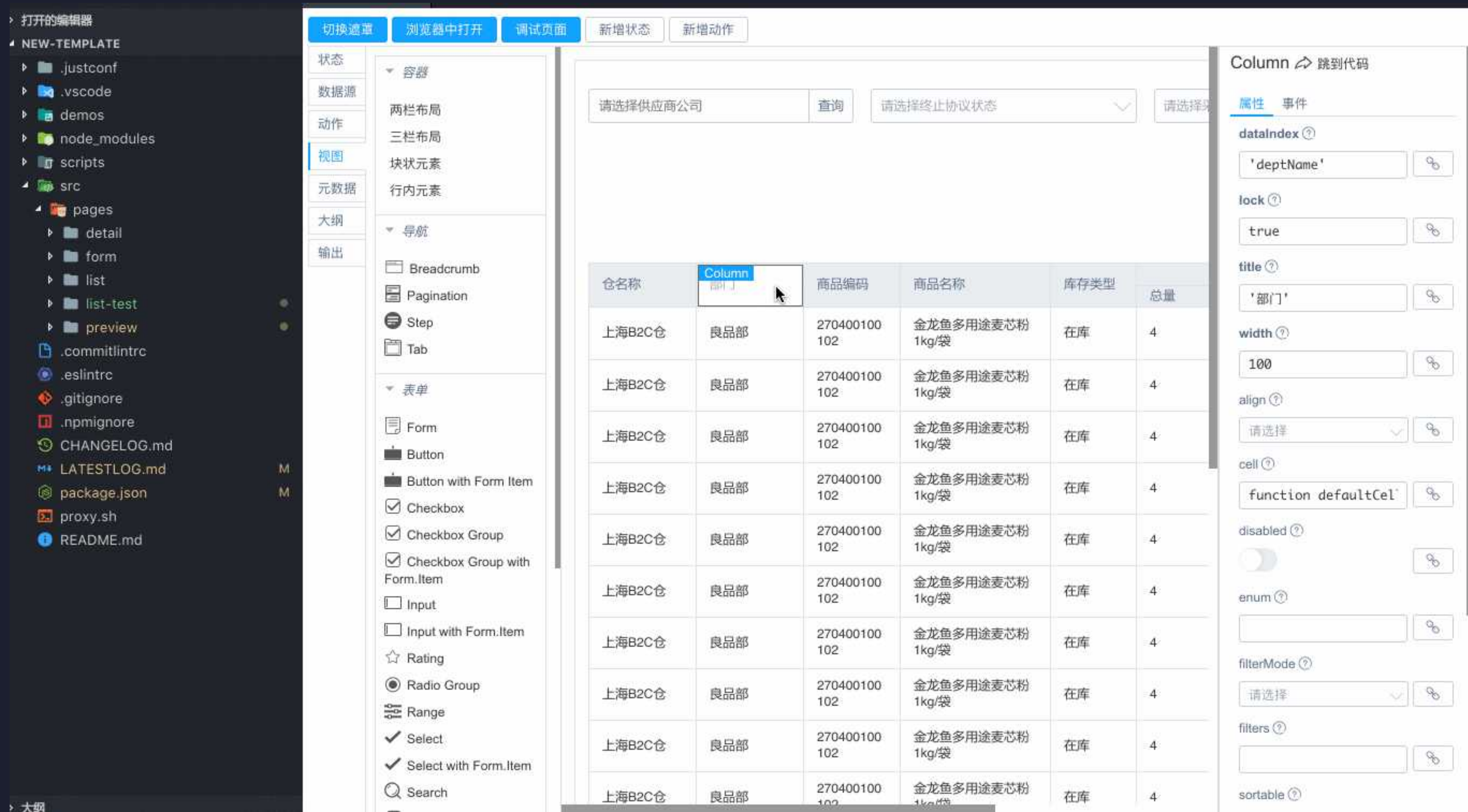
```
import { Button } from 'antd';
```

```
<Button></Button>
```

```
import { Button, Table } from 'antd';
```

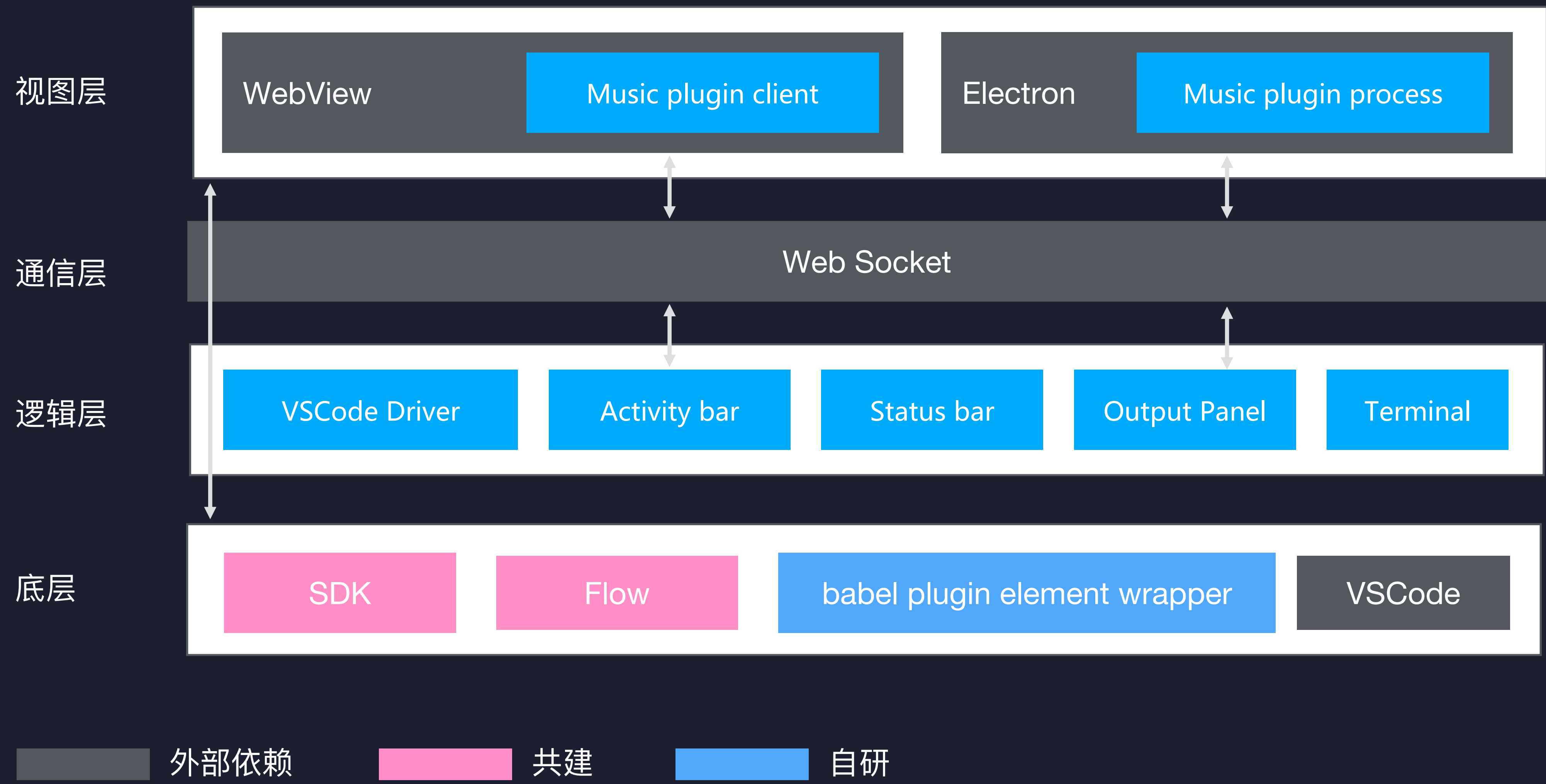

低代码平台的核心 - UI可视化编程

基于AST的UI可视化编程 - 实现 LowCode 和 ProCode 的双向互通



低代码平台的核心 - UI可视化编程

VSCode线下工具架构图



低代码平台的核心 - UI可视化编程

性能是线下方案的阿克琉斯之踵

- webpack 构建性能
- websocket 通信耗时
- 磁盘IO耗时
- webpack HMR 通信耗时

无论如何优化总是很难达到**100ms**以内

低代码平台的核心 - UI可视化编程

性能是线下方案的阿克琉斯之踵

把流程搬到线上去

webpack能在浏览器端运行吗？

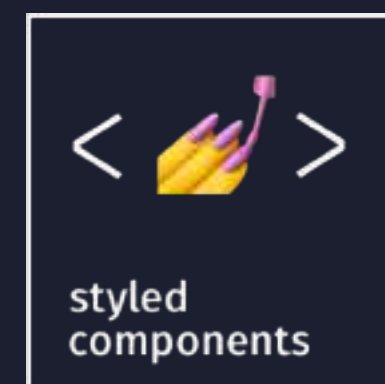
答案是肯定的

低代码平台的核心 - UI可视化编程

UI可视化编译过程线上化

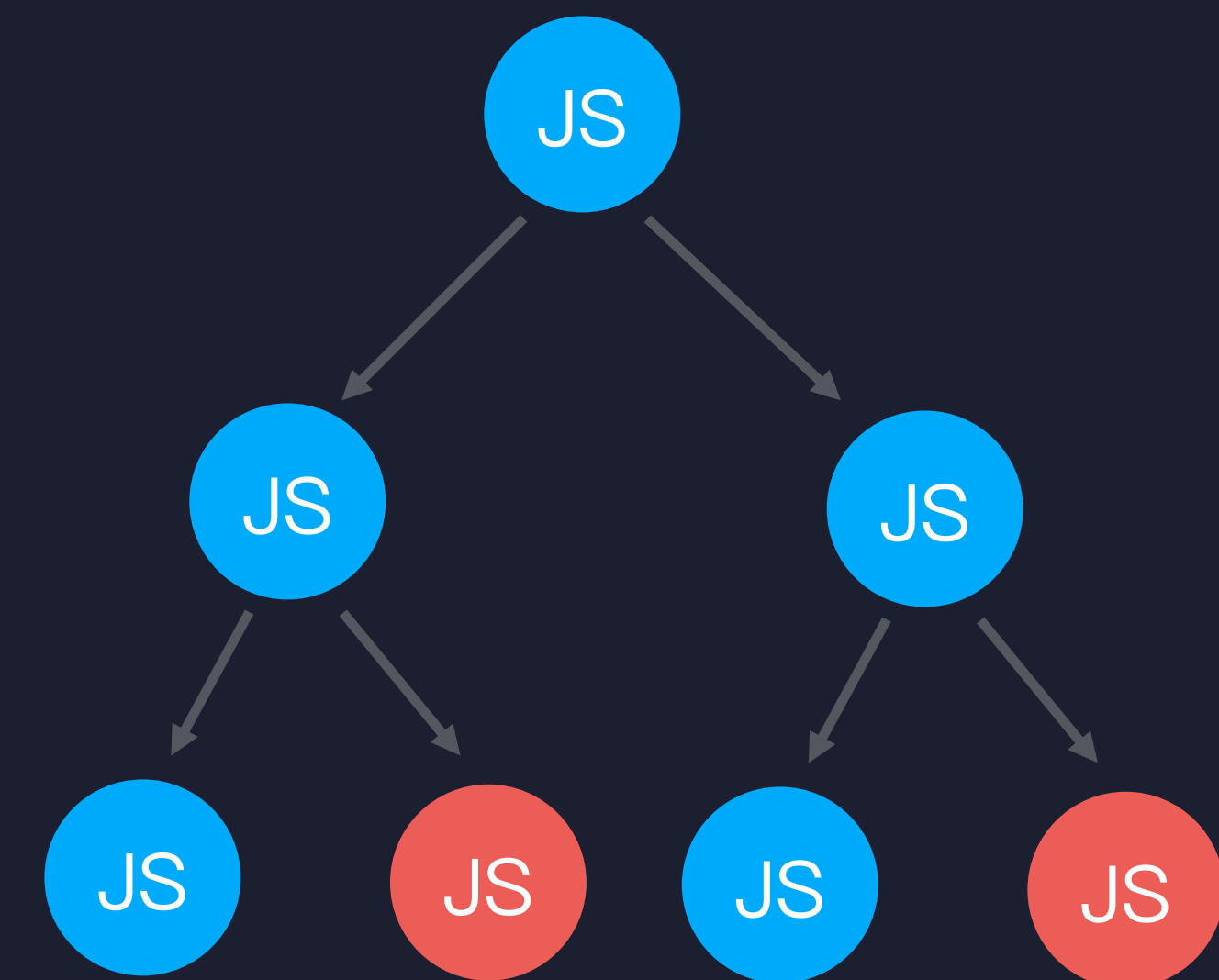
webpack 的本质

modular compile + modular bundle + modular JIT



Sea.js

sea.js 模块构建器

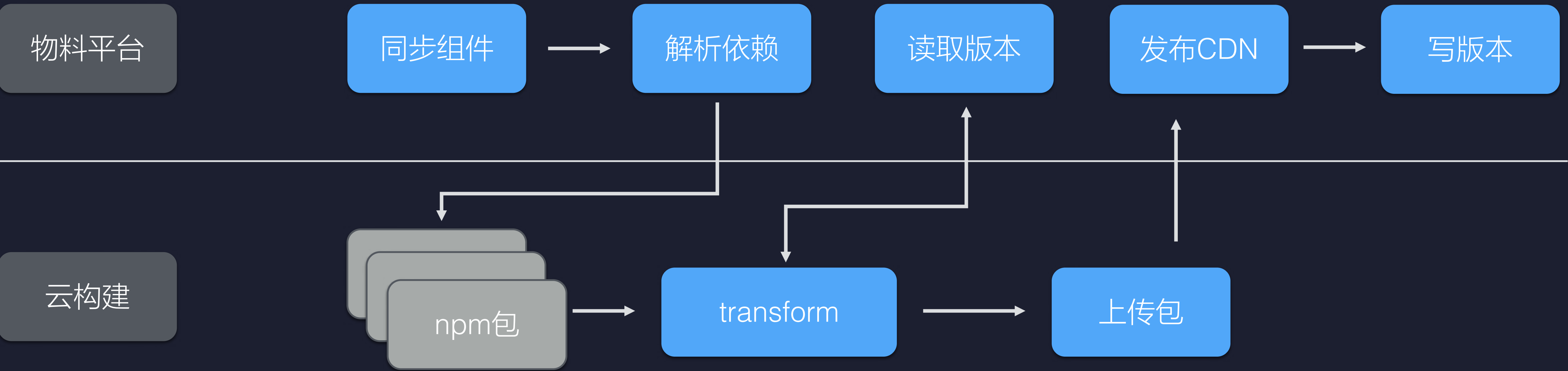


低代码平台的核心 - UI可视化编程

UI可视化编译过程线上化的额外挑战

挑战一: npm生态的问题解决

```
define("@alife/legion/1.12.2/lib/basement/Logger", ["../core/base"], function(require, exports, module) {  
  // your code here  
})
```



低代码平台的核心 - UI可视化编程

UI可视化编译过程线上化的额外挑战

挑战二：Seajs下的HMR

```
seajs.refresh = (url, entry, callback) => {  
  delete seajs.cache[entry];  
  delete seajs.data.fetchedList[entry];  
  
  const cache = seajs.cache;  
  Object.keys(cache).forEach(key => {  
    const module = cache[key];  
    const deps = module.dependencies.map(dep => {  
      return seajs.resolve(dep, key);  
    });  
    if (deps.indexOf(item) !== -1) {  
      delete seajs.cache[key];  
      delete seajs.data.fetchedList[key];  
    }  
  });  
  
  delete seajs.cache[item];  
  delete seajs.data.fetchedList[item];  
});  
  
seajs.use([entry], () => {  
  if (callback) callback();  
});
```

低代码平台的核心 - UI可视化编程

UI可视化编译过程线上化的额外挑战

挑战三：React diff的问题

相同的组件类型才会diff，由于代码每次都进行了HMR
所以每次都是新的组件，页面会重新Render

```
ProxyRegister('/src/example/view.jsx#FormItem', FormItem);
ProxyRegister('/src/example/view.jsx#FormSubmit', FormSubmit);
ProxyRegister('/src/example/view.jsx#FormReset', FormReset);
ProxyRegister('/src/example/view.jsx#FormFieldSet', FormFieldSet);
ProxyRegister('/src/example/view.jsx#Dialog', Dialog);
ProxyRegister('/src/example/view.jsx#SearchForm', SearchForm);
ProxyRegister('/src/example/view.jsx#BasicNormalList', BasicNormalList);
```

```
import React from 'react';
```

```
import { createProxy } from 'react-proxy';
```

```
const PROXIES = {};
```

```
const UNIQUE_ID = '__uniqueId';
```

```
export default function proxyRegister(id, type) {
```

```
  if (typeof type === 'function' && !type[UNIQUE_ID] && type.prototype) {
```

```
    Object.defineProperty(type, UNIQUE_ID, {
```

```
      value: id
```

```
    });
```

```
    let proxy = PROXIES[id];
```

```
    if (proxy) {
```

```
      proxy.update(type);
```

```
    } else {
```

```
      PROXIES[id] = createProxy(type);
```

```
    }
```

```
  }
```

```
}
```

```
const oldCreateElement = React.createElement;
```

```
React.createElement = function createElement(type, ...args) {
```

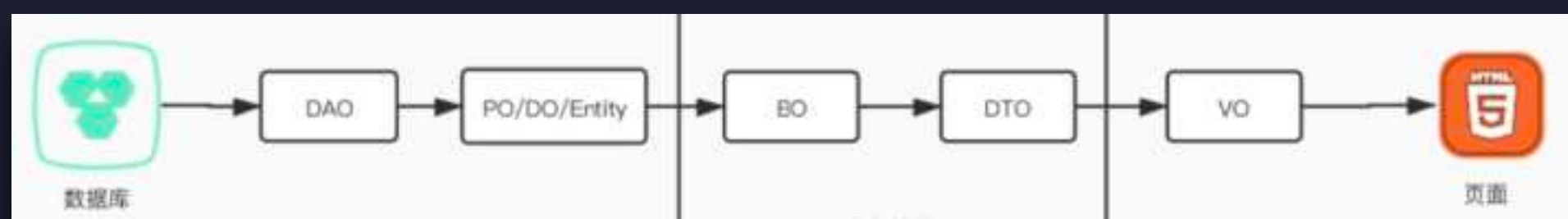
```
  if (type && type[UNIQUE_ID]) {
```

```
    type = PROXIES[type[UNIQUE_ID]].get();
```

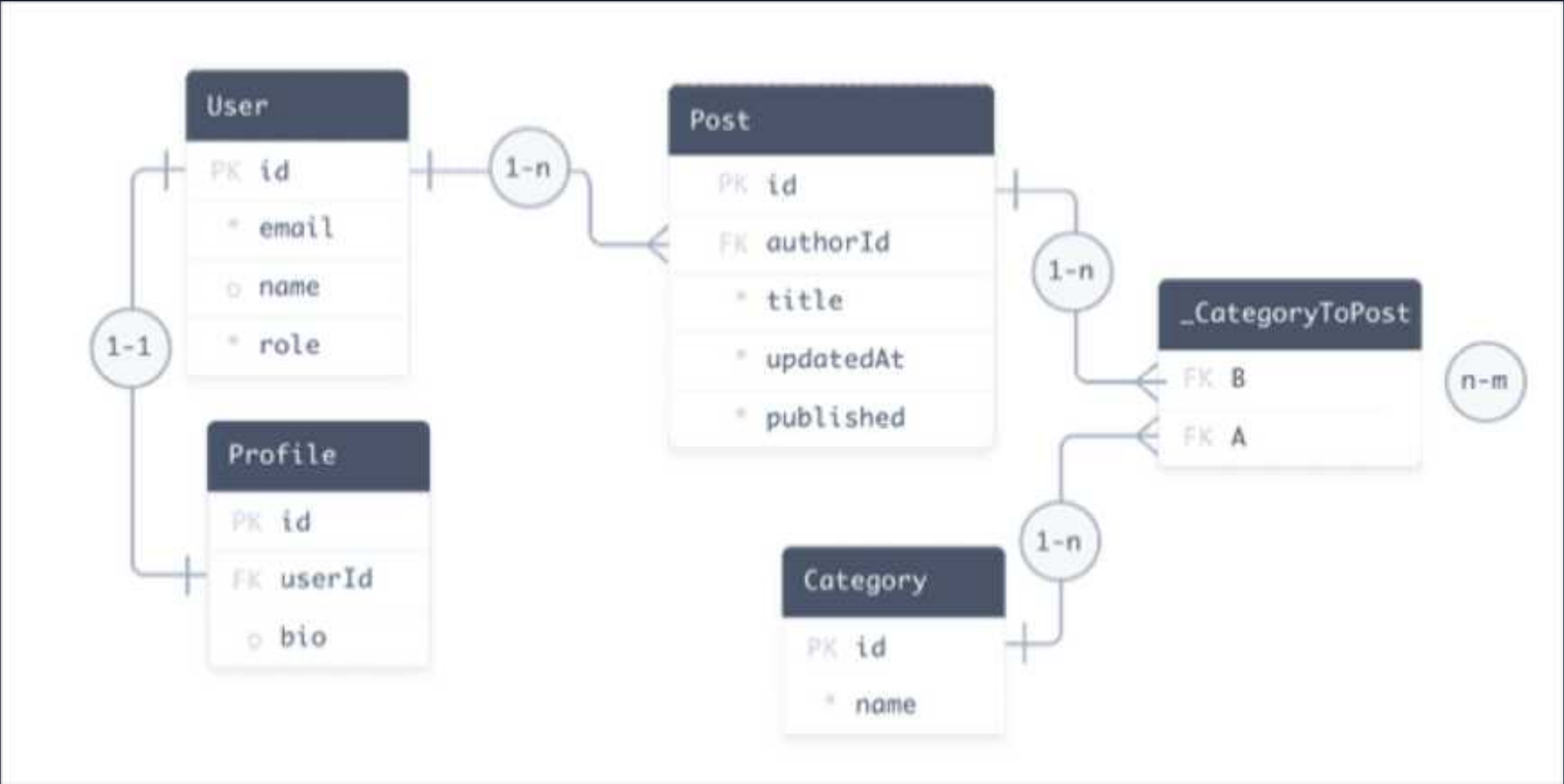
低代码平台的核心 - 模型驱动架构

UI可视化只专注在前端侧解决问题，如何加速整个中后台应用的研发速度呢？

它是一种基于UML以及其他工业标准的框架，支持软件设计和模型的可视化、存储和交换

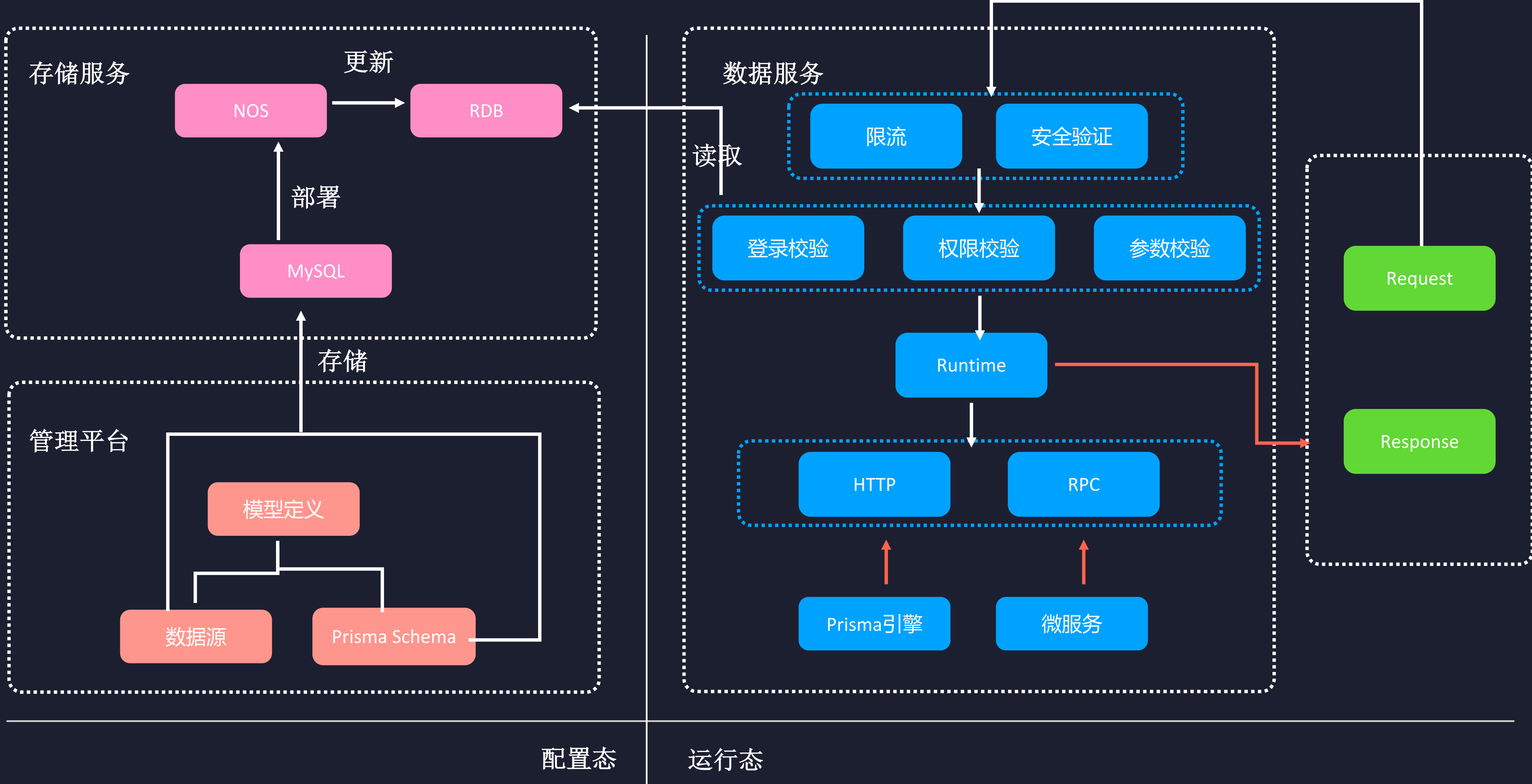


实际上模型并不等于实体，但是在这里简化了其概念



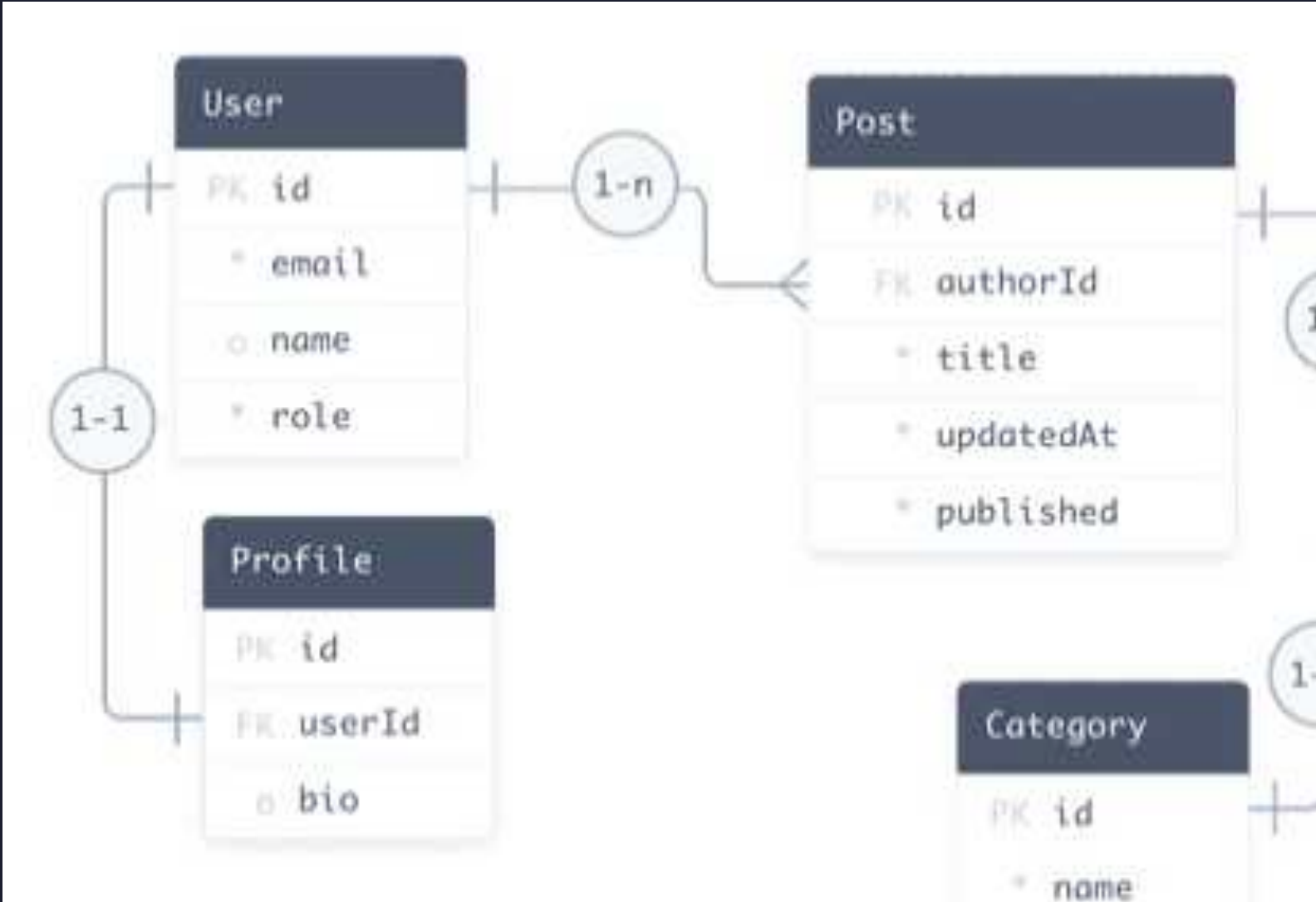
实体关系图

低代码平台的核心 - 模型驱动服务端整体流程



低代码平台的核心 - 模型驱动 - 数据建模

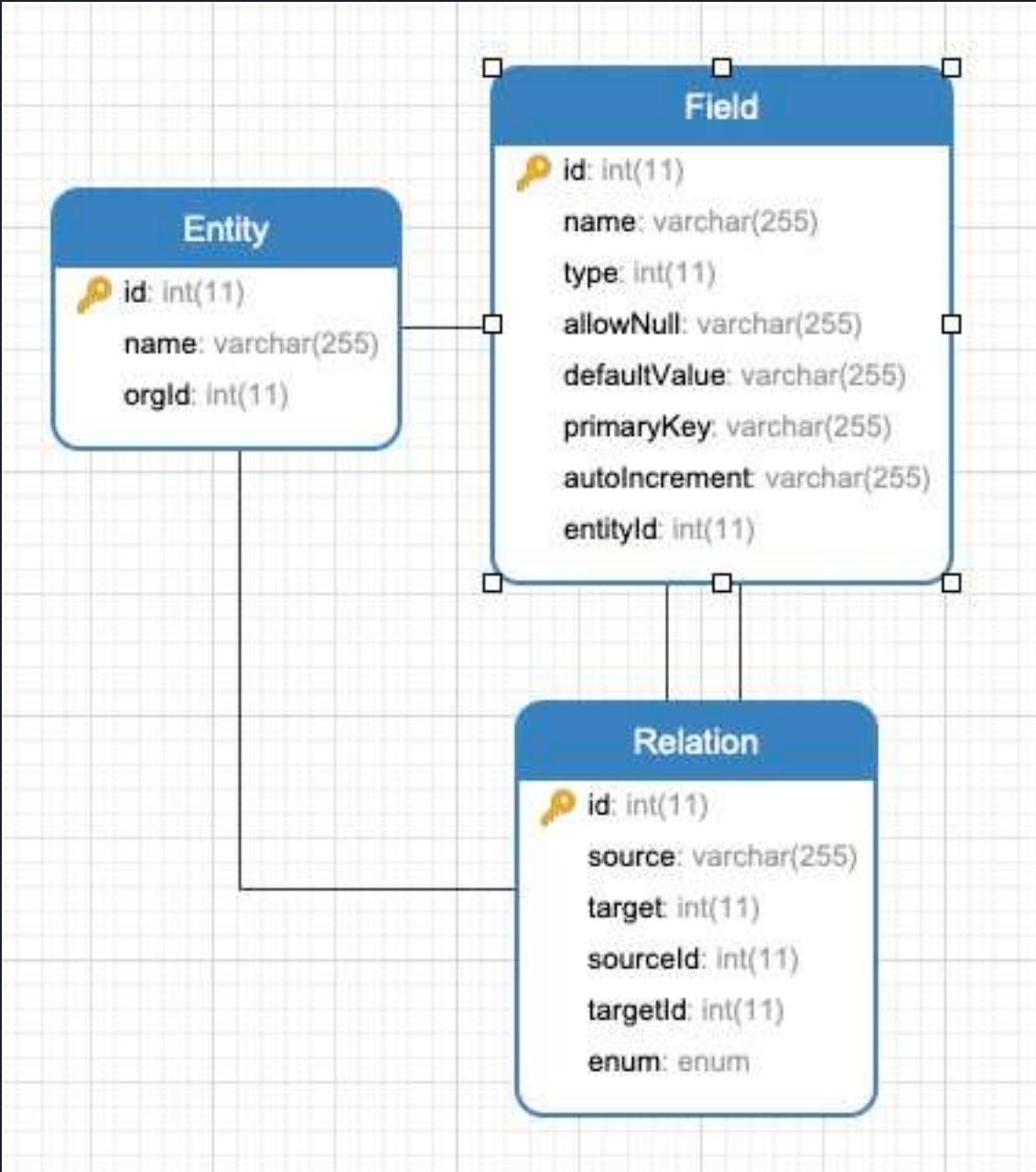
普通多租户 物理模型



普通多租户的问题

- 1. 用户具有执行DDL权限，模型爆炸导致的冲突，模型升级障碍
- 2. 由于都是物理模型，没法做有效的隔离，平台升级是个问题

元数据多租户 为租户分配一个标志位，比如orgId



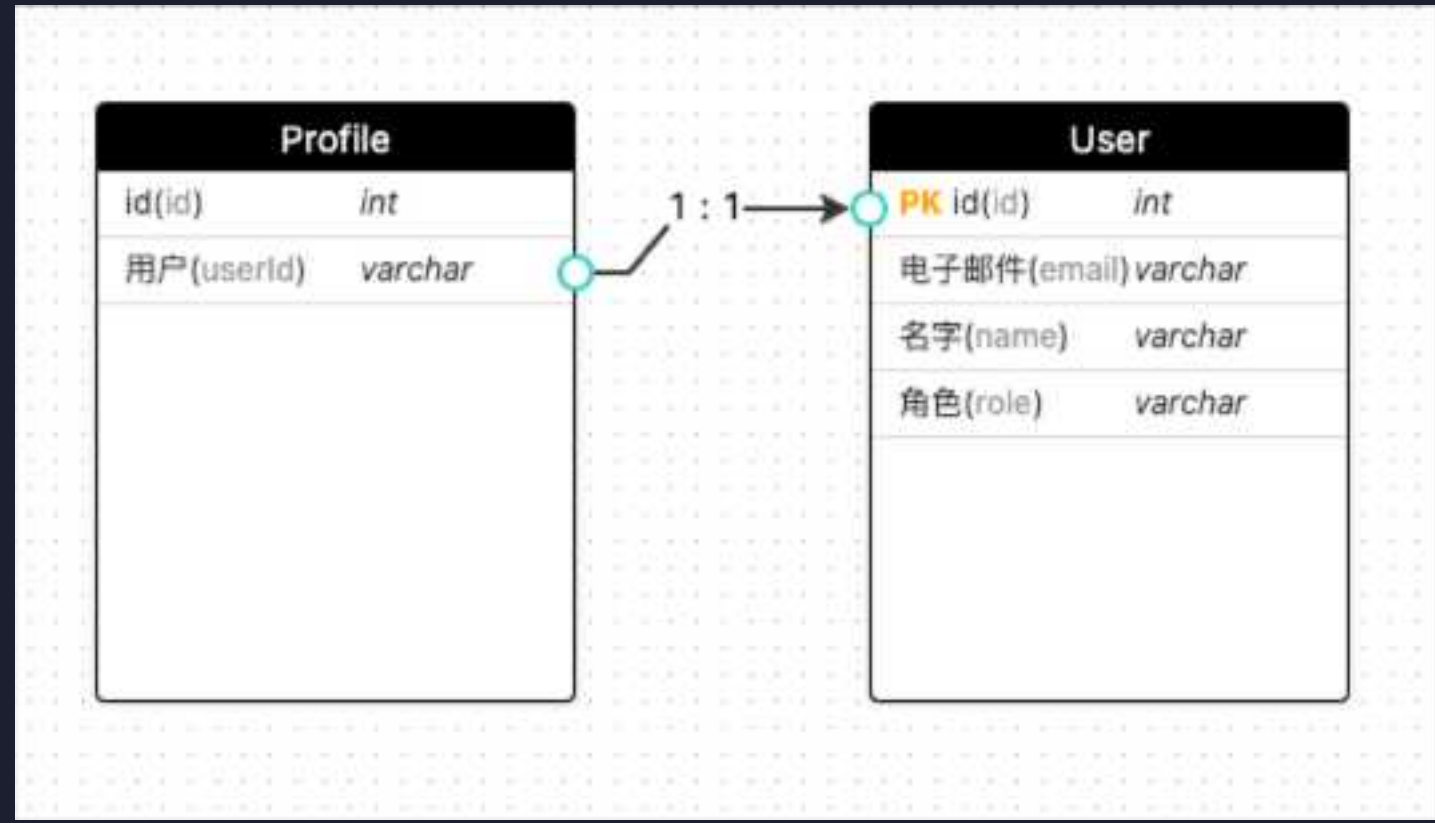
元数据驱动的优势

用户客制化变得容易，用户定义的模型和平台模型通过orgId有效的隔离

低代码平台的核心 - 模型驱动 - 代码生成

RTE（双向工程）
语言无关性

模型关系



Server Code

Single Source of truth

```
model User {
  id      Int      @id @default(autoincrement())
  email   String   @unique
  name    String?
  role    Role     @default(USER)
  posts   Post[]
  profile Profile?
}

model Profile {
  id      Int      @id @default(autoincrement())
  bio     String
  user    User     @relation(fields: [userId], references: [id])
  userId  Int
}
```

```
return await prisma.user.findUnique({
  where: {
    id
  }
})

find(options) {
  return await prisma.user.findMany(options)
}

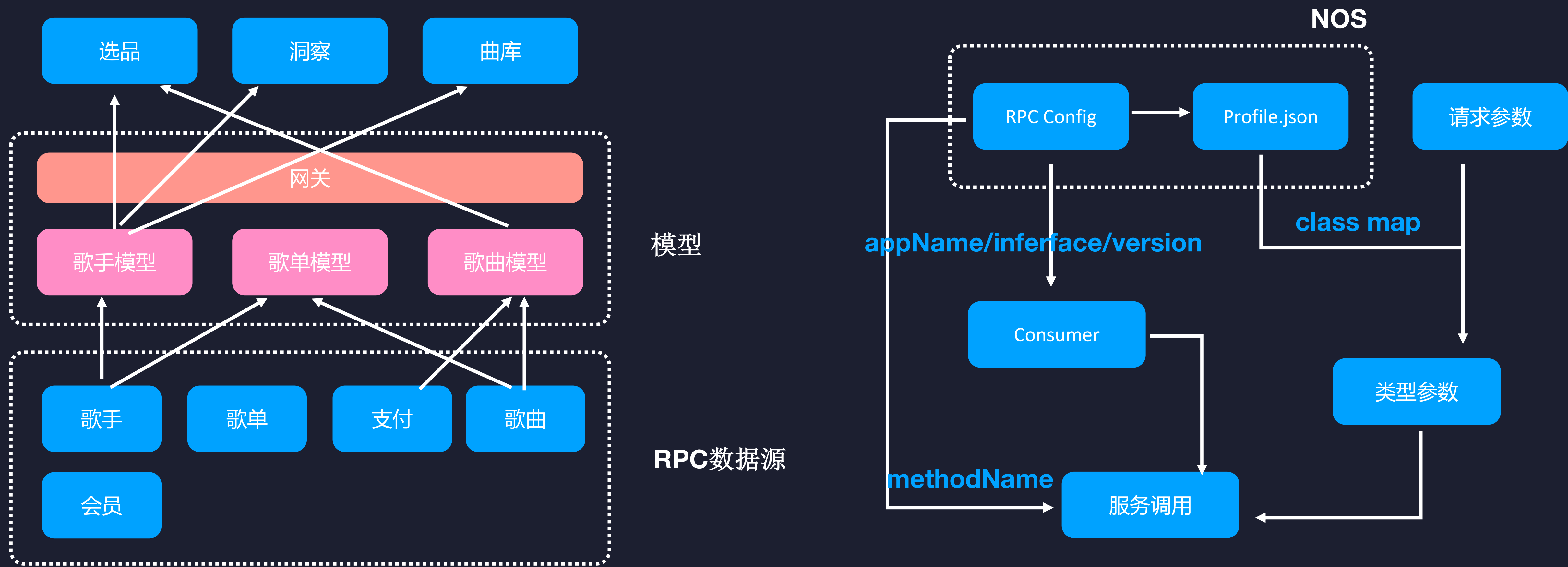
create(data) {
  return await prisma.user.create({
    data,
  })
}

remove(id) {
  return await prisma.user.delete({
    where: id
  })
}

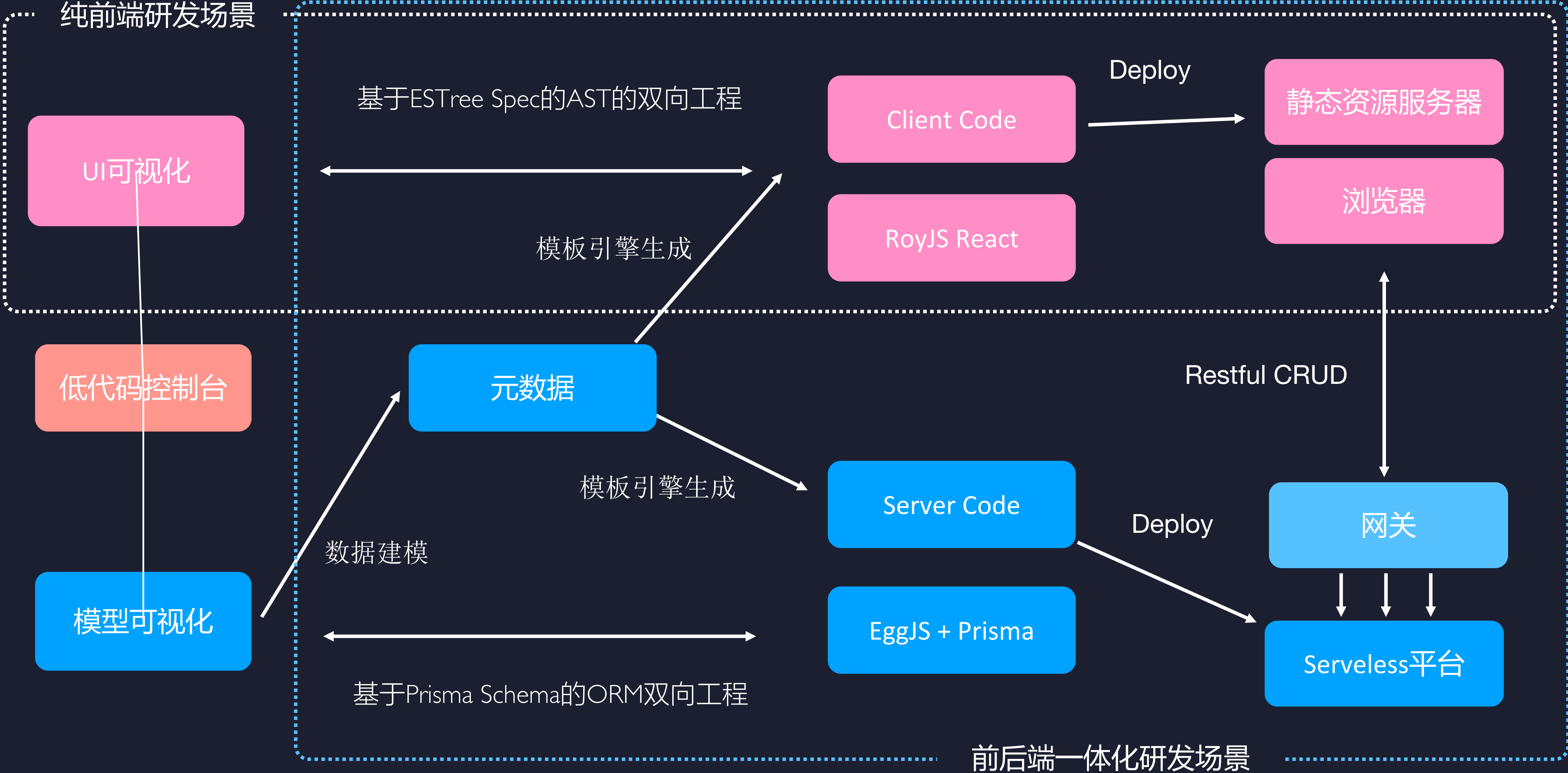
update(data) {
  return await prisma.user.update({
    data,
    where: {
```

低代码平台的核心 - 模型驱动 - RPC泛化调用

大多数情况下模型并不等于数据库的实体，更多是**DTO**的聚合体



低代码平台 - Put All Together



云音乐低代码平台后续规划

Snowpack
Vite
Code sandbox



复杂的后端逻辑如何解？



Java代码的双向工程

最后一句话

一定程度上决定技术产品的应用范围的主要是运营

THANKS

QCon⁺ 案例研习社