再谈Vue SSR

About Me

@天翔Skyline

快手前端工程师

欢迎体验

https://live.kuaishou.com

回顾一下

定位

Web Page → Web App

VUE CONF HANGZHOU 2018

我需要

组件化能力

响应式编程能力

Vue.js

问题

解决方案

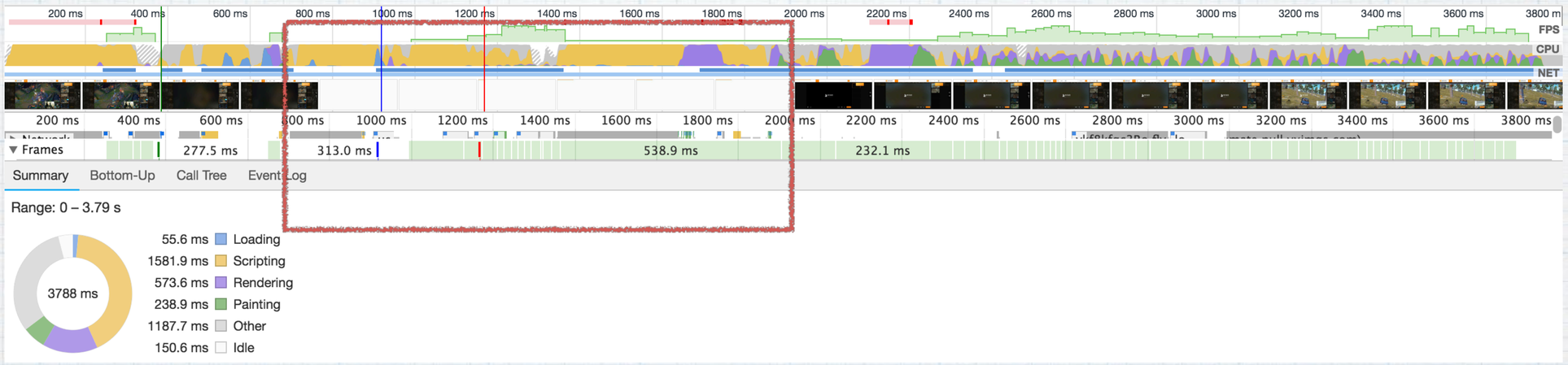
~~Vue SPA~~

VUE SSR

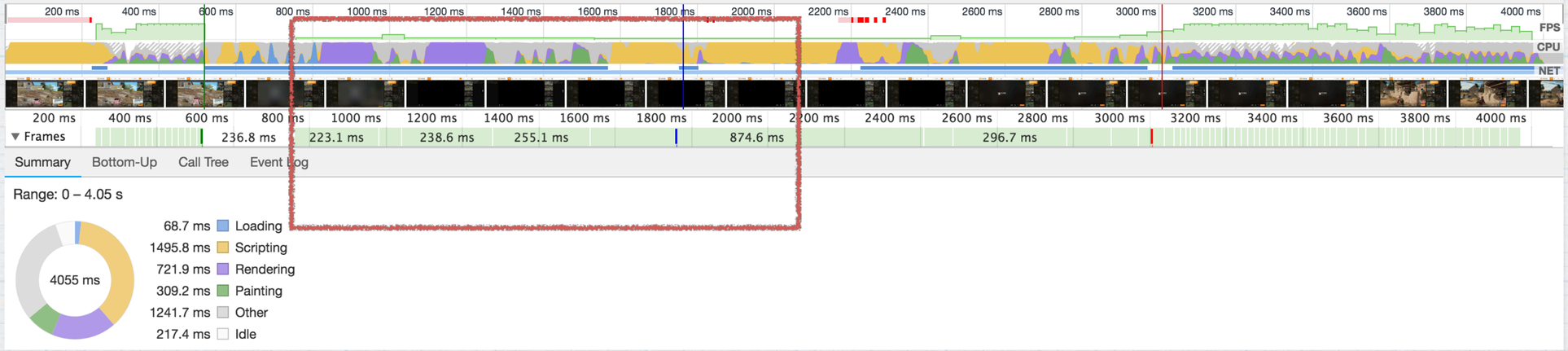
VUE CONF
HANGZHOU 2018

首屏

SPA


FPS
CPU
NET
Frames
277.5 ms
313.0 ms
538.9 ms
232.1 ms
Summary
Bottom-Up
Call Tree
Event Log
Range: 0 – 3.79 s
3788 ms
55.6 ms Loading
1581.9 ms Scripting
573.6 ms Rendering
238.9 ms Painting
1187.7 ms Other
150.6 ms Idle

SSR


FPS
CPU
NET
Frames
236.8 ms
223.1 ms
238.6 ms
255.1 ms
874.6 ms
296.7 ms
Summary
Bottom-Up
Call Tree
Event Log
Range: 0 – 4.05 s
4055 ms
68.7 ms Loading
1495.8 ms Scripting
721.9 ms Rendering
309.2 ms Painting
1241.7 ms Other
217.4 ms Idle

SEO

Google

快手游戏直播

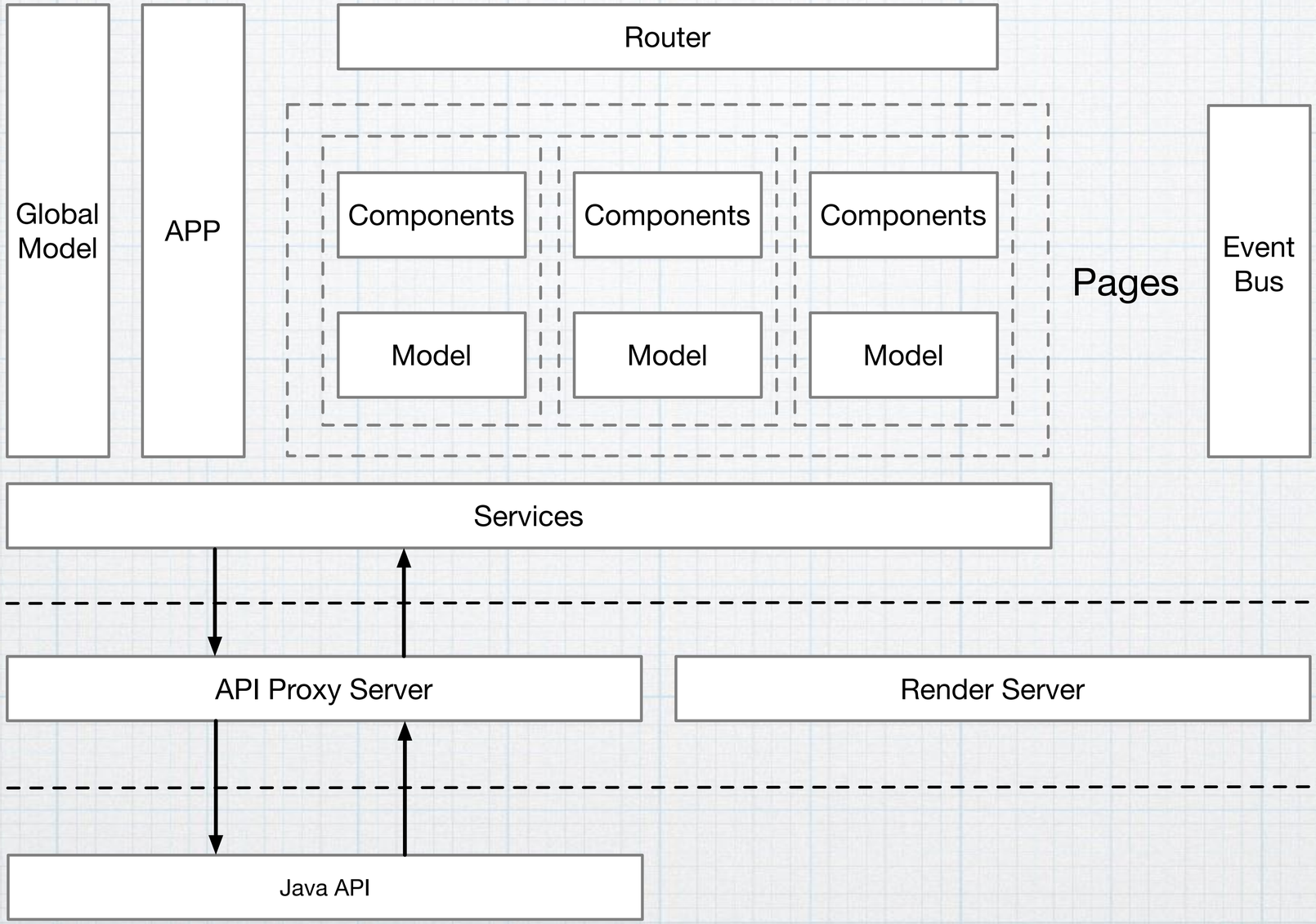
全部 新闻 图片 视频 更多 设置 工具

找到约 6,690,000 条结果 （用时 0.65 秒）

快手直播: 首页
https://live.kuaishou.com/
快手游戏直播，又快又好的游戏直播。 ... 直播推荐 大家都在看的直播间. 蓝光. 静美玩新图 穿越火线 · CF静美 385. 超清. 圆满成功 梦幻西游 · 梦幻西游是浅画哟 580.
直播伴侣 · 当前热门 · 分类 · 更多 Created with Sketch.
您已访问过该网页 2 次。最后访问日期：18-6-25

直播伴侣-快手直播
https://live.kuaishou.com/live-partner
快手游戏直播，又快又好的游戏直播。 ... 注意：有快手直播权限的小伙伴才能使用直播伴侣哦~. PC 客户端下载 ... 手游伴侣一键开播，畅快玩手游，高清直播无烦恼.

大卫游戏直播-快手直播
https://live.kuaishou.com/u/gggggggggg
快手直播gggggggggg，为您提供精彩直播，❤ 精通各类主机热门游戏🧡 绝地求生游戏时长3000+ （不太是技术主播） 一个可以在单机游戏与竞技游戏游走的男人 ...


Router
Global Model
APP
Components
Model
Components
Model
Components
Model
Pages
Event Bus
Services
API Proxy Server
Render Server
Java API

前 · 架构设计方案

问题 ---- 绕不开的asyncData

我们将在路由组件上暴露出一个自定义静态函数 `asyncData` 。注意，由于此函数会在组件实例化之前调用，所以它无法访问 `this` 。需要将 store 和路由信息作为参数传递进去：

```html
<!-- Item.vue -->
<template>
  <div>{{ item.title }}</div>
</template>

<script>
export default {
  asyncData ({ store, route }) {
    // 触发 action 后，会返回 Promise
    return store.dispatch('fetchItem', route.params.id)
  },
  computed: {
    // 从 store 的 state 对象中的获取 item。
    item () {
      return this.$store.state.items[this.$route.params.id]
    }
  }
}
</script>
```

Component与Model的边界是否可以更加明确?

新的Model需要满足

1.足够声明式

2.足够响应式

3.支持Observable的集成

VUE CONF HANGZHOU 2018

重构

我需要解决

Model
书写方式

Request
Client
注入

组件LifeCycle
VS
状态LifeCycle

数据处理边界

VUE CONF
HANGZHOU 2018

Model
书写方式

之前图方便……直接new Vue(...)

但其实……我只需要Reactive

一个易于接受 / 无歧义的写法

配合Vue的依赖收集能力

Class

+

defineReactive

Request Client 注入

?

SSR与CSR的请求需要区别对待

区别对待axios在server端与client端处理

↓

封装request client

SSR与CSR注入不同的request client

组件LifeCycle
VS
状态LifeCycle

State的生命周期跟随组件生命周期

But 它应该还是一个集中式的Store管理（by 勾股）

Component注入代替asyncData

新建的同时完成初始化数据获取

集中管理

数据处理边界

?

过滤过程一定要在Server端进行

转换工作既可以在Server端，也可在Client端

现有API Proxy与Service有一定重叠性

Node: 数据的声明 + 转换

SSR: 数据的聚合 / 过滤

```
export default class Search extends BaseModel {
    public static namespace = 'search';
    public currentUser: any;
    public limit = 5;
    public get searchHot() {
        return this.response.searchHot;
    }
    changeLimit(limit) {
        this.limit = limit;
    }
    @apolloQuery({
        query: gql`
            query SearchHot($limit: Int) {
                searchHot(limit: $limit) {
                    hotWords
                }
            }
        `,
        initState: {
            searchHot: {
                hotWords: [],
            },
        },
        // TODO这块类型匹配还得重新搞一下感觉……
        variables(this: Search) {
            return {
                limit: this.limit,
            };
        },
        prefetch: true,
    })
    public response: any;
}
```

```
export default class Live extends Vue {
    @Models(['live', 'feed'])
    $models: {
        live: LiveModel;
        feed: FeedModel;
    };

    TABS = TABS;

    selectedTab = 'detail';

    @Meta
    metaInfo() {
        return {
            title: `${this.$models.live.user.name}-快手直播`,
        };
    }
```

Now

```
export function createApp({
    graphqlClient = createApolloClient(),
    inApp = false,
    platform = '',
}) {
    const router = createRouter();
    const store = new Store(graphqlClient);

    const app = new Vue({
        router,
        store,
        render: h => h(App),
        provide() {
            return {
                inApp,
                platform,
            };
        },
    });

    return {app, router, store};
}
```

```
import {registerModel} from '@/packages/vue-apollo-model';
import UserModel from '@/models/user';
import LiveModel from '@/models/live';
import SearchModel from '@/models/search';
import FeedModel from '@/models/feed';
import VideoModel from '@/models/video';
import GameDetailModel from '@/models/gameDetail';
import GameIndexModel from '@/models/gameIndex';
import GameCategoryModel from '@/models/gameCategory';

registerModel(UserModel);
registerModel(LiveModel);
registerModel(SearchModel);
registerModel(FeedModel);
registerModel(VideoModel);
registerModel(GameDetailModel);
registerModel(GameIndexModel);
registerModel(GameCategoryModel);
```

Now

GraphQL

over restful

GraphQL

https://www.graphql.org/

GraphQL

Describe your data

```
type Project {
  name: String
  tagline: String
  contributors: [User]
}
```

Ask for what you want

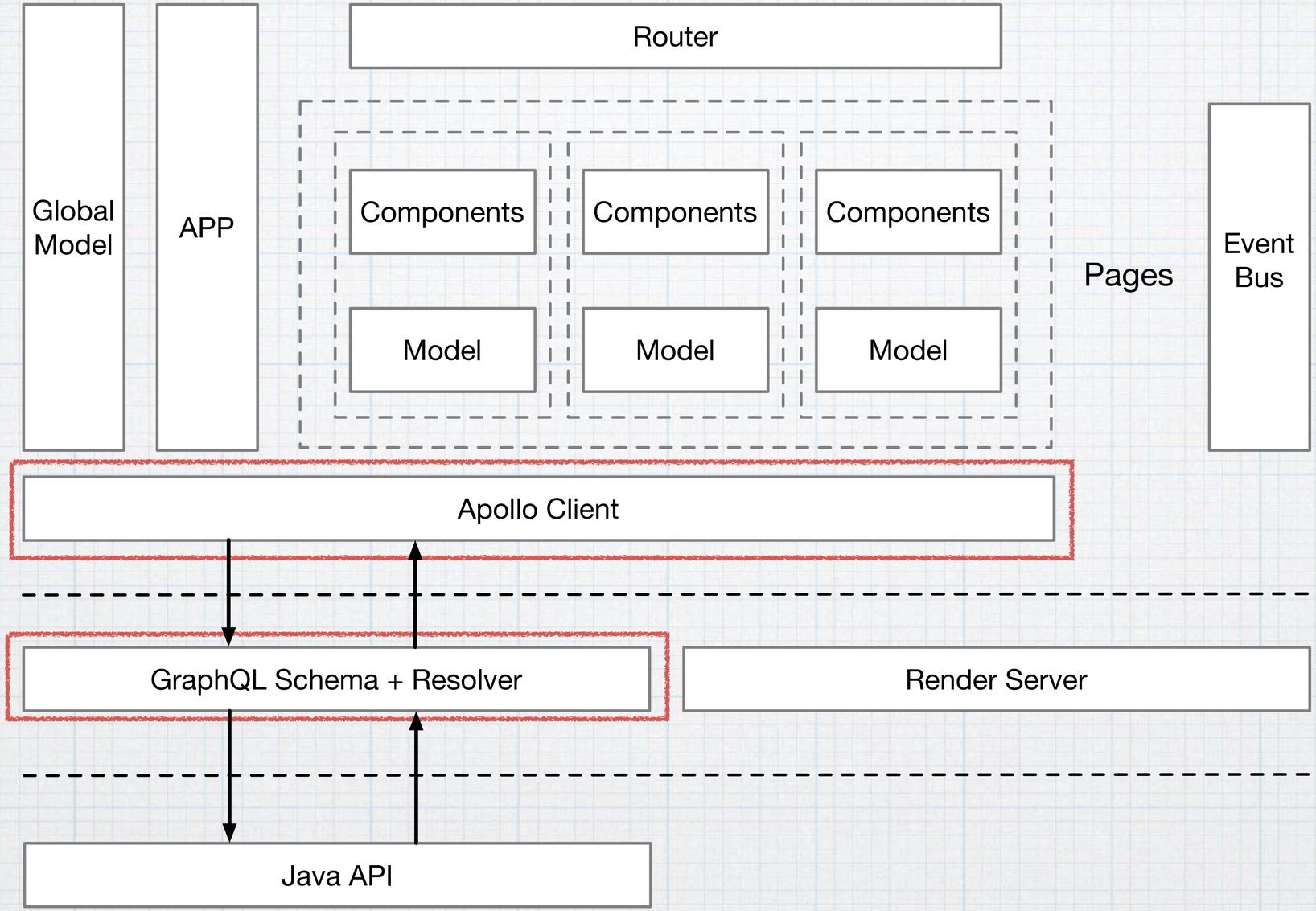
```
{
  project(name: "GraphQL") {
    tagline
  }
}
```

Get predictable results

```
{
  "project": {
    "tagline": "A query language for APIs"
  }
}
```

Get Started

Learn More


Global Model
APP
Router
Components
Components
Components
Model
Model
Model
Pages
Event Bus
Apollo Client
GraphQL Schema + Resolver
Render Server
Java API

现 · 架构设计方案

Observable

状态管理，等同于状态结构及其变化过程

你可以从描述状态入手，以某个时刻的状态为侧重点，淡化其变迁过程

也可以从变迁过程入手，侧重某个过程对数据的变动

By 民工精髓

举个栗子

杨早早💕破二百万

绝地求生 › 欢迎乘坐早早航班~ 2小时前开播

分享 | 举报 ＋关注

6020 7.1w 159.4w

你队友是外国人吗?
是想去天然珍珠^_^wwww因为我ww问问＇w^_

6666

直播间宝宝们
帮我@一下送礼物的大哥
穿云箭上车+V
三根永久车位
晚上带大哥
心愿单
棒棒糖 9999/9999
么么哒 1586/3333
安排 9722/9999
心愿回馈
求大哥投食~

48 存活

爱粉格式：XX早早
语音频道：29655883
粉丝群：865470303

弹幕 蓝光

0 ＋充值

在线观众

聊天室

悟空呀，亮了 x18
送安排
猪嘉明，：袁隆平

悟空呀， 送

陪你到世界的终结479 点亮了

悟空呀， 送

悟空呀， 送

自己 Skyline0705：6666

AA 昔日舞曲：你队友是外国人吗?

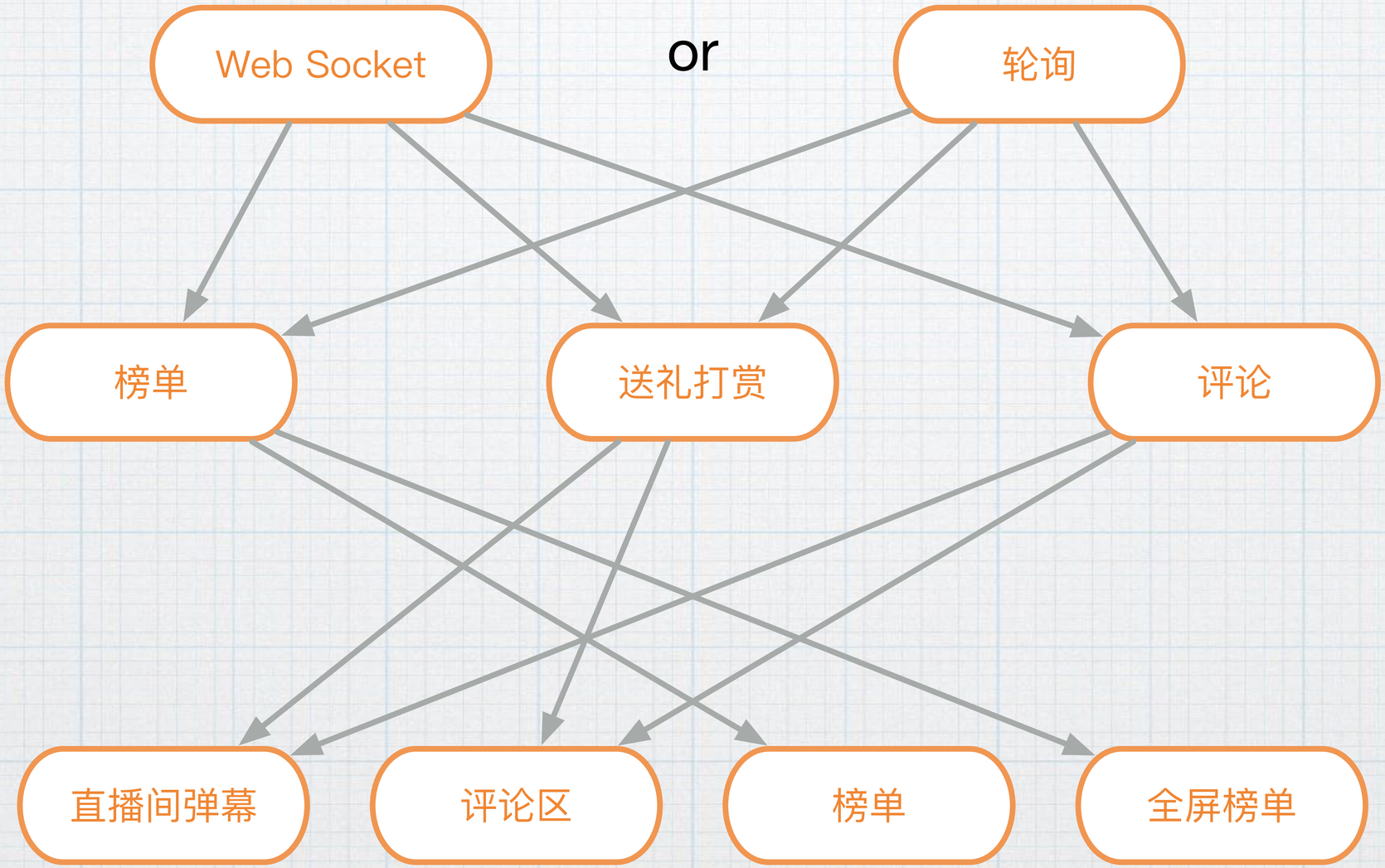
协驰汽车维修：是想去天然珍珠^_^wwww因为我ww问问＇w^_^岳

辉弟..！ 点亮了

悟空呀， 送

快去和大家一起互动吧 发送

消费关系图



在此基础上

1 个人发送的评论、送礼要实时展示出来，不能等待ws数据返回再显示

2 小直播间的榜单可能会有延迟，超过xxx没有下发，你就去主动请求拉一下

我们试着传统方式写一下

```
const commentFeeds = [];
const watchingUser = [];

function fillMessageToEachList(payload) {
    if (payload.watchingUser && payload.watchingUser.length) {
        watchingUser.push(...payload.watchingUser);
    }
    if (payload.commentFeeds && payload.commentFeeds.length) {
        commentFeeds.push(...payload.commentFeeds);
    }
}

function pulling() {
    setInterval(async () => {
        const payload = await axios.get('/feed');
        fillMessageToEachList(payload);
    }, 1000);
}

ws.on('message', fillMessageToEachList);

ws.on('error', startPolling)
```

再考虑下刚才要加的两个需求……

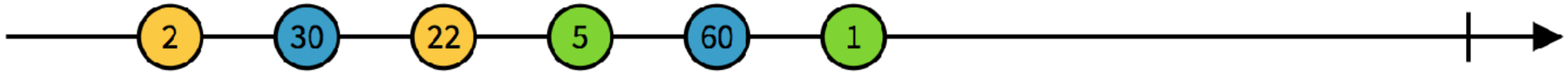
画面太美，真的想不出来代码会写成什么鸟样

试着从变迁过程入手……

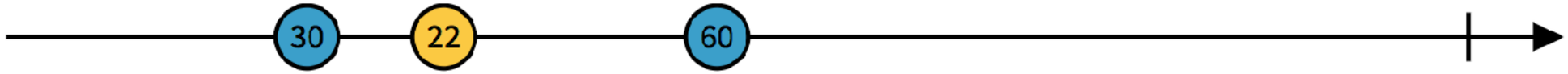
先学弹珠图！

http://rxmarbles.com

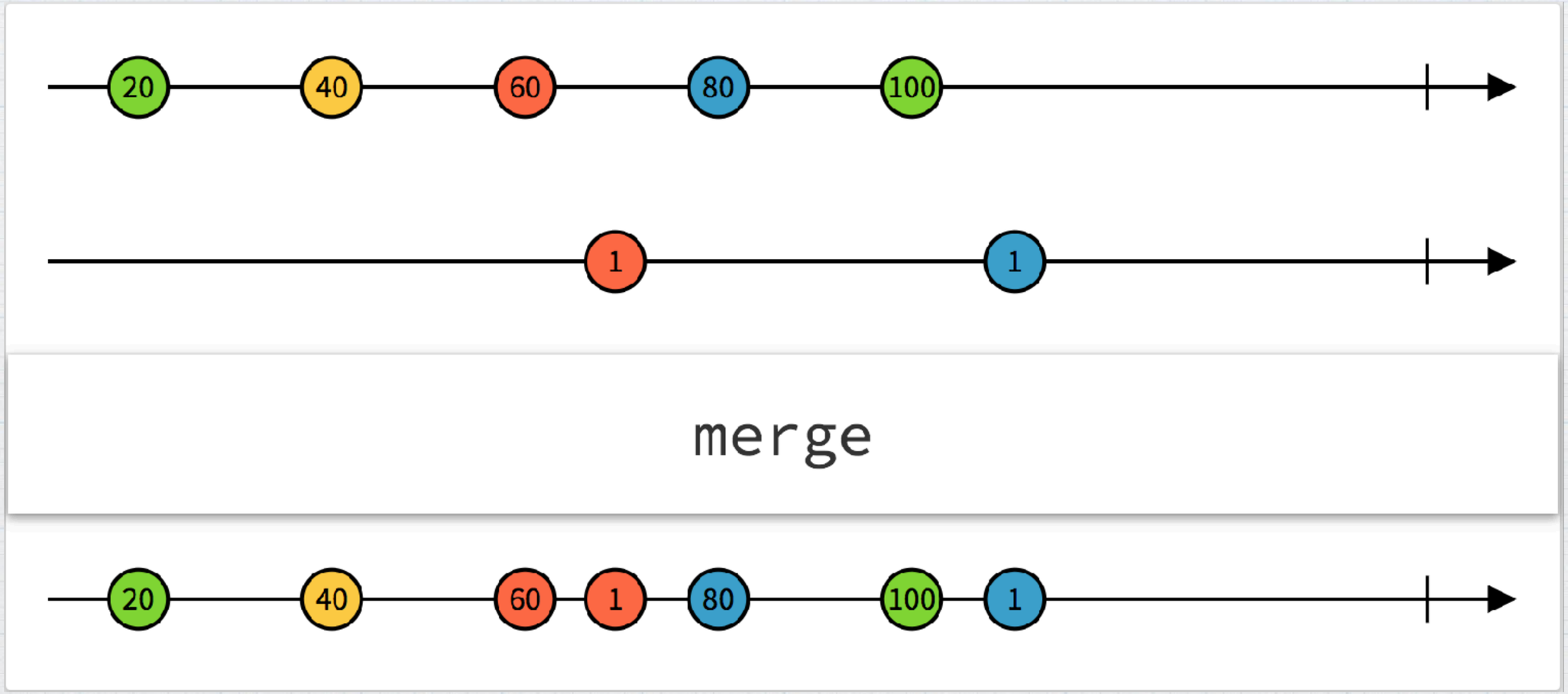
数据分发---- One For All




```
filter(x => x > 10)
```



数据聚合 ---- All For One



重新写一下刚才的需求

```
1   const polling$ = xs.periodic(1000)
2       .map(xs.fromPromise(axios.get('/feed')))
3       .flatten();
4
5   const feeds$ = ws$.replaceError(() => polling$);
6
7   const watchingUserFromStream$ = feeds$
8       .filter(({ watchingUser }) => watchingUser && watchingUser.length);
9       .map(({ watchingUser }) => watchingUser);
10  const commentFeedsFromStream$ = feeds$
11      .filter({ commentFeeds } => commentFeeds && commentFeeds.length)
12      .map({ commentFeeds } => xs.of(...commentFeeds))
13      .compose(flattenConcurrently);
14
```

再加上刚才的两个需求

```
1  const polling$ = xs.periodic(1000)
2      .map(xs.fromPromise(axios.get('/feed')))
3      .flatten();
4
5  const feeds$ = ws$.replaceError(() => polling$);
6
7  const watchingUserFromStream$ = feeds$
8      .filter(({ watchingUser }) => watchingUser && watchingUser.length);
9      .map(({ watchingUser }) => watchingUser);
10
11 const watchingUserFromPolling$ = watchingUserFromStream$
12     .buffer(xs.periodic(20000))
13     .filter(data => !data.length)
14     .map(xs.fromPromise(axios.get('/watchingUser')))
15     .flatten();
16
17 const watchingUser$ = xs.merge(watchingUserFromStream$, watchingUserFromPolling$);
18
19 const commentFeedsFromStream$ = feeds$
20     .filter({ commentFeeds } => commentFeeds && commentFeeds.length)
21     .map({ commentFeeds } => xs.of(...dropMyComment(commentFeeds)))
22     .compose(flattenConcurrently);
23
24 const myComment$ = xs.fromEvent('#submit', 'click')
25     .map(() => myCommentInfo)
26     .flatten();
27
28 const comments = xs.merge(commentFeedsFromStream$, myComment$);
```

VUE CONF HANGZHOU 2018

完美！

可是，怎么与Vue结合起来呢?

Angular ---- AsyncPipe

https://angular.io/api/common/AsyncPipe

AsyncPipe

PIPE IMPURE

Unwraps a value from an asynchronous primitive.

See more...

```
{{ obj_expression | async }}
```

NgModule

CommonModule

Input value

obj	Observable | Promise

Vue没有支持啊……别急

Async Pipe 只是个语法糖……帮你自动 subscribe 了而已……

我们可以自己干啊，比如：vue-rx

或者

我自己在Model手动subscribe一下就行了…

Model ---- 业务

VUE CONF HANGZHOU 2018

```
export default class Feed extends BaseModel {
    public static namespace = 'feed';
    public isRunning = false;
    public subscriptions() {
        const liveRoom$ = xs.create();

        const feed$ = liveRoom$
            .map((feedInfo: any) => {
                return generateFeedStream(feedInfo);
            })
            .flatten();

        // @ts-ignore
        const feedData$ = feed$.map(feedInfo => feedInfo.data || {});

        const comment$ = liveRoom$
            .map(() => feedData$)
            .flatten()
            .filter(feed => feed.commentFeeds && feed.commentFeeds.length)
            .map(feed => feed.commentFeeds);

        const myComment$ = xs.create();

        const watchingList$ = liveRoom$
            .map(() => feedData$)
            .flatten()
            .filter(feed => feed.watchingUser)
            .map(feed => feed.watchingUser);
```

Model ---- 内部处理

VUE CONF HANGZHOU 2018

```
public startSubscriptions() {
    Object.keys(this.$streams).forEach(key => {
        const rKey = key.lastIndexOf('$') === key.length - 1 ? key.slice(0, -1) : key;
        if (!(rKey in this)) {
            Vue.util.defineReactive(this, rKey, null);
        }
        this.$streams[key].addListener({
            next: val => {
                this[rKey] = val;
            },
        });
    });
}
```

Vue帮我们解决了同步响应式

+

Observable帮我们解决了异步响应式

最后一个问题……

GraphQL Query

我也想要响应式……

The `watchQuery` method returns a `QueryRef` object which has the `valueChanges` property that is an `Observable`.

We can see that the result object contains `loading`, a Boolean indicating if the query is “in-flight.” The observable will only emit once when the query is complete, and `loading` will be set to false unless you set the `watchQuery` parameters `notifyOnNetworkStatusChange` or `returnPartialData` to true. Once the query has completed, it will also contain a `data` object with `currentUser`, the field we’ve picked out in `CurrentUserForProfile`.

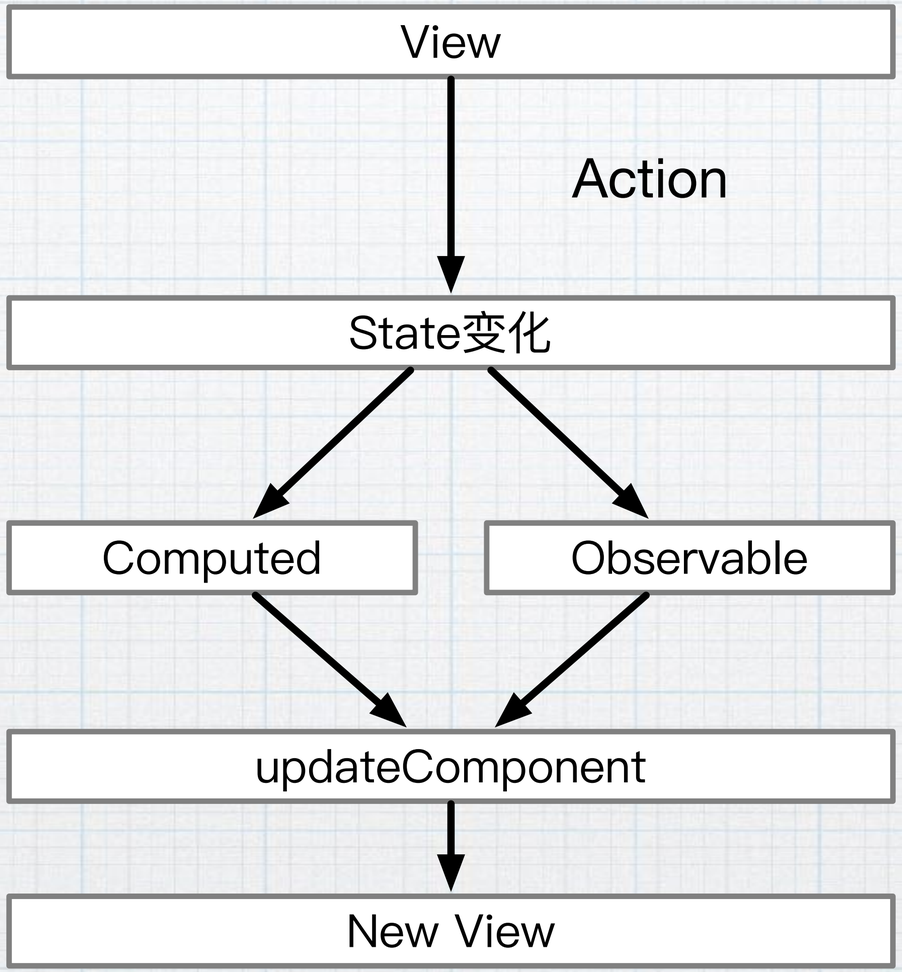
We can expect the `data.currentUser` to change as the logged-in-ness of the client and what it knows about the current user changes over time. That information is stored in Apollo Client’s global cache, so if some other query fetches new information about the current user, this component will update to remain consistent.

It’s also possible to fetch data only once. The `query` method of `Apollo` service returns an `Observable` that also resolves with the same result as above.

```
private init() {
    this.observer = this.client.watchQuery(this.queryOptions);
    if (typeof this.option.variables === 'function') {
        const watcher = this.vm.$watch(
            () => this.variables,
            this.changeVariables.bind(this),
        );
        this.listeners.push(watcher);
    }
}

private async changeVariables() {
    await this.vm.$nextTick();
    this.observer.refetch(this.variables);
}
```

真 · 响应式编程

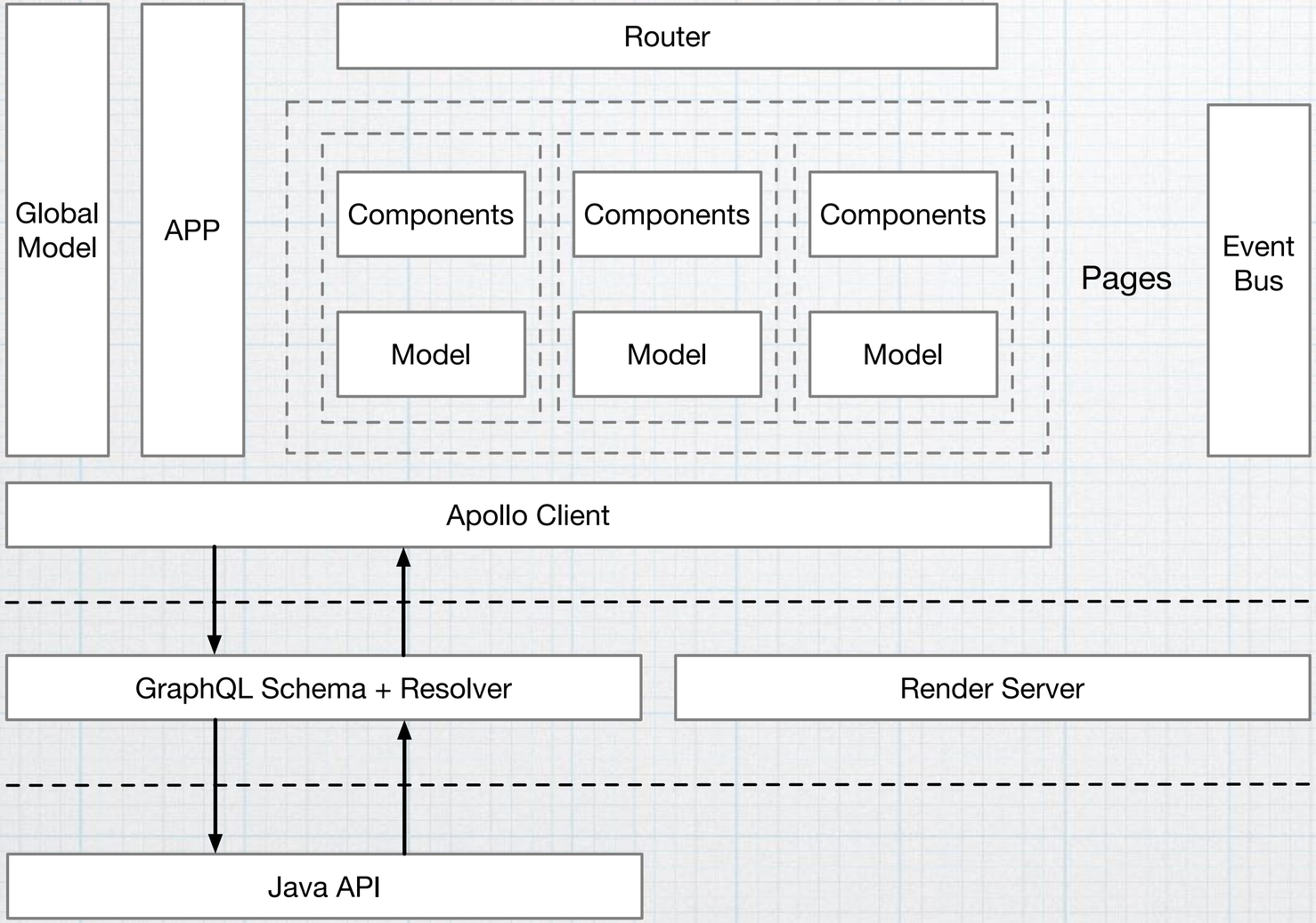


（此处我们看个case）

一切看起来终于美好了起来

加上Apollo Client的cache能力，如虎添翼！

现 · 架构设计方案



1.声明式的Model设计

2.多Store设计的数据管理

3.全链路的响应式数据流

4.灵活的cache逻辑处理

Thanks

Q / A