Teambition 数据层重构经验分享

ReactiveDB powered by RxJS & lovefield

Teambition 前端高级工程师

龙逸楠

自我介绍

龙逸楠

- 2年前端经验
- Teambition 高级前端工程师
- TypeScript RxJS

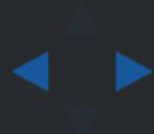
Contact

知乎: https://www.zhihu.com/people/Broooooklyn

github: https://github.com/Broooooklyn

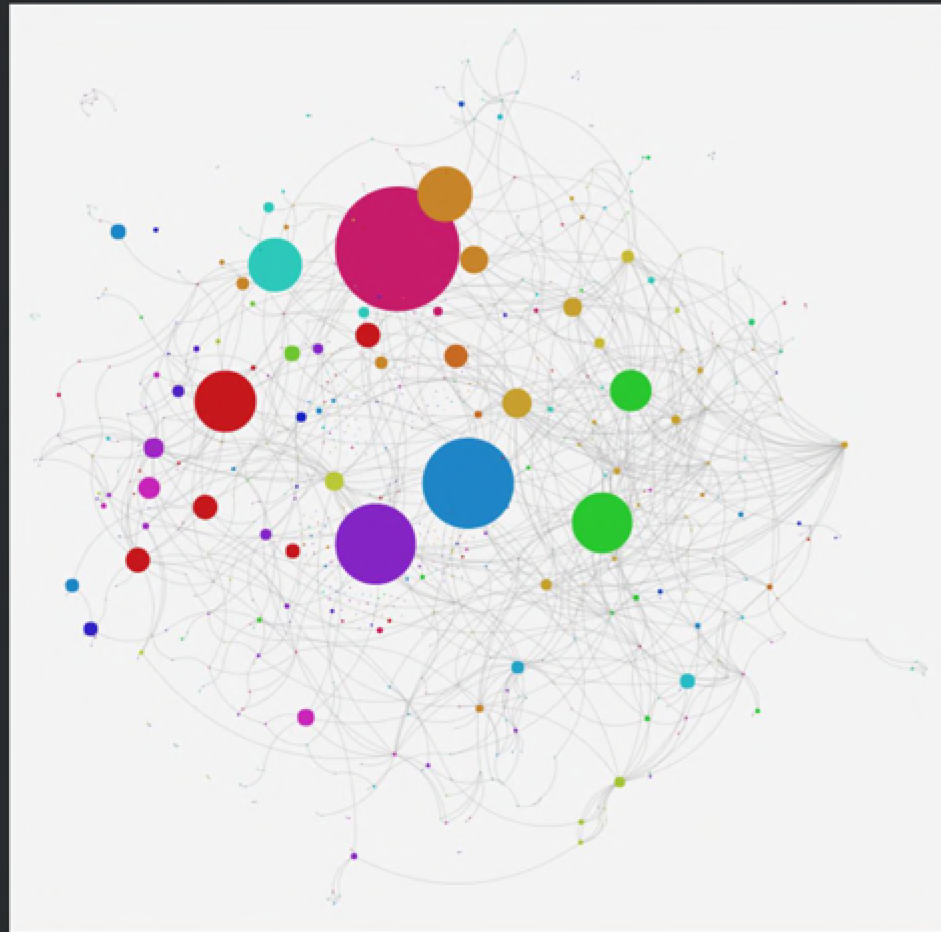
微信: lynweklm

- Teambition 的业务特点
- 各种数据层设计的对比与分析
- **介绍 ReactiveDB**



数据的种类多，数据间的关联性强

60+ Schemas，40+ 关联



任务分组 任务， 任务列表 待处理 ① ② 更多

在 Demo 中逾期的任务

执行者 太狼 ③ | 截止时间 2月1日 18:00 | 优先级 普通 | 重复 不重复

添加备注

逾期的子任务1 ④ 周三 18:00 ③

添加子任务

⑧ Tag

添加提醒

关联的分享

③ Post 7 ⑤ ReactiveDB Demo

post 7 content

关联内容

参与者 ③

龙逸楠， ⑥ 赞了

显示较早的 1 条动态

太狼 认领了任务 2月27日 21:59

太狼 开启了任务邀请共享 ⑦ 昨天 14:07

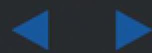
@提及他人，按 Ctrl+Enter 快速发布

1. Tasklist 类型的数据
2. Stage 类型的数据
3. Member 类型的数据
4. Subtask 类型的数据
5. ObjectLink 类型的数据
6. Like 类型的数据
7. Activity 类型的数据
8. Tag 类型的数据

```
{
  "_id": "58b42f74428e764903e86395",
  "_creatorId": "54cb6200d1b4c6af47abe570",
  "_projectId": "58b42ef1c655de4203cad186",
  "_tasklistId": "58b42ef1428e764903e8636e",
  "_stageId": "58b42ef1428e764903e8636f",
  "involveMembers": [
    "54cb6200d1b4c6af47abe570",
    "556466f750e9c0503d58a37b"
  ],
  "dueDate": "2017-02-01T10:00:00.000Z",
  "note": "我逾期啦",
  "content": "在 Demo 中逾期的任务",
  "project": {
    "_id": "58b42ef1c655de4203cad186",
    "name": "ReactiveDB Demo"
  },
  "creator": {
    "_id": "54cb6200d1b4c6af47abe570",
    "name": "龙逸楠",
    "avatarUrl": "https://striker.teambition.net/thumbnail/110f238c339c7dacefbc
  },
  "tagIds": [ "58b42ef1428e724303e2636f" ],
  "stage": {
    "_id": "58b42ef1428e764903e8636f",
    "name": "待处理"
  },
  "executor": {
    "_id": "556466f750e9c0503d58a37b",
```

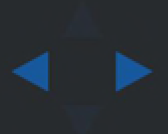
实时性要求高
几乎所有数据都需要通过
WebSocket 更新

针对这种业务场景，我们是如何设计数据层的？



Teambition 技术演进史

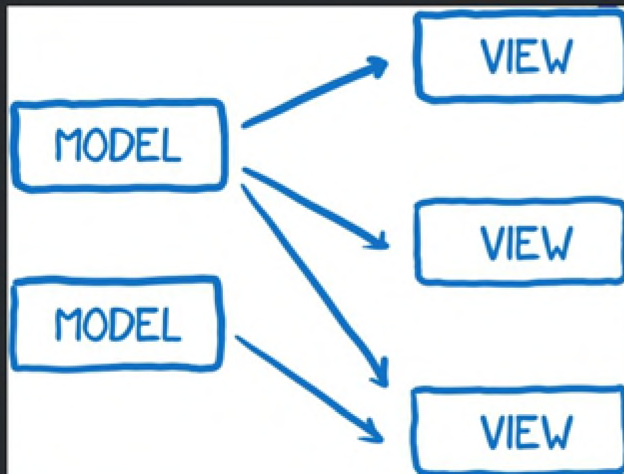
- 2013~2015 Backbone
- 2016
    1. Redux + Normalizr + Reselect
    2. K-V based Cache + RxJS
    3. Lovefield + RxJS (ReactiveDB) + redux-observable + redux
- 2017 ...

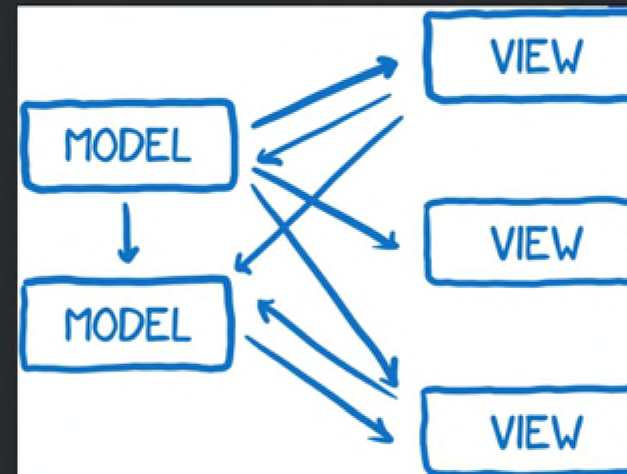


Backbone

- Event Driven
- 同一份数据多个实例，相互关联
- 通过 Collection / Model 切分业务逻辑，Controller 需要自行组合数据

Backbone + jQuery





```
class TaskModel extends Model {
    listen() {
        this.socket.on('change', patch => {
            this.set(patch)
        })
    }
}
class TaskView {
    listenTo() {
        this.taskModel.on('change', taskModel => {
            this.renderTask(taskModel)
        })

        this.subtaskCollection.on('new remove change:_taskId', () => {
            this.taskModel.set({
                subtasksCount: this.subtaskCollection.length
            })
        })

        this.stageModel.on('change: name', stageModel => {
            this.renderStage(stageModel)
        })

        this.tasklistModel.on('change', tasklistModel => {
            this.renderTasklist(tasklistModel)
        })

        this.projectModel.on('change', projectModel => {
            this.renderProject(projectModel)
        })
```

TaskCardView

待处理 · 2

在 Demo 中逾期的任务

2月1日 截止 0/1

1 Tag 嘎嘎

321

其它的测试任务

添加任务

执行者	截止时间	优先级	重复
太狼	2月1日 18:00	普通	不重复

456

逾期的子任务2 周三 18:00

添加子任务

Tag 嘎嘎 嘎嘎嘎 321

添加提醒

TaskDetailView

关联的分享

Post 7 ReactiveDB Demo

post 7 content

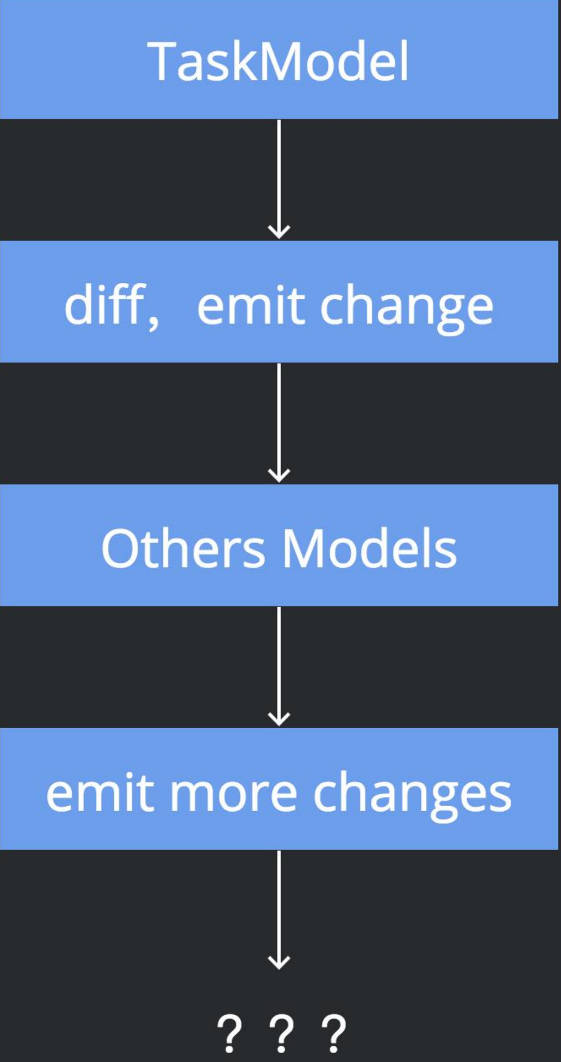
关联内容

参与者

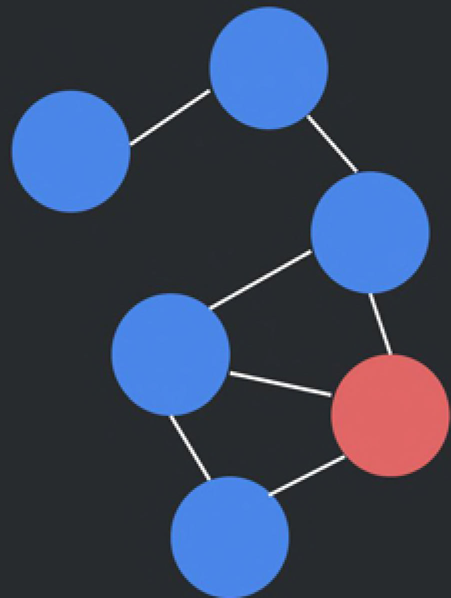
龙逸楠， 1人赞了

显示较早的 9 条动态

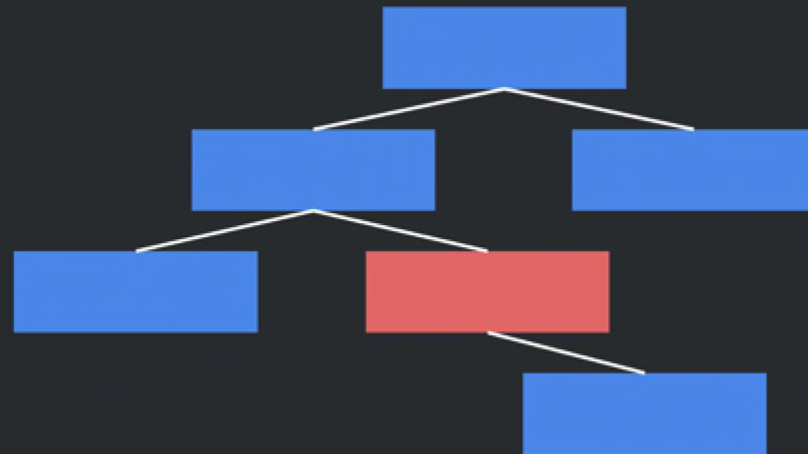
@提及他人，按 Ctrl+Enter 快速发布


TaskModel
diff，emit change
Others Models
emit more changes
？？？

Model



View



Redux + Normalizr + Reselect + React

Backend

Http & Websocket

Redux Store

Normalizr

Component

Reselect

Connected Container

```
// redux middleware
const epic = action$ => action$
    .ofType(`${someAction}`)
    .switchMap(action => {
        const { param }  = action.payload
        return Ajax$.get(param)
          .map(response => otherAction(normalize(response,
          .catch(err => Observable.of(errorAction(err)))
          .takeUntil(`${componentUnmount}`)
    })


// reselect
const selector = createSelector(
    state => state.someModule.entities,
    state => state.someOtherModule.entities.something,
    (dataYouCareAbout, relatedData) =>
        // filter and reselect
)


// container
const mapStateToProps = state => ({
    selector1: selector1(state),
    selector2: selector2(state),
    selector3: selector3(state),
    // ...
})
```

复杂的数据场景处理起来异常费力

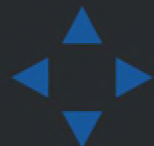
近期的事　任务　日程　收藏　笔记

今天的事 · 5

①

橙色表示：紧急的任务	示例项目3 ②	2月24日 截止
在 Demo 中逾期的任务	ReactiveDB Demo	2月28日 截止
明天任务	示例项目3	今天 18:00 截止
其它的测试任务	ReactiveDB Demo	昨天 开始
今天的日程	ReactiveDB Demo	12:00 - 13:00

```
// reselect

/**
 * 存储未开始的任务
 * 定义:
 * & 未被完成
 * & 未被归档
 * & _projectId 字段匹配
 * &
 * (
 * | 没有截止日期
 *   | 有开始时间 & 开始时间在今天以后
 *   | 没有开始时间
 * | 有截止日期 & 截止日期在今天以后
 *   | 有开始时间 & 开始时间在今天以后
 *   | 没有开始时间
 * )
 */
const taskFilter = taskData => {
    return data._projectId === projectId &&
        !data.isArchived &&
        !data.isDone &&
        (
          !data.dueDate &&
```



Redux + Normalizr + Reselect + React

优点：

- 中心化数据存储
- 单向数据流
- Normalized Data Shape
- 可以在 selector 里面做细粒度的过滤， 不需要在 component 里写 SCU 就能获得相对好的性能

缺点：

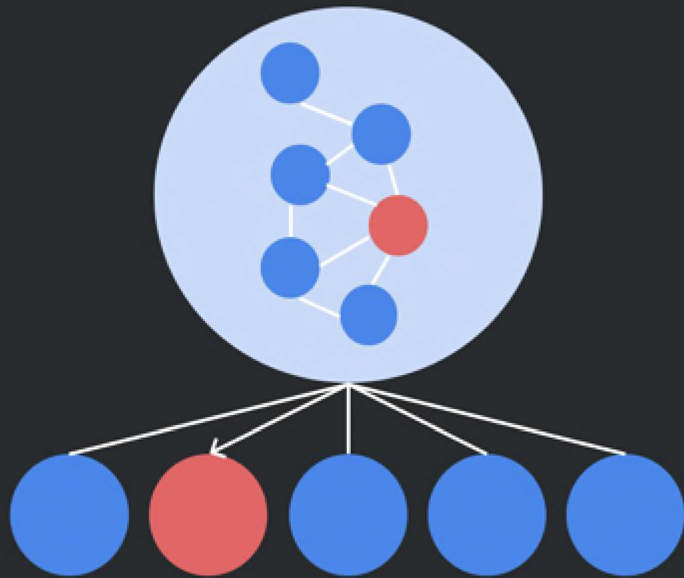
- 无法自动处理数据之间的关联关系
- 需要为 Container 写大量selector来过滤状态的变化
- 开发人员疲于编写大量代码用于满足产品上需要的排序、分页等需求

K-V based Cache + RxJS

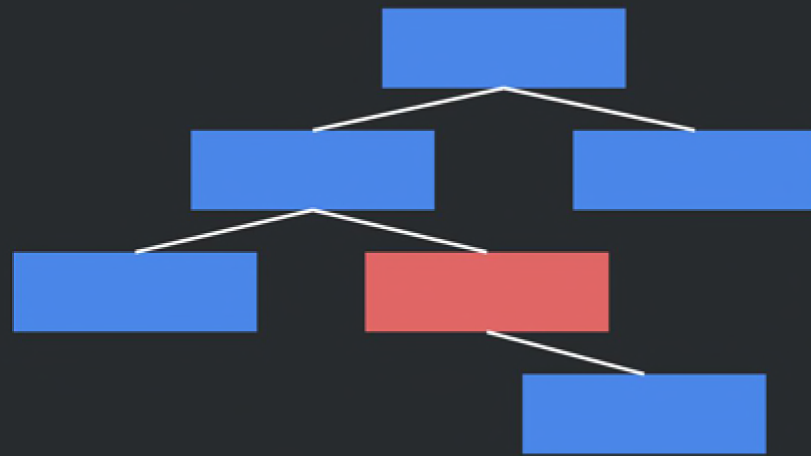
```
class Database {
    cache = new Map()

    save(data, primaryKey = '_id') {
        // normalize
    }

    update(primaryKey, patch) {
        // ...
    }

    get(primaryKey) {
        // reselect
        return Observable.create(observer => {
            observer.next(this.cache.get(primaryKey))
        })
           /**
            * updateStream$ = socket.getUpdate(primaryKey)
            *     .merge(http.getUpdate(primaryKey))
            **/
            .combineLatest(updateStream$)
            .map(([data, patch]) => { ...data, patch })
            .takeUtil(deleteStream$)
    }
}
```

```
class TaskSchema {
    @primaryKey _id: string
    _exectorId: string
    _tasklistId: string
    _stageId: string
    _projectId: string
    content: string
    note: string
    @assosiate('_exectorId') exector: ExectorSchema
    @assosiate('_projectId') project: ProjectSchema
    @assosiate('_tasklistId') tasklist: TasklistSchema
    ...
}

class ProjectSchema {
    @primaryKey _id: string
    _id: string
    _organizationId: string | null
    name: string
    avatarUrl: string
    @assosiate('_organizationId') organization: OrganizationSche
    @assosiateMany('_projectId') tasklists: TasklistSchema[]
    ....
```

Model



View



K-V based Cache + RxJS

优点：

- 中心化数据存储
- 单向数据流（从整个应用角度来看）
- Normalized Data Shape
- 自动处理数据间的关联关系

缺点：

- 无法做细粒度的 diff 以及过滤
- 数据层内的数据流是网状的，出问题了很难追踪
- 开发人员疲于编写大量代码用于满足产品上需要的数据查询条件、排序、分页等需求

我们理想中的解决方案是什么样子的?

1. 中心化的数据管理：

- 统一维护所有客户端接受到的数据
- 保存 Normalized Data , 拥有同一个 Id 的数据只存一份

2. 查询逻辑降噪

- 以声明式的形态定义查询需要的字段及其关联属性
- 直接处理简单的排序、过滤、分页需求，一定程度上解放生产力

3. Observing & Cancelling

- 允许订阅所查询数据的变化，在不需要时自动取消订阅

Nomalized Data

Backend Database

Nested Data

API Service

Other Client

Nomalized Data

????

Frontend

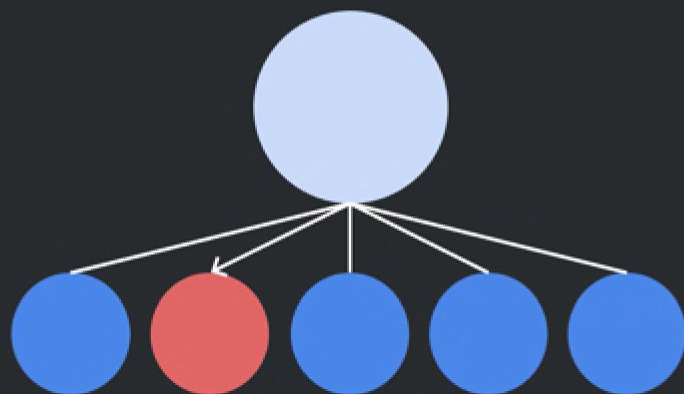
Observed Nested Data

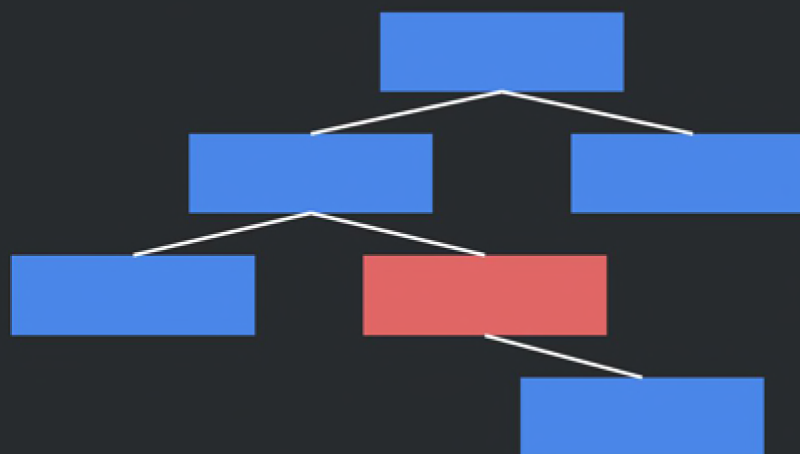
Components

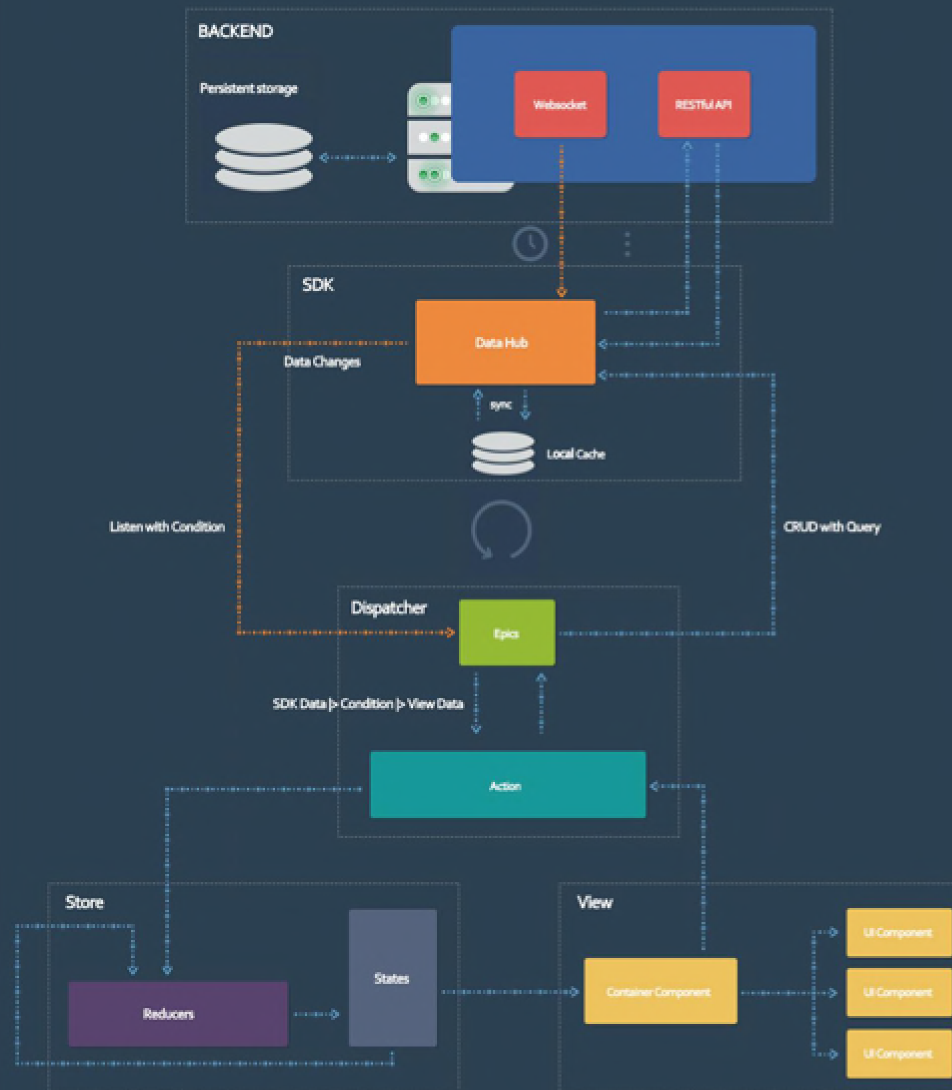
我们期望的架构

Model

View







Application Diagrams (draft)
BACKEND
Persistent storage
Websocket
RESTful API
SDK
Data Hub
Data Changes
sync
Local Cache
Listen with Condition
CRUD with Query
Dispatcher
Epics
SDK Data |> Condition |> View Data
Action
Store
Reducers
States
View
Container Component
UI Component
UI Component
UI Component

Lovefield

Basic

- Relational
- 支持 Observe
- Query 支持 Predicate、limit、skip、OrderBy
- Index-able Data store

More

- Transaction
- IndexedDB/LocalStorage 等持久化策略

```
// Define Schema and Connect
const schemaBuilder = lf.schema.create('teambition', 1)

schemaBuilder.createTable('Task')
    .addColumn('_id', lf.Type.STRING)
    .addColumn('content', lf.Type.STRING)
    .addColumn('created', lf.Type.STRING)
    .addColumn('dueDate', lf.Type.STRING)
    .addColumn('priority', lf.Type.INTEGER)
    // ...
    .addPrimaryKey(['_id'])
    .addNullable(['content', 'created', 'dueDate', 'priority'])

schemaBuilder.createTable('Member')
    .addColumn('_id', lf.Type.STRING)
    .addColumn('name', lf.Type.STRING)
    .addColumn('avatarUrl', lf.Type.STRING)
    // ...
    .addPrimaryKey(['_id'])
    .addNullable([/** ... */])

//...

// Schema is defined, now connect to the database instance.
schemaBuilder
    .connect({ /** storeType: lf.DataStoreType.INDEXED_DB */ })
    .then(db => {
      // Schema is not mutable once the connection to DB has establis
    })
```

```
const taskTable = db.getSchema().table('Task')
const memberTable = db.getSchema().table('Member')
const query = db.select(
    taskTable._id, taskTable.content, memberTable._id,
    memberTable.name, memberTable.avatarUrl
)
    // join
    .from(taskTable, memberTable)
    // oh no....
    .where(lf.op.and(
        taskTable.created.lt(moment().startOf('day').valueOf(
        taskTable._executorId.eq(member._id)
    ))
    .orderBy(taskTable.priority, lf.Order.DESC)
    .orderBy(taskTable.dueDate, lf.Order.DESC)
    // 第二页，每页20条数据
    // Sad story，因为查询的结果是笛卡尔积，这里不能这么查询
    .limit(20)
    .skip(20)

// Select Data
query.exec().then(data => // 笛卡尔积，Graph it)

// Observe
db.observe(query, diff => // effect)

//Unobserve
```

```
const Tasks = [Task1, Task2, Task3 ...]
const Subtasks = [Subtask1, Subtask2, Subtask3 ...]
const Projects = [Project1, Project2]

const result = [
    {
        Task: Task1,
        Subtask: Subtask1,
        Project: Project1
    },
    {
        Task: Task1,
        Subtask: Subtask2,
        Project: Project1
    },
    {
        Task: Task2,
        Subtask: Subtask3,
        Project: Project1
    },
    {
        Task: Task2,
        Subtask: Subtask4,
        Project: Project1
    }
    //
```

缺点

- Low level API，数据的定义与查询等操作的代码较为冗长
- 无法自动处理客户端出现的数据类型不一致 (ISO string & Date)
- 不支持传统 RDBMS 的 SubQuery
- 需要手动处理数据的订阅与取消

ReactiveDB

- 基于 Lovefield，继承其所有强大的功能
- 使用 RxJS， 天然契合 Observe
- 直观的定义数据类型及其关联属性并且自动处理 Join 后的数据
- 扩展 Lovefield 的数据类型，自动对后端接口的类型做transform
- 声明式的查询逻辑

```
// 定义 Schema
ReactiveDB.defineSchema('Task', {
  _creatorId: {
    type: RDBType.STRING
  },
  _executorId: {
    type: RDBType.STRING
  },
  _id: {
    type: RDBType.STRING,
    primaryKey: true
  },
  _projectId: {
    type: RDBType.STRING
  },
  _stageId: {
    type: RDBType.STRING
  },
  _tasklistId: {
    type: RDBType.STRING
  },
  accomplished: {
    type: RDBType.DATE_TIME
  },
  content: {
    type: RDBType.STRING
  },
```

```
// 查询数据
const QueryToken = ReactiveDB.get('Task', {
  where: {
    created: {
      $gt: moment().startOf('day').valueOf()
    }
  },
  fields: [
    '_id',
    'content',
    {
      member: ['_id', 'name', 'avatarUrl']
    }
  ],
  // yes it works
  skip: 20,
  limit: 20,
  orderBy: [
    { fieldName: 'priority', orderBy: 'DESC' },
    { fieldName: 'dueDate', orderBy: 'DESC' }
  ]
})
```

QueryToken

- concat : 处理列表
- combine : 关联多个 Query
- values : 对 Query 做直接查询求值
- changes : 产生一个 Observable，订阅所有满足 Query 约束数据的变化

Backend

Http Response

WebSocket

Http Request

Datahub

Cache Validate

ReactiveDB

Get

Post Update Patch Delete

Components

Example

https://github.com/teambition/ReactiveDB

https://github.com/teambition/teambition-sdk