About Node.js ORM

XadillaX of Souche

MySQL ORMs

- Sequelize
- Waterline
- bookshelf
- ...

Sequelize

- Schema definition
- Schema synchronization/dropping
- 1:1, 1:M & N:M Associations
- Transactions
- Migrations
- #Bugs#, e.g. multiple primary keys, sync…
- NO column <==> name
- …

Sequelize

```
Foo.findAll({
    where: {
        $or: [{
            title: {
                $like: 'Boat%'
            }
        }, {
            description: {
                $like: '%boat%'
            }
        }]
    }
});
```

SELECT * FROM FOO WHERE title LIKE 'Boat%' OR description LIKE '%boat%'

"Sequelize is powerful and
heavy with lots of bugs."

Waterline

- Deep integration
- column <==> name
- NO Multiple primary keys long ago (?)

Waterline

```
Message.findOne({
    messageId: message.messageId
}).exec(function(err, message) {
});
```

SELECT * FROM MESSAGE WHERE message_id = “foo” LIMIT 0, 1

"More elegant than sequelize,
but not that high completion."

"Sometimes you don't need a that big ORM with slow query like group, join, *like, etc."

写不来英文了

字体难看点，大家别介意

“我不是 DBA，请听我满口胡言。”

–XadillaX

“真的需要大而全吗？你用了百分之多少功能？”

“大型互联网应用真的需要 group by, join 这些操作吗？”

在 Sequelize 一句搞定

```
SELECT * FROM a LEFT JOIN b ON
a.user_id = b.user_id WHERE article_id = 1
ORDER BY updated_at DESC
LIMIT 0, 10
```

减少开发成本

“1. MySQL 关联性能有点差

2. 数据量多的，我们基本上不让在数据层处理

3. 关联复杂，数据很多的，都应该放在中间层”

–某厂美女 DBA

“我司老司机告诉我互联网公司不要在 DB 里做链表操作，任何外键、Join、查两张表的，都可以在业务层实现。”

~屁股群某君

在业务逻辑中搞定

```
SELECT * FROM a WHERE article_id = 1
ORDER BY updated_at DESC
LIMIT 0, 10
```

```
SELECT * FROM b WHERE user_id = ?
```

缓存+拼接

“查询之后用缓存取结果，更新之后删缓存；
硬盘和内存的速度差距 Biubiubiu！”

不同情境不同用法

- 很多互联网应用并没有传统软件数据关系那么复杂
- 业务层加缓存，单表查询基本够用
- 有点拿 MySQL 当 NoSQL 用的感觉
- 万不得已，那就用一下咯

Toshihiko

衍生的适合很多互联网产品的 Node.js MySQL ORM。

“土”

~Toshihiko 的最大特色

土·Toshihiko

- ORM 本身不支持 Group、Join 等低效查询
- ORM 本身不支持事务（暂时没需求）
- 没有外键概念
- 没有多表概念（1:N, M:1, N:M）
- 没有通过模型定义生成 SQL 语句直接同步到数据库
- 简陋（Maybe）
- 暴露 exec 接口可以自己怼入 SQL 语句查询

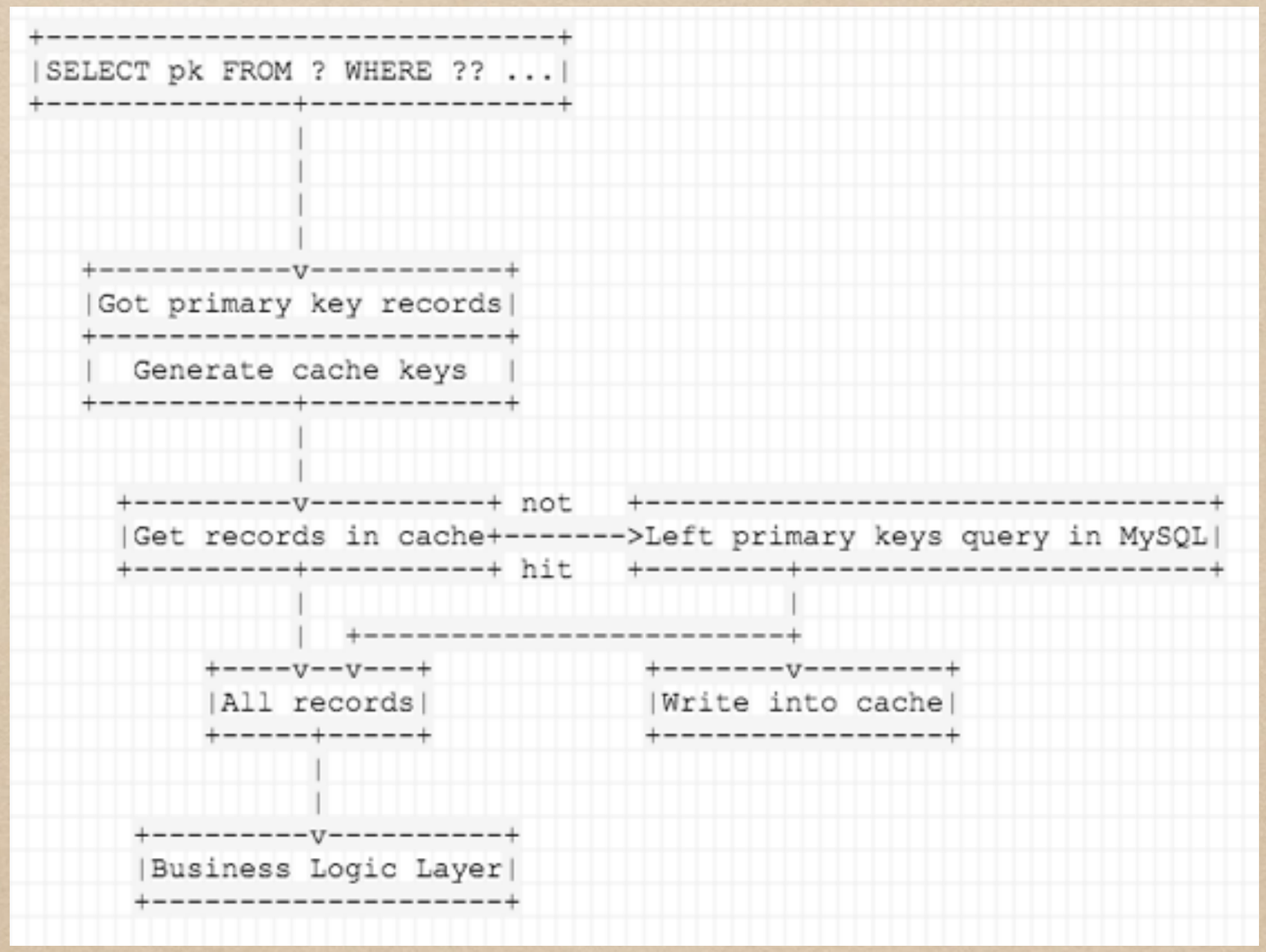
玉·Toshihiko

- name 和 column，暗里一套，明里一套
- 内置缓存逻辑，可插拔可更换缓存层介质
- 简易的查询条件建模（参考 sequelize 和 waterline）
- 自定义数据类型，如 JSON 等
- 使用 mysql2 代替 mysql，benchmark 结果比较好
- Promise 兼容（虽然我不怎么喜欢）
- 暴露原生执行接口 exec

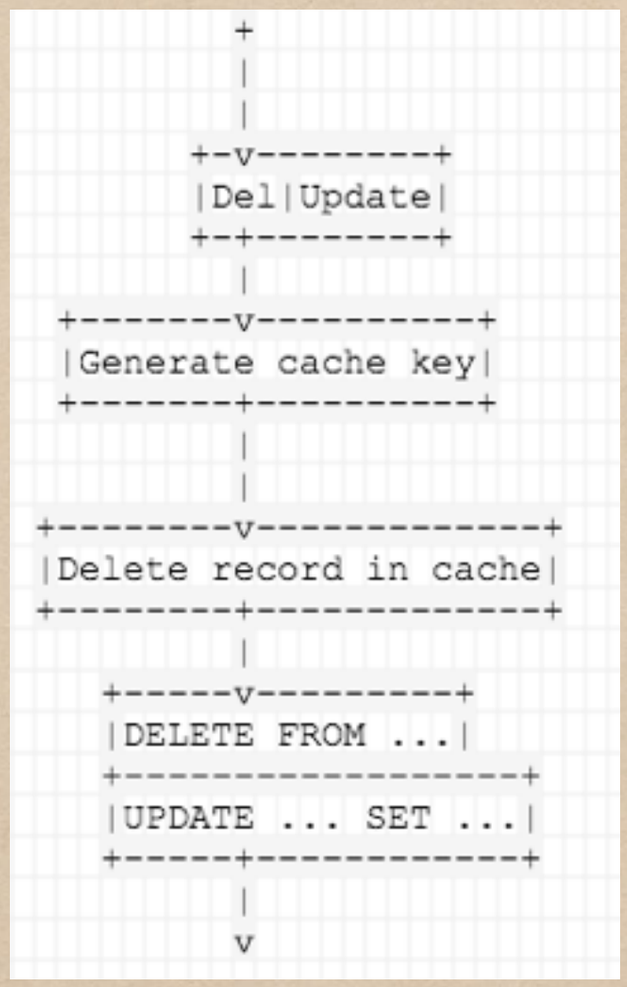
```
Model = toshihiko.define("test", [
    { name: "key1", column: "id", primaryKey: true, type: T.Type.Integer },
    {
        name: "key2",
        type: T.Type.Float,
        defaultValue: 0.44,
        validators: [
            function(v) {
                if(v > 100) return "`key2` can't be greater than 100";
            }
        ]
    },
    { name: "key3", type: T.Type.Json, defaultValue: {} },
    { name: "key4", type: T.Type.String, defaultValue:"Ha!"}
]);
```

“还记得大明湖畔的 `zhuce_id` 吗？现在改了个马甲叫 `userId` 了。”

Toshihiko 缓存层



Toshihiko 缓存层



```
var Model = toshihiko.define("posts", [
    { name: "postId", type: T.Type.Integer, primaryKey: true },
    { name: "userId", type: T.Type.Integer },
    { name: "sex", type: T.Type.String, defaultValue: "female" },
    { name: "area", type: T.Type.String },
    { name: "smallIndustry", type: T.Type.String },
    { name: "bigIndustry", type: T.Type.String },
    { name: "content", type: T.Type.String },
    { name: "imageId", type: T.Type.Integer, allowNull: true },
    { name: "status", type: T.Type.Integer },
    { name: "postedAt", type: T.Type.Integer },
    { name: "endedAt", type: T.Type.Integer },
    { name: "extra", type: T.Type.Json },
    { name: "ups", type: T.Type.Integer },
    { name: "downs", type: T.Type.Integer },
    { name: "top", type: T.Type.Integer, defaultValue: 0 },
    { name: "replies", type: T.Type.Integer, defaultValue: 0 }
]);

var condition = { "$or": [
    { status: 32, bigIndustry: "", smallIndustry: "" },
    { status: 32, bigIndustry: "民航特业", smallIndustry: "" },
    { status: {
        $neq: [ 2, 4 ]
    }, bigIndustry: "民航特业", smallIndustry: "空管" }
]};

var sql = Model.where(condition).makeSQL("find");
var answer = "SELECT `postId`, `userId`, `sex`, `area`, `smallIndustry`, `bigIndustry`, `content`, `imageId`, " +
    "`status`, `postedAt`, `endedAt`, `extra`, `ups`, `downs`, `top`, `replies` FROM `posts` WHERE (((" +
    "`status` = 32 AND `bigIndustry` = \"\" AND `smallIndustry` = \"\") OR (`status` = 32 AND `bigIndustry` = " +
    "\"民航特业\" AND `smallIndustry` = \"\") OR ((`status` != 2 AND `status` != 4) AND `bigIndustry` = " +
    "\"民航特业\" AND `smallIndustry` = \"空管\")))";

answer.should.be.eql(sql);
```

... WHERE (((`status` = 32 AND `bigIndustry` = "" AND `smallIndustry` = "") OR (`status` = 32 AND `bigIndustry` = "民航特业" AND `smallIndustry` = "") OR ((`status` != 2 AND `status` != 4) AND `bigIndustry` = "民航特业" AND `smallIndustry` = "空管")))

Toshihiko 自定义数据类型

`key3` varchar(200) NOT NULL DEFAULT ''

{ name: "key3", type: T.Type.Json, defaultValue: {} }

INSERT INTO .... key3 = "{foo:\"bar\"}"

foo.key3 => { foo: "bar" }

Toshihiko 自定义数据类型

```
var Industry = {};
Industry.name = "Industry";
Industry.needQuotes = true;

Industry.restore = function(parsed) {
    if(!parsed) return "";
    return parsed.big + "|" + parsed.small;
};

Industry.parse = function(orig) {
    if(!orig) return {};
    var temp = orig.split("|");
    return {
        big: temp[0] || "",
        small: temp[1] || ""
    };
};

Industry.equal = function(a, b) {
    return (a.big === b.big && a.small === b.small);
};

Industry.defaultValue = "";

module.exports = Industry;
```

Toshihiko 考验

- 大搜车
- 花瓣网
- 飞秘
- 搜狐（maybe）
- ...

Toshihiko 里面用到的小技巧

- 链式调用
- unicode 卖萌代码
- 跑了单测才能 push
- 跑了 lint 才能提交
- ...

“F & Q”

https://github.com/xadillax/toshihiko

!FIN

Thanks