设计Twitter

相关推荐：

- 面试官：你说对MySQL事务很熟？那我问你10个问题
- 一行代码就能解决的算法题

读完本文，你不仅学会了算法套路，还可以顺便去 LeetCode 上拿下如下题目：

355.设计推特

「design Twitter」是 LeetCode 上第 355 道题目，不仅题目本身很有意思，而且把合并多个有序链表的算法和面向对象设计（OO design）结合起来了，很有实际意义，本文就带大家来看看这道题。

至于 Twitter 的什么功能跟算法有关系，等我们描述一下题目要求就知道了。

一、题目及应用场景简介

Twitter 和微博功能差不多，我们主要要实现这样几个 API：

```
class Twitter {

    /** user 发表一条 tweet 动态 */
    public void postTweet(int userId, int tweetId) {}

    /** 返回该 user 关注的人（包括他自己）最近的动态 id，
    最多 10 条，而且这些动态必须按从新到旧的时间线顺序排列。*/
    public List<Integer> getNewsFeed(int userId) {}

    /** follower 关注 followee，如果 Id 不存在则新建 */
    public void follow(int followerId, int followeeId) {}

    /** follower 取关 followee，如果 Id 不存在则什么都不做 */
    public void unfollow(int followerId, int followeeId) {}
}
```

举个具体的例子，方便大家理解 API 的具体用法：

```
Twitter twitter = new Twitter();
```

```
twitter.postTweet(1, 5);
// 用户 1 发送了一条新推文 5

twitter.getNewsFeed(1);
// return [5], 因为自己是关注自己的

twitter.follow(1, 2);
// 用户 1 关注了用户 2

twitter.postTweet(2, 6);
// 用户2发送了一个新推文 (id = 6)

twitter.getNewsFeed(1);
// return [6, 5]
// 解释：用户 1 关注了自己和用户 2，所以返回他们的最近推文
// 而且 6 必须在 5 之前，因为 6 是最近发送的

twitter.unfollow(1, 2);
// 用户 1 取消关注了用户 2

twitter.getNewsFeed(1);
// return [5]
```

这个场景在我们的现实生活中非常常见。拿朋友圈举例，比如我刚加到女神的微信，然后我去刷新一下我的朋友圈动态，那么女神的动态就会出现在我的动态列表，而且会和其他动态按时间排好序。只不过 Twitter 是单向关注，微信好友相当于双向关注。除非，被屏蔽...

这几个 API 中大部分都很好实现，最核心的功能难点应该是 `getNewsFeed`，因为返回的结果必须在时间上有序，但问题是用户的关注是动态变化的，怎么办？

这里就涉及到算法了：如果我们把每个用户各自的推文存储在链表里，每个链表节点存储文章 id 和一个时间戳 time（记录发帖时间以便比较），而且这个链表是按 time 有序的，那么如果某个用户关注了 k 个用户，我们就可以用合并 k 个有序链表的算法合并出有序的推文列表，正确地 `getNewsFeed` 了！

具体的算法等会讲解。不过，就算我们掌握了算法，应该如何编程表示用户 user 和推文动态 tweet 才能把算法流畅地用出来呢？**这就涉及简单的面向对象设计了**，下面我们来由浅入深，一步一步进行设计。

二、面向对象设计

根据刚才的分析，我们需要一个 User 类，储存 user 信息，还需要一个 Tweet 类，储存推文信息，并且要作为链表的节点。所以我们先搭建一下整体的框架：

```
class Twitter {
    private static int timestamp = 0;
    private static class Tweet {}
    private static class User {}

    /* 还有那几个 API 方法 */
    public void postTweet(int userId, int tweetId) {}
    public List<Integer> getNewsFeed(int userId) {}
    public void follow(int followerId, int followeeId) {}
    public void unfollow(int followerId, int followeeId) {}
}
```

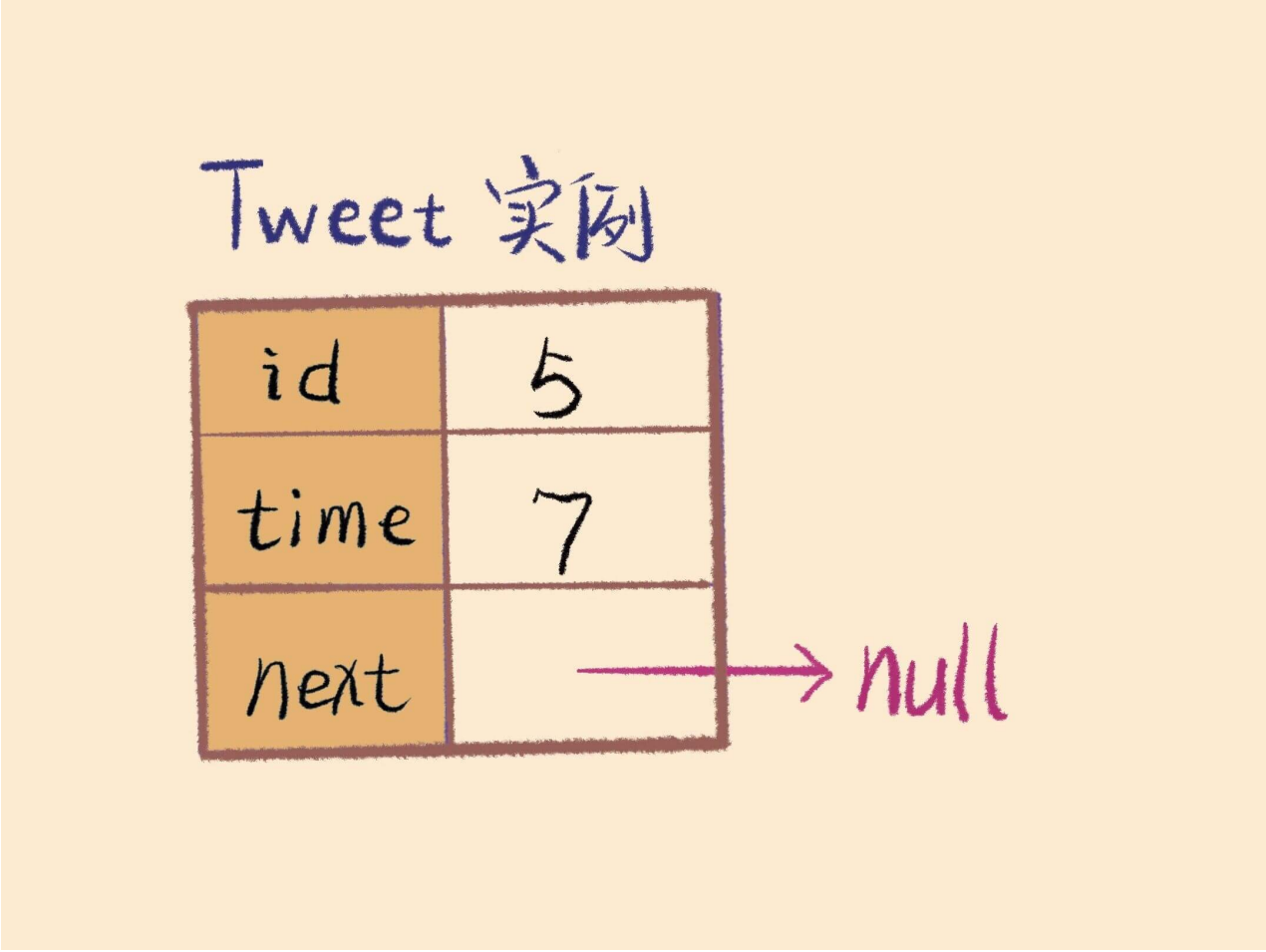
之所以要把 Tweet 和 User 类放到 Twitter 类里面，是因为 Tweet 类必须要用到一个全局时间戳 timestamp，而 User 类又需要用到 Tweet 类记录用户发送的推文，所以它们都作为内部类。不过为了清晰和简洁，下文会把每个内部类和 API 方法单独拿出来实现。

1、Tweet 类的实现

根据前面的分析，Tweet 类很容易实现：每个 Tweet 实例需要记录自己的 tweetId 和发表时间 time，而且作为链表节点，要有一个指向下一个节点的 next 指针。

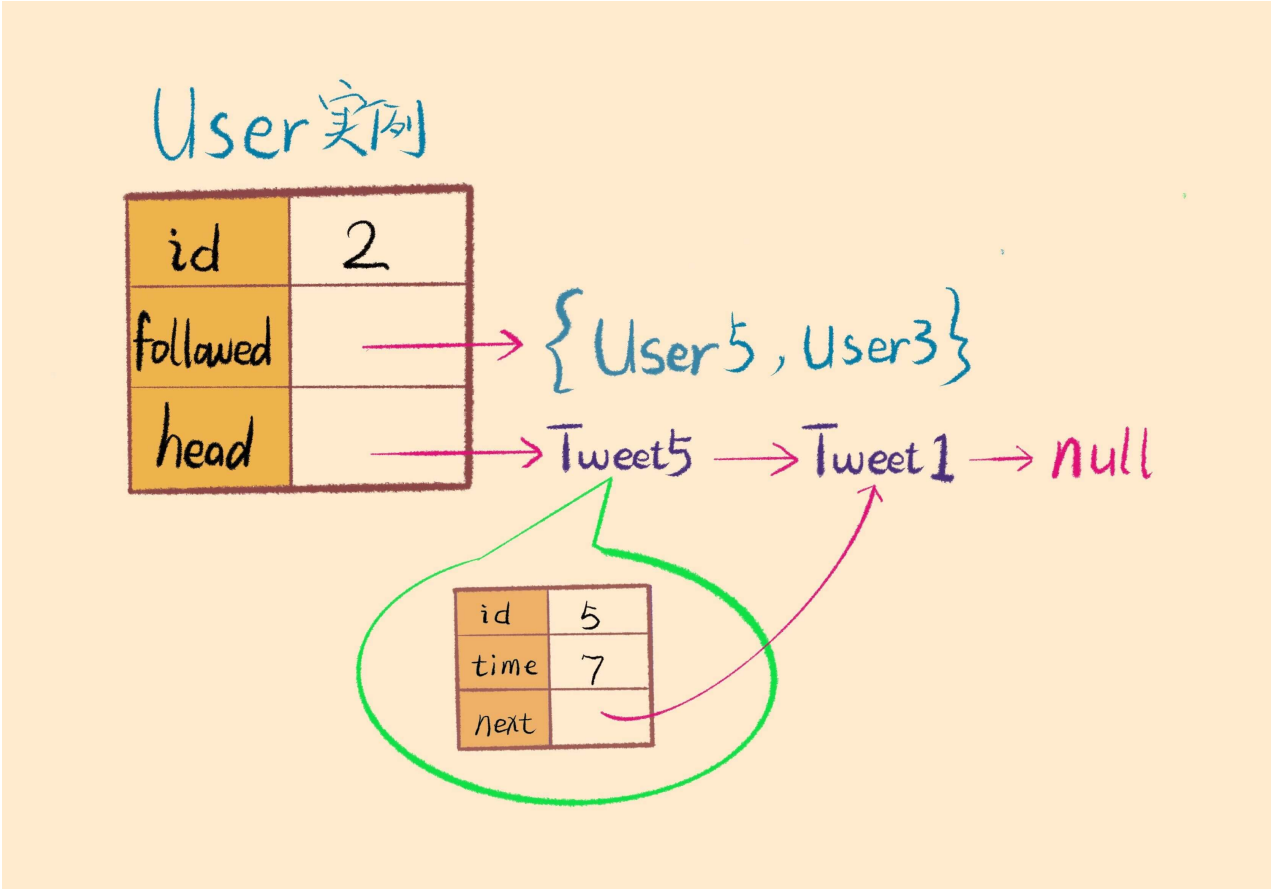
```
class Tweet {
    private int id;
    private int time;
    private Tweet next;

    // 需要传入推文内容（id）和发文时间
    public Tweet(int id, int time) {
        this.id = id;
        this.time = time;
        this.next = null;
    }
}
```



2、User 类的实现

我们根据实际场景想一想，一个用户需要存储的信息有 userId，关注列表，以及该用户发过的推文列表。其中关注列表应该用集合（Hash Set）这种数据结构来存，因为不能重复，而且需要快速查找；推文列表应该由链表这种数据结构储存，以便于进行有序合并的操作。画个图理解一下：



除此之外，根据面向对象的设计原则，「关注」「取关」和「发文」应该是 User 的行为，况且关注列表和推文列表也存储在 User 类中，所以我们也应该给 User 添加 follow，unfollow 和 post 这几个方法：

```
// static int timestamp = 0
class User {
    private int id;
    public Set<Integer> followed;
    // 用户发表的推文链表头结点
    public Tweet head;

    public User(int userId) {
        followed = new HashSet<>();
        this.id = userId;
        this.head = null;
        // 关注一下自己
        follow(id);
    }

    public void follow(int userId) {
        followed.add(userId);
    }

    public void unfollow(int userId) {
        // 不可以取关自己
        if (userId != this.id)
```

```
            followed.remove(userId);
    }

    public void post(int tweetId) {
        Tweet twt = new Tweet(tweetId, timestamp);
        timestamp++;
        // 将新建的推文插入链表头
        // 越靠前的推文 time 值越大
        twt.next = head;
        head = twt;
    }
}
```

3、几个 API 方法的实现

```
class Twitter {
    private static int timestamp = 0;
    private static class Tweet {...}
    private static class User {...}

    // 我们需要一个映射将 userId 和 User 对象对应起来
    private HashMap<Integer, User> userMap = new HashMap<>();

    /** user 发表一条 tweet 动态 */
    public void postTweet(int userId, int tweetId) {
        // 若 userId 不存在，则新建
        if (!userMap.containsKey(userId))
            userMap.put(userId, new User(userId));
        User u = userMap.get(userId);
        u.post(tweetId);
    }

    /** follower 关注 followee */
    public void follow(int followerId, int followeeId) {
        // 若 follower 不存在，则新建
    if(!userMap.containsKey(followerId)){
      User u = new User(followerId);
      userMap.put(followerId, u);
    }
        // 若 followee 不存在，则新建
    if(!userMap.containsKey(followeeId)){
      User u = new User(followeeId);
      userMap.put(followeeId, u);
    }
    userMap.get(followerId).follow(followeeId);
    }

    /** follower 取关 followee，如果 Id 不存在则什么都不做 */
```

```
    public void unfollow(int followerId, int followeeId) {
        if (userMap.containsKey(followerId)) {
            User flwer = userMap.get(followerId);
            flwer.unfollow(followeeId);
        }
    }

    /** 返回该 user 关注的人（包括他自己）最近的动态 id,
    最多 10 条，而且这些动态必须按从新到旧的时间线顺序排列。*/
    public List<Integer> getNewsFeed(int userId) {
        // 需要理解算法，见下文
    }
}
```

三、算法设计

实现合并 k 个有序链表的算法需要用到优先级队列（Priority Queue），这种数据结构是「二叉堆」最重要的应用，你可以理解为它可以对插入的元素自动排序。乱序的元素插入其中就被放到了正确的位置，可以按照从小到大（或从大到小）有序地取出元素。

```
PriorityQueue pq
# 乱序插入
for i in {2,4,1,9,6}:
    pq.add(i)
while pq not empty:
    # 每次取出第一个（最小）元素
    print(pq.pop())

# 输出有序：1,2,4,6,9
```

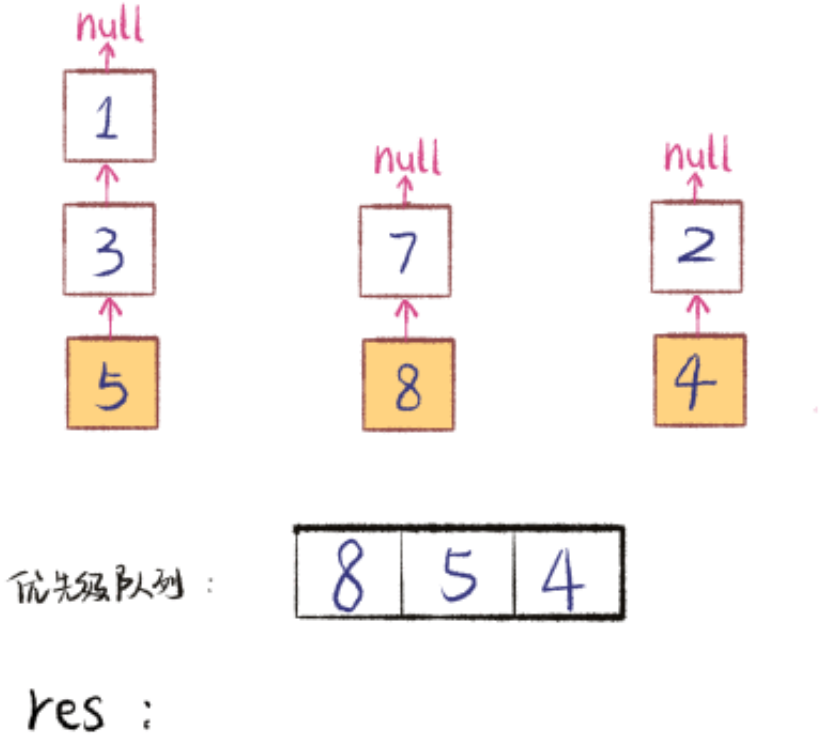
借助这种牛逼的数据结构支持，我们就很容易实现这个核心功能了。注意我们把优先级队列设为按 time 属性**从大到小降序排列**，因为 time 越大意味着时间越近，应该排在前面：

```
public List<Integer> getNewsFeed(int userId) {
    List<Integer> res = new ArrayList<>();
    if (!userMap.containsKey(userId)) return res;
    // 关注列表的用户 Id
    Set<Integer> users = userMap.get(userId).followed;
    // 自动通过 time 属性从大到小排序，容量为 users 的大小
    PriorityQueue<Tweet> pq =
        new PriorityQueue<>(users.size(), (a, b)->(b.time - a.time));

    // 先将所有链表头节点插入优先级队列
    for (int id : users) {
        Tweet twt = userMap.get(id).head;
        if (twt == null) continue;
        pq.add(twt);
    }
```

```
    while (!pq.isEmpty()) {
        // 最多返回 10 条就够了
        if (res.size() == 10) break;
        // 弹出 time 值最大的（最近发表的）
        Tweet twt = pq.poll();
        res.add(twt.id);
        // 将下一篇 Tweet 插入进行排序
        if (twt.next != null)
            pq.add(twt.next);
    }
    return res;
}
```

这个过程是这样的，下面是我制作的一个 GIF 图描述合并链表的过程。假设有三个 Tweet 链表按 time 属性降序排列，我们把他们降序合并添加到 res 中。注意图中链表节点中的数字是 time 属性，不是 id 属性：



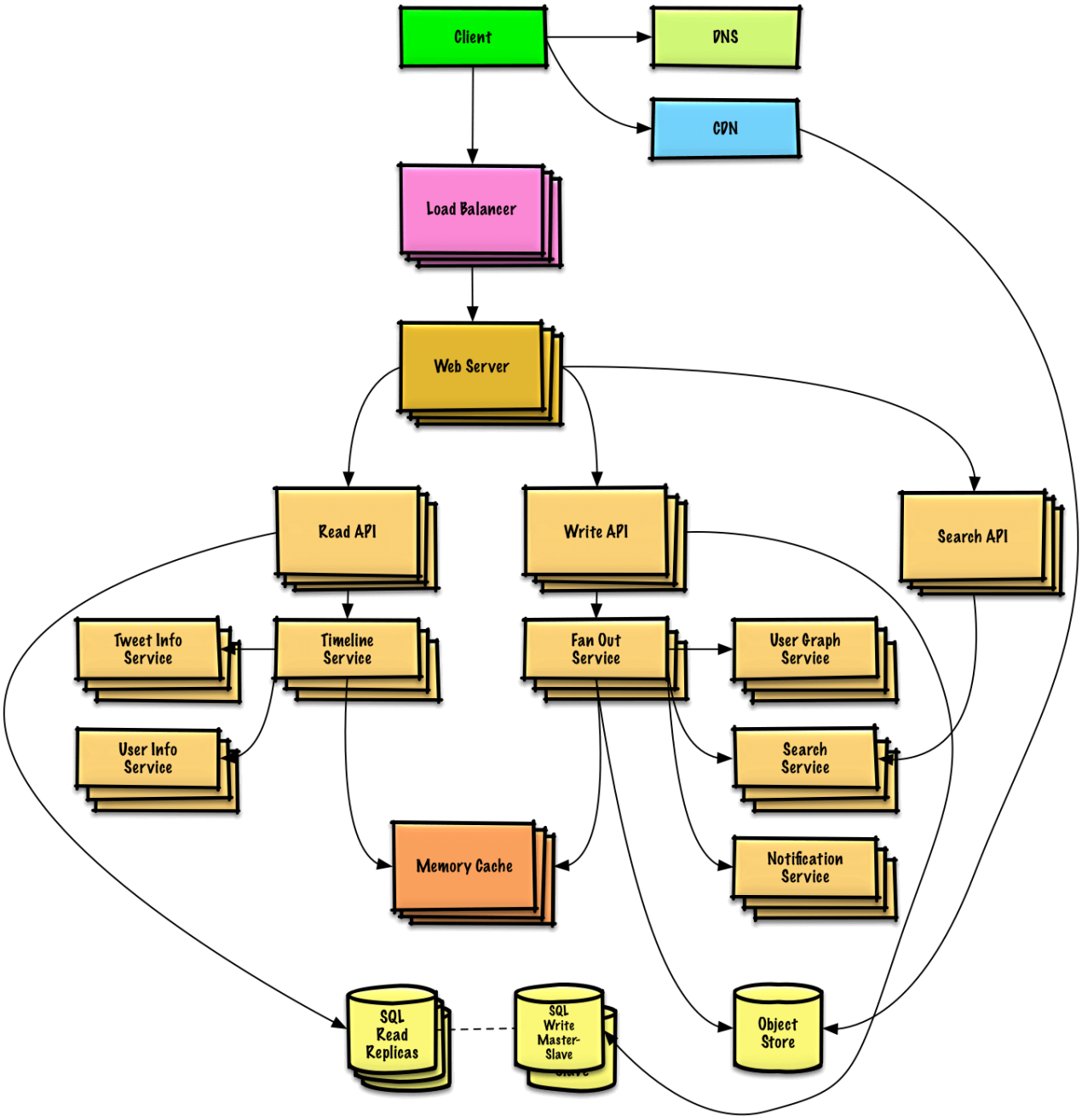
至此，这道一个极其简化的 Twitter 时间线功能就设计完毕了。

四、最后总结

本文运用简单的面向对象技巧和合并 k 个有序链表的算法设计了一套简化的时间线功能，这个功能其实广泛地运用在许多社交应用中。

我们先合理地设计出 User 和 Tweet 两个类，然后基于这个设计之上运用算法解决了最重要的一个功能。可见实际应用中的算法并不是孤立存在的，需要和其他知识混合运用，才能发挥实际价值。

当然，实际应用中的社交 App 数据量是巨大的，考虑到数据库的读写性能，我们的设计可能承受不住流量压力，还是有些太简化了。而且实际的应用都是一个极其庞大的工程，比如下图，是 Twitter 这样的社交网站大致的系统结构：



我们解决的问题应该只能算 Timeline Service 模块的一小部分，功能越多，系统的复杂性可能是指数级增长的。所以说合理的顶层设计十分重要，其作用是远超某一个算法的。

最后，Github 上有一个优秀的开源项目，专门收集了很多大型系统设计的案例和解析，而且有中文版本，上面这个图也出自该项目。对系统设计感兴趣的读者可以点击 这里 查看。

PS：本文前两张图片和 GIF 是我第一次尝试用平板的绘图软件制作的，花了很多时间，尤其是 GIF 图，需要一帧一帧制作。如果本文内容对你有帮助，点个赞分个享，鼓励一下我呗！

==其他语言代码==