**关于数据库‘状态’字段设计的思考与实践**

最近在做订单及支付相关的系统，在订单表的设计阶段，团队成员就‘订单状态’数据库字段设计有了一些分歧，网上也有不少关于这方面的思考和探讨，结合这些资料和项目的实际情况，拟对一些共性问题进行更深一层的思考，笔耕在此，和大家一起探讨。

# 问题综述

这里的分歧点即有团队内部的分歧点，也有网络上常见的一些分歧点，先将存在的分歧点抛出来：

1、订单表的‘订单状态’字段对应的字典值应当包含哪些状态值？对于‘已评论’、‘已退货’、’已退款’这类状态是放到‘订单状态’中？还是独立一个字段标识？

2、订单表的‘订单状态’字段对应的字典值如何表示？可选项有：使用数字标识、使用多‘位’存储方式标识、使用具有明确业务含义的英文字符串标识；

3、订单表的‘订单状态’字段使用何种类型？可选项有：number(N)、char(N)、varchar2(N);

如果嫌分析过程过于啰嗦，可以直接拉到最后看结论。

# 业务分析

我们先不去看问题，先来看看和‘订单（Order）’实体相关的业务是怎样的。下面我们会针对可能改变订单实体状态的行为已经状态变化的可能性进行详细的分析。

订单业务实体相关的业务流程如下：下单（create）--> 买家付款（pay）--> 卖家发货（deliver）-->买家收货（receive）-->退货（rereturn）；此外，还有退款（refund）和评论（comment），这两个行为比较特殊，其前向行为可能存在多个。

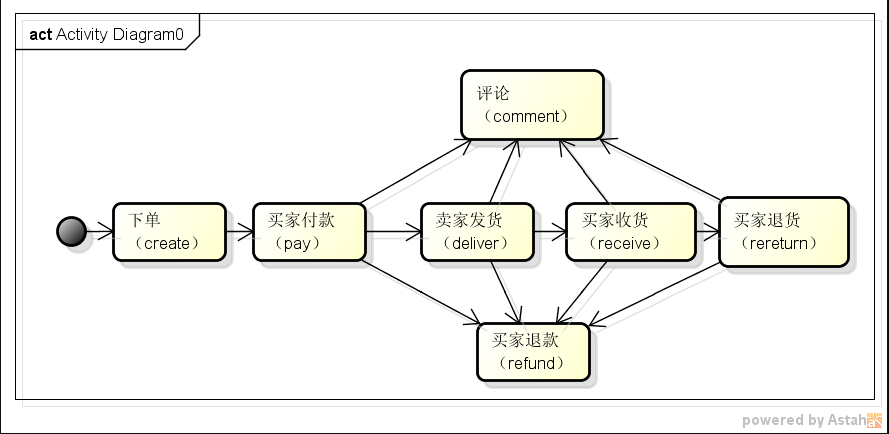


Figure0：订单业务实体相关业务流程

首先，可以改变订单业务状态【这里的状态不是指‘订单状态’（OrderState）这个数据库字段，而是指实际业务状态，我们简记为（BizState），以和OrderState区分开】的行为有哪些？按照典型电商的业务流程，主要的行为（action）有：下单、付款、发货、收货、退款/退货、评论；每一种行为的发生，都会导致订单的业务状态BizState发生变化，比如‘下单’行为会创建订单，‘付款’行为会使订单变为‘已付款’，‘发货’行为可以使订单状态变为‘已发货’，‘收货’行为会使订单状态变为‘已收货’，‘评论’行为会使订单状态变为‘已评论’。‘退款/退货’action不是所有订单都支持的，为减小复杂度，暂不考虑它们。

其次，细分下每种action对BizState带来的影响，会发现还可以细分为四种子状态（subState）：action未开始（标记为0）、action进行中（标记为1）、action成功（标记为2）、action失败（标记为3）；理论上，将所有action的所有subState进行排列得到4\*4\*4\*4\*4=1024（暂未考虑‘退货’）；实际上，很多组合是没有业务意义的，是不可能存在的，比如‘未开始已付款...’（\*\*\*20）这一类组合是不可能发生的，应当舍弃。用表格将上述的组合分析如下：

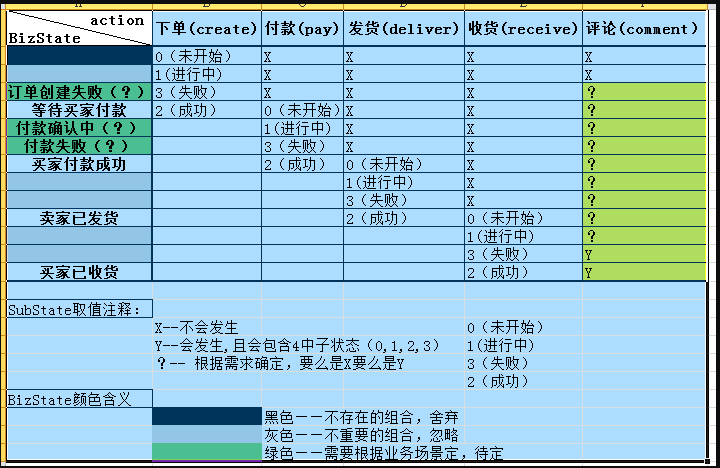


Table1、多种action作用下的BizState组合情况

通过上表，我们可以发现些的规律：

**‘下单’、‘付款’、‘发货’、‘收货’前四种action是存在依赖关系的，亦即后一个action依赖于前一个action的完成；**所以，他们的SubState组合情况就会非常少；

**‘评论comment’这个action的SubState和其他状态组合会有很多种可能性；**除了前面了两行是‘X’，后面是‘?’或者‘Y’，‘？’是指需求上是否允许在对应的BizState上进行评论，如果允许，则每种BizState需要多出4种可能，这样组合的可能性就会变得很大。

**没有业务意义的SubState组合被舍弃**。表中的标黑单元格，表示这个BizState是毫无意义的，因为‘未下单’的订单对于我们来讲是不存在的，这类组合需要舍弃；同样的，还有很多其他的组合也是不存在的，被舍弃掉，未展示在上表中，如‘已下单已付款未发货已收货’这种。

**通常某个action的SubState为‘1进行中’、‘3失败’时，会被忽略，但也有例外；**比如‘付款’action的‘3失败’状态，和‘付款’action的‘1进行中’状态，具体分析见后面内容。

**忽略所有action的‘0未开始’SubState状态**。因为这类SubState对于BizState不会带来变化。

综合下来，我们得到上表的BizState，注意这里的Comment action未进行细化处理，如果细化处理，会发现BizState的可能性会增大很多很多。

接下来我们就之前提出的这些问题进行逐个讨论。

# 问题一、订单表的‘订单状态’字段应当包含哪些状态值？

什么样的‘订单业务状态’（BizState）需要记录到系统层面的‘订单状态’（OrderState）字段呢？如果记录多了，则系统处理的复杂度会增大；记录少了，那么‘订单状态’（OrderState）字段就不能完整的表示出订单实体状态变化情况。

## 核心状态

通过上面的业务分析可知：大部分存在依赖关系的action（create、pay、deliver、receive），他们产生的合理的SubState组合是非常少的，而且他们之间的依赖是单向依赖，状态机的处理也很简单，因此，我们先将这部分BizState纳入到OrderState中：

等待买家付款

买家付款成功

卖家已发货

买家已收货

目前的订单状态流转：

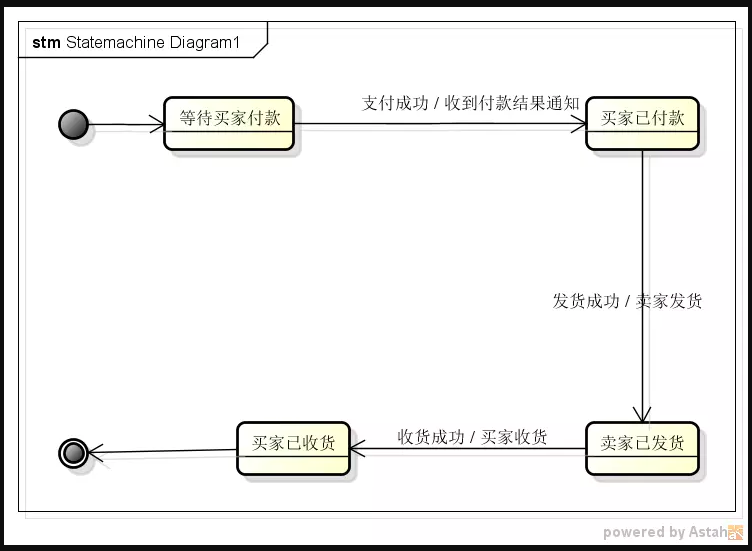


Figure1：订单状态流转图--base

## ‘action行为’失败的情况

对于action的SubState是‘3失败’的处理，需要针对不同的action进行分析。类似‘下单Create’这样的action，如果失败，则可以直接将OrderState置为‘订单创建失败’，因为Create action是第一个action，它的失败意味着Order实体出生即死，BizState置为终态，对于这个BizState应当纳入到OrderState中记录，不过这个OrderState其实对于用户并无多大用处，因为用户并不会关心下单失败的订单，他更关心的是重新下单；

对于‘支付’失败，则要看需求，如果需求要求用户可以继续支付，则订单需要保留，并且状态仍然为‘等待买家付款’，如果不允许再支付，则理论上可以将BizState置为‘支付失败’终态，所以，‘支付失败’的BizState终态也应当记录到OrderState字段中。

对于‘发货’失败、‘收货’失败的情况，通常是不会发生的，即使发生也不属于系统能够控制的范畴，系统记录并无意义，更具建设性的做法是通过线下手段尽快解决问题，重新发货等等，所以对于这些状态系统的OrderState字段不予记录。

这样下来我们的OrderState字典值增加到6个，加粗项为新增：

**创建订单失败（终态）**

等待买家付款

**买家付款失败（终态，依赖需求而定）**

买家付款成功

卖家已发货

买家已收货

目前的订单状态流转：

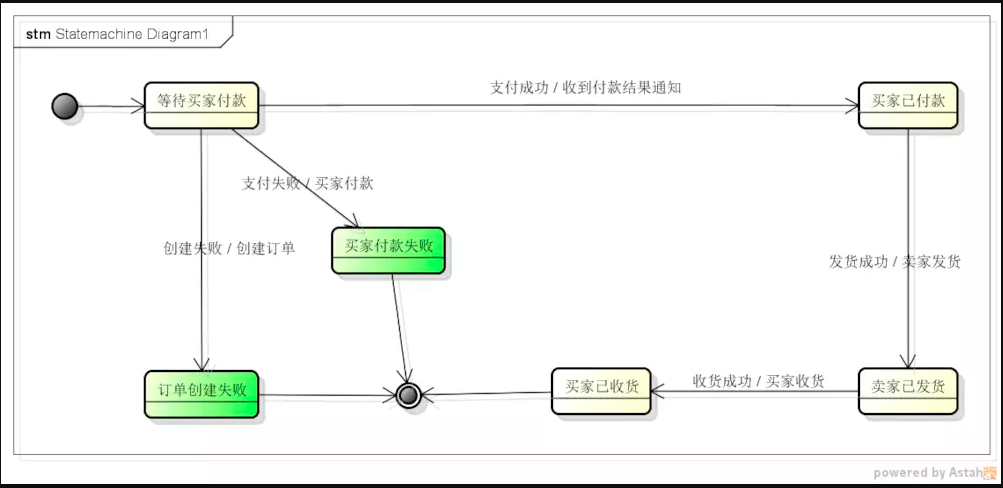


Figure2：订单状态流转图--add '失败'subState‘

## ‘action行为’进行中的情况

对于action的SubState是‘1进行中’的处理，同样需要具体场景具体分析。‘付款’行为是用户发起的，但是并不是和订单系统之间的交互，涉及到支付系统的处理，这个领域也不是订单系统可控的，但关系到钱，用户比较关系，所以对于这样一个中间态，我们需要记录，以便用户通过订单系统查询订单状态，为便于用户理解，将此状态在OrderState中记为‘付款确认中’；‘发货’‘收货’进行中的情况，不是订单系统可以控制的领域，我们可以把他们当着行为‘未开始’处理，比如‘发货进行中’，订单系统的OrderState值为‘买家已付款’，但给用户看到的提示信息是‘买家已付款，等待卖家发货’，实际上这时候卖家可能正在发货中，但是用户不会去关心到底有没有打包好货物什么的，所以这类‘进行中’状态可以舍弃。这样下来订单系统的OrderState字段又多了一个字典值：‘付款确认中’：

创建订单失败（终态）

等待买家付款

**付款确认中**

买家付款失败（终态，依赖需求而定）

买家付款成功

卖家已发货

买家已收货

目前的订单状态流转：

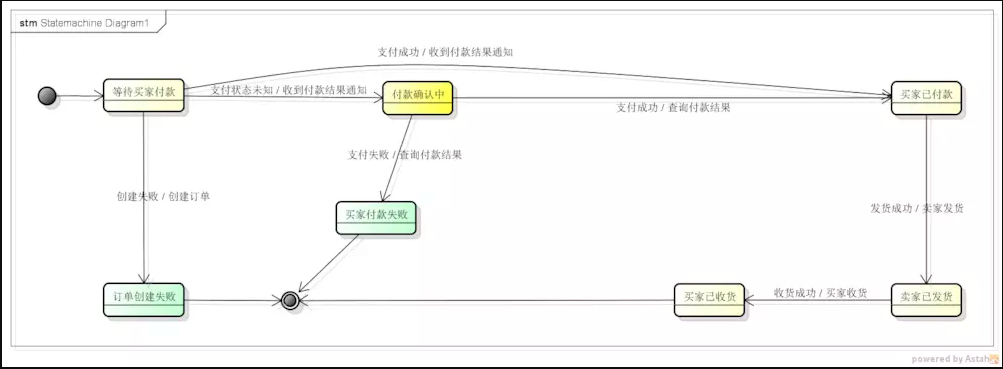


Figure3：订单状态流转图--add '进行中'subState‘

## ‘action行为’未开始的情况

忽略所有action的‘0未开始’SubState状态。因为这类SubState对于BizState不会带来变化。

## ‘评论comment’的处理

最后，再来看看‘评论comment’这个action。如果需求上要求：只有买家收货后才能发起‘评论’操作，则可以任务‘评论comment’单向依赖于‘receive收货’行为，那么可以将这个action的subState对应的少量BizState（应当只有‘买家已评论’、‘卖家已评论’状态）纳入OrderState字段统一记录；但是如果需求是：买家在下单后就可以开始评论，比如如果卖家发货慢了，买家可以上去吐槽，那么‘评论comment’就不是单向依赖于‘receive收货’行为了，而是多向依赖于‘pay付款’、‘deliver发货’、‘receive收货’，那么这些actions的subState组合可能性就暴增，BizState的字典取值也会暴增，显然，不应当将这么多的BizState交给OrderState来记录，而应当由一个独立的数据库字段负责记录‘评论comment’的SubState，我们可以将这个字段取名为‘CommentState’（评论状态），它的字典值不多，只有：‘未评论’、‘买家已评论’、‘卖家已评论’；其实，对于前一种需求，也可以不讲‘评论comment’对应的SubState产生的BizState纳入OrderState，因为用户对于评论与否其实并不是那么关心的，也就是说‘评论comment’并不是核心业务流程，为了降低核心业务流程的系统处理复杂度，将其从核心业务流程中剥离出来较好。

综上，我们应当将‘评论comment’对应的BizState独立到一个字段中记录。

## ‘退货rereturn’的处理

再来看看‘退货rereturn’行为对应的BizState的处理。‘退货rereturn’并不是所有订单都会经历的，但是一旦涉及，则‘退货rereturn’在业务流程上必定是单向依赖于单向依赖于‘receive收货’，所以应当将‘退货rereturn’产生的BizState（‘退货中’、‘退货成功’，‘退款失败’和‘未退货’被忽略，见上面解释）纳入OrderState一并记录；这样我们的OrderState有多了两种字典值，这里我们不考虑一个订单中有多种商品的情况，故把‘退货成功'当着终态处理，如果是一个订单多种货物的情况，需要重新仔细分析。加粗项为新增：

创建订单失败（终态）

等待买家付款

付款确认中

买家付款失败（终态，依赖需求而定）

买家付款成功

卖家已发货

买家已收货

**退货中**

**退货成功（终态）**

目前的订单状态流转：

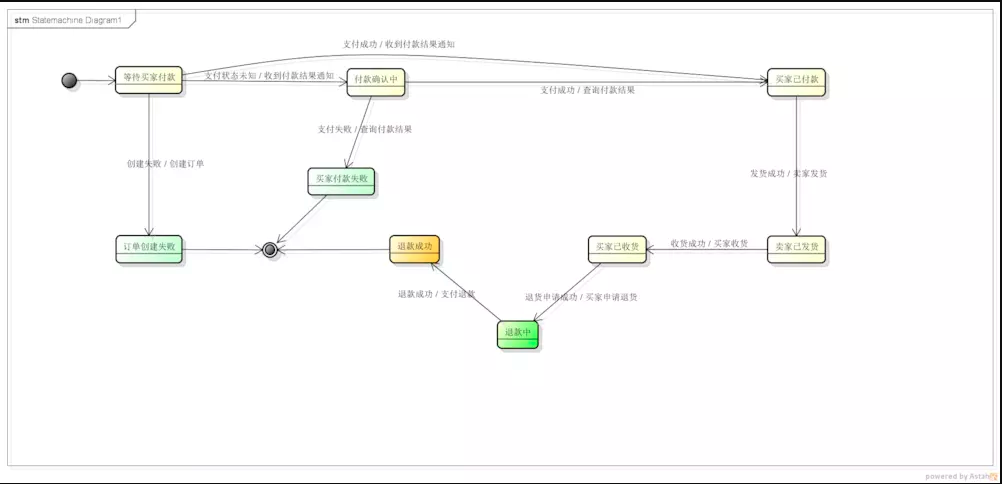


Figure4：订单状态流转图--add rereturn subState

## ‘退款refund’的处理

最后来看下‘退款refund’行为对应的BizState的处理。首先，我们需要知道‘退货’和‘退款’是两种不同的业务行为，他们的关系是：通常意义上，‘退货’必然导致‘退款’，但是‘退款’可以没有‘退货’的参与（这里不讨论特殊情况，比如对于虚拟货物来讲，付款成功通常以为着收货成功，这时候就只能是在由‘退货’导致‘退款’），比如电商允许用户付款成功后收到货物前发起‘退款’。也就是说‘退款refund’并不单向依赖于‘退货rereturn’，和‘评论comment’一样是多项依赖，所以，我们可以参考‘评论comment’的处理方式，单独建立一个字段‘RefundState退款状态’记录‘退款refund’产生的BizState，这个状态字段的字典值有：退款中，退款成功。

## 其他情况考虑

另外，可能还有一些增强型需求，让客户体验更好，比如用户可以创建订单之后付款之前，将订单取消，或者由系统跑批将用户长时间未支付的订单关闭，这会产生一种新的action——‘close关闭’，对应的会产生一种新的有意义的BizState——‘订单关闭/取消’，这个不属于核心流程中的，且并无纠结之处，不予详细讨论，罗列如下：

创建订单失败（终态）

等待买家付款

付款确认中

买家付款失败（终态，依赖需求而定）

买家付款成功

卖家已发货

买家已收货

退货中

退货成功（终态）

**订单关闭（终态）**

## 结论

综上，我们可以得出放入数据库’订单状态‘字段的标准：核心业务流程，向前单向依赖。扩展到其他业务实体是一样的，这里说的’订单状态‘字段实际是指该业务实体对应的数据表的主业务状态字段。我们把结论扩展一下：

**如果某个action属于业务实体对应的核心业务流程，且该action单向依赖于其前向的action，则需要将这个action产生的BizState放入到业务实体对应的数据库表的主状态字段中记录。**

OrderState字段记录的BizState业务状态有10种，其中4种是终态，其余状态为中间态。这些状态的流转关系为：

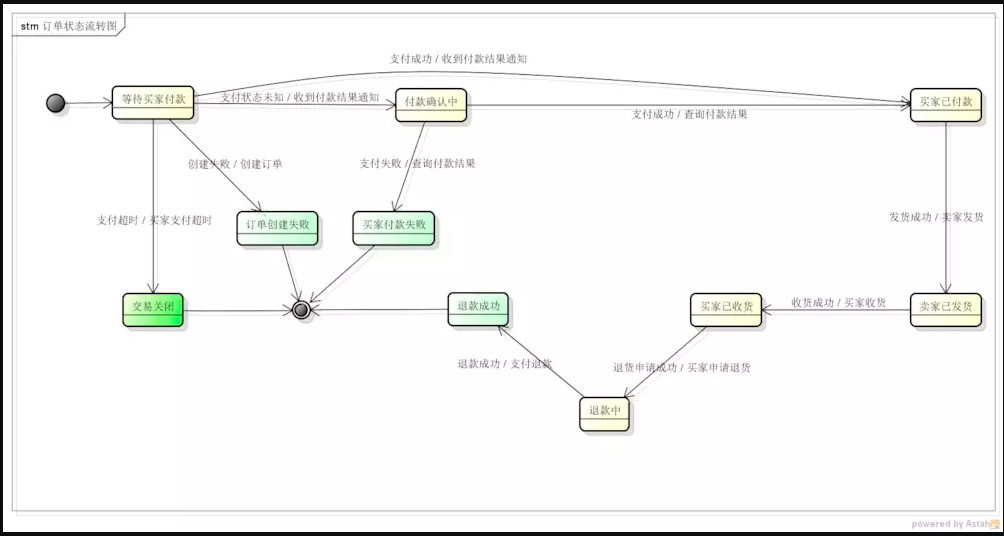


Figure5：订单状态流转图--final

# 问题二、订单表的‘订单状态’字段的字典值的表示形式？

先列出可选项：使用数字标识、使用多‘位’存储方式标识、使用具有明确业务含义的英文字符串标识；对可选项做逐一解释：

a、**使用数字标识**——使用一个数字标识一种状态，并未要求是sequence的；如‘等待买家付款’表示为‘0’;

b、**使用多‘位’存储方式标识**——将某种行为是否发生对应的状态对应到一个位上，比如‘是否付款’定义在第一位，‘是否发货’定义在第二位，‘是否收货’定义在第三位，‘是否评论’定义在第四位，则状态‘卖家已收货未评论’可以表示为：0111；而‘等待买家付款’则表示为‘0000’；当然这里的‘位’可能是二进制的也可能是N进制，后面我们详细讨论。

c、**使用具有明确业务含义的英文字符串标识**——该方案和方案a类似，不过字典值变为具有明确业务含义的英文支付串，如‘等待买家付款’表示为‘WAIT\_BUYER\_PAY’;

方案a是数据库字段字典的惯用方式，简单直观，但是有一个坏处在于：当字典值较多时，数据库表的使用者记不住字典的含义，需要反复查找资料确认；有人会说将字典值写到字段的注释里，这个在实践中不是很靠谱，通常表建立后，如果字段增加了字典值，通常开发人员都会忽略更改字典值；而且在使用工具（如pl/sql）查询数据库时，并不会将所有字典值展示出来；

通过问题一的分析，可知：方案b使用多‘位’存储方式会增加复杂度，并没有必要，可以通过将‘是否评论’状态独立成一个字段进行表示。

方案c和方案a类似，好处在于通过字典值直接知道业务含义，坏处在于会给编码和手工查询时带来复杂度，通常人们也记不住‘等待买家付款’的英文字典是‘WAIT\_BUYER\_PAY’，那么手动写sql查询‘等待买家付款’时就犯迷糊了。

折中之后，我们组合方案a和方案c，得到方案d：另外建立一张字典表，存储：数字形式的字典值、字典英文名称、字典中文简称、字典解释；订单实体表的OrderState字段使用数字作为字典值。

对于方案d，看到OrderState的数字形式状态时，可以先看看字段注释是否有此字典的定义，如果没有就取查下字典表，得到字典值和含义；在编码和手动sql查询时也会变得比较容易，数字的位数毕竟要少些；建立字典表的其他好处还有：字典的解释可以写的很详细，在报表中要求展示字典中文名时，也能直接从数据库联表查询得到，而不必额外做一次映射。（有参考：[数据库表设计（状态字段）](https://link.jianshu.com?t=http:/www.itpub.net/thread-1447193-1-1.html" \t "_blank)）

那么对于字典数量很少的状态字段是否有必要额外新建一张字典表呢？这个根据实际情况考虑，通常可以先不建，如果后续有业务场景需要再行创建也不迟。

而对于非业务实体表的系统日志/跑批记录表等的状态，则完全可以使用数字形式的字典，因为通常不会有业务场景使用到这些字典值，而且这些字典值域应当会比较小，所以没有必要为他们创建单独的字典表。

综上得出结论：

1、字典值域较多、变化较多、报表等业务场景会使用到的业务实体表的业务状态字段，使用‘方案d：新建字典表’的方案处理；如‘订单业务实体表’中的‘订单状态’字段。

2、字典值域较少、变化较少、报表等业务场景不会使用到的业务实体表的业务状态字段，使用‘方案a：使用数字标识字典’的方案处理；如‘支付宝的支付流水表’的‘支付流水状态’字段。

3、系统日志/跑批记录表的状态字段，使用‘方案a：使用数字标识字典’的方案处理；如‘待收货记录表’的‘跑批状态’字段。

# 问题三、数据库表的‘状态’字段使用何种类型

列出可选项：number(N)、char(N)、varchar2(N)，其中N是一个长度值。

这个问题主要需要考虑使用场景、扩展性、性能、存储。

‘状态’字段主要使用在查询场景，且通常是‘=’或者‘in’的查询，并没有区间类的查询，故三者差别不大；

对于性能，参考[[原创]在Oracle 10g，Number、Char和Varchar2类型作为主键，查询效率分析](https://link.jianshu.com?t=http:/blog.csdn.net/tan_jianhui/article/details/8571342) char(N)、varchar2(N)性能优于number(N)，故舍弃number(N)。

考虑到扩展性，char(N)、varchar2(N)差不多；

考虑到存储，varchar2更加占用空间更小，故选择varchar2(N)。

**综上：选择varchar2(N)作为数据库‘状态’字段的类型。**

# 问题结论汇总

1、订单表的‘订单状态’字段对应的字典值应当包含哪些状态值？对于‘已评论’、‘已退货’这类状态是放到‘订单状态’中？还是独立一个字段标识？

**如果某个action（行为，如支付）属于业务实体对应的核心业务流程，且该action单向依赖于其前向的action，则需要将这个action产生的业务状态放入到业务实体对应的数据库表的主状态字段中记录。**

问题中的‘已评论’由‘评论’行为产生，而‘评论’这个action并不是订单业务实体的核心业务流程，且可能存在多个前向依赖action（支付、发货、收货等），所以应当独立到一个字段标识。

问题中的‘已退货’由‘退货’行为产生，而‘退货’这个action是订单业务实体的核心业务流程，用户非常关心，且只单向依赖于‘收货’action，所以应当记录到订单业务实体表的‘订单状态’字段中。

问题中的‘已退款’由‘退款’行为产生，而‘退款’这个action是订单业务实体的核心业务流程，用户非常关心，但是这个action存在多个前向依赖action（支付、发货、收货等），所以应当独立到一个字段标识。

2、订单表的‘订单状态’字段对应的字典值如何表示？可选项有：使用数字标识、使用多‘位’存储方式标识、使用具有明确业务含义的英文字符串标识；

1、字典值域较多、变化较多、报表等业务场景会使用到的业务实体表的业务状态字段，使用‘方案d：新建字典表’的方案处理；如‘订单业务实体表’中的‘订单状态’字段。

2、字典值域较少、变化较少、报表等业务场景不会使用到的业务实体表的业务状态字段，使用‘方案a：使用数字标识字典’的方案处理；如‘支付宝的支付流水表’的‘支付流水状态’字段。

3、系统日志/跑批记录表的状态字段，使用‘方案a：使用数字标识字典’的方案处理；如‘待收货记录表’的‘跑批状态’字段。

3、订单表的‘订单状态’字段使用何种类型？可选项有：number(N)、char(N)、varchar2(N);

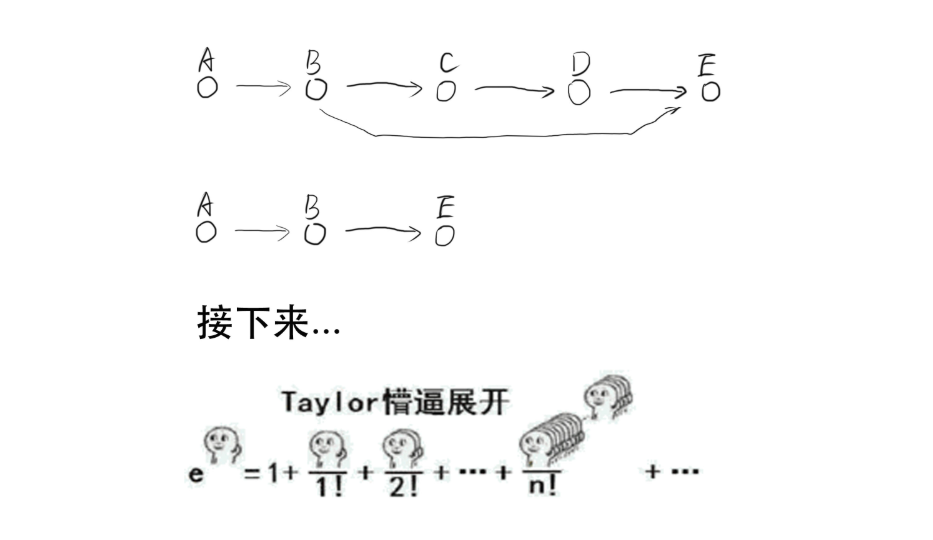
**varchar2(N)占用存储更少，且具有同等的性能、扩展性，选择varchar2(N)作为数据库‘状态’字段的类型。**

# 参考资料

[**数据库表设计（状态字段）**](https://link.jianshu.com?t=http:/www.itpub.net/thread-1447193-1-1.html)

[[原创]在Oracle 10g，Number、Char和Varchar2类型作为主键，查询效率分析](https://link.jianshu.com?t=http:/blog.csdn.net/tan_jianhui/article/details/8571342)

一个事物有5个流程，ABCDE一般的程序员通常会定义为 status 有5个状态 0/1/2/3/4 ，流程的顺序是 A->B->C->D->E ，可是有一天项目经理说这个流程会有个快捷流程 A->B->E 。于是原本的 0->1->2->3->4 就有部分数据变为了 0->1->4 ，这些都不会有太大的问题，然而，后来项目经理跟你说我需要区分在E状态下它是经历了 ABCDE 还是直接经历 ABE ，于是乎，你就懵逼了。



有另外一种设计方式倒是可以获取详细的经历的流程结点，那就是 A:1, B:2, C:4, D:8, E:16 。于是A->B->C->D->E 表示的状态码切换过程为 1->3->7->15->31 。A->B->E 表示的状态码切换过程为 1->3->20, 有人说，你这样当处于E状态时就有可能有两个状态码了31和20，我如何判定他是处于E状态呢，其实这是位运算上的一个小技巧，我们可以很简单的判定出他是否在E状态 status & 0x10 > 0 若计算结果为true，则表示有经历E状态，按照你的流程方向便可以判定是处于哪一未置

|  |
| --- |
| **switch (true) {**  **case status & 0x10:**  **return "E";**  **case status & 0x08:**  **return "D";**  **case status & 0x04:**  **return "C";**  **case status & 0x02:**  **return "B";**  **case status & 0x01:**  **return "A";**  **}** |

这一设计方式在ERP等系统的权限控制中更是常见

例如某一项目下有10个操作不同的用户可以配置不同的操作权限，那么就有 0b0000000000 到 0b1111111111 ，呃……，1024种权限配置方式，如果是100个【weightValue.size()】操作，那么就有 2^100 种权限配置方式。我们只需要给该用户一个 int(11) 类型的字段就可以完成权限配置。

又例如，某简历需要计算简历完整度，简历中分为10个【weightValue.size()】模块，不同的模块占分权值不同，那么我们也可以用一个字段来标记该简历的各模块的完成状况，0b0000000000 到 0b1111111111 ，你可以这样计算分值

|  |
| --- |
| **int value = 0;**  **int weightValueNumber = weightValue.size();**  **int zerokeep = math.pow(2, weightValueNumber);**  **status = status + zerokeep;**  **for (int i = 0; statusMove - zerokeep > 0; i++) {**  **statusMove = status < (i + 2); // 左移 (i+1)+1 位**  **int nakedValue = (statusMove > (i + 2)) ^ status - zerokeep;**  **boolean checked = nakedValue > 0;**  **if (checked) {**  **value += weightValue.get(i);**  **}**  **status = nakedValue + zerokeep;**  **}**  **return value;** |

这种算法的始祖应该算是 Linux 系统上的权限管理机制吧，chmod 777 /xxx 这种操作后面的智慧便是如此。若是铁定的不涉及到取某个结点ABCDE的独立状态详情，那么你可以随意定义状态码 1/2/3/4/5 或是 -2/-1/0/1/2 随你所好，但是假如某一天你遇上了坑爹的项目经理突然告诉你需要知道是否经历了CD状态，那么你在懵逼的时候是不是也会想起Linux系统上 chmod 指令的智慧呢？1234还是1248取决于你最初的设计。

今天想讲下自己对订单表状态字段的值设计的一个浅薄的思考和理解。

先说下我一年前对这个状态字段的一个值设计。

0：创建

-1：买家取消

-2：卖家取消

-3：系统取消

-4：失败

1：已支付

2：已发货

3：交易完成

4：已评价

这是一个很容易就设计出来的值。也是我之前的理解：负面状态为负数，正面状态为正数，初始化状态为0。

一开始用着，感觉还没什么问题。但时间久了，需求一变更，就发现这个值设计不对劲了。

假设有个需求，买卖双方可以对失败订单或者取消的订单进行评价。那状态值应该是多少？-4?4?

再假设另外一个需求，订单可以卖家先发货，买家再付款。那状态值怎么设计？

上面两个需求，在目前的状态值设计中，都已经无法满足需求，要么加状态值，要么加字段。

那，这个状态值应该怎么设计才可以呢？

思考了好久，然而还没在实践中使用过，如有不对，欢迎提出讨论。

由于订单的状态值，不是那种非1即2的单纯情况，是有可能出现多种状态共存的情况。所以，至少，按照我们的需求，可以把状态值设计为四位数。千分位表示订单状态，百分位表示金钱状态，十分位表示发货状态，个分位表示评论状态。看下面的值例子。

千分位

-4000：交易失败

-3000：系统取消

-2000：卖家取消

-1000：买家取消

0000：创建，初始化状态

1000：交易完成

百分位

100：已支付

十分位

10：已发货

个分位

1：买家已评论

2：卖家已评论

3：双方已评论

不知道上面有没有把我的想法写清楚。还是举例子来说明。

买家取消，但是买家已支付，未发货，未评论的订单状态值，为－1100。

交易完成，已支付，已发货，双方已评论的订单状态值，为1113。

已支付，已发货，未评论的订单状态值，为0110。

以上，就是根据我的一些浅薄的经验反思出来的经验。其实，不止在订单的状态值设计上。任何表设计中，涉及到类似状态字段这种值的设计，都可以做类似的设计。分解状态的多种类别，然后给每个类别分配一个位数。如果可以，还可以设计多几个冗余位数。

你说，如果每个状态类别的值超过9种情况怎么办？你忘了16进制这种东西。