- **UNDO Logs**
  - **only Before Image, Recovery for the transaction didn’t commit.**
  - **U_1：如果事务T修改数据库元素X，则更新日志<T,X,V>必须在X的新值写到磁盘前写到磁盘；**
  - **U_2：如果事务T提交，则日志记录<Commit T>必须在事务T改变的所有DB元素写到磁盘后再写到磁盘。**
- **REDO Logs**
  - **only After Image, Recovery for the transaction did commit.**
  - **R_1：在修改磁盘上的任何数据库元素X之前，要保证所有与 X 的这一修改有关的日志记录 (包括更新记录<T,X,V> 和提交记录 <Commit T>) 都必须出现在磁盘上。**

- undo与redo日志的不足
  - Undo 日志要求数据在事务结束后立即写到磁盘，可能增加需要执行磁盘I/O的次数；
  - Redo 日志要求事务提交和日志记录刷新之前将所有修改过的数据保留在内存缓冲区中，可能增加事务需要的平均缓冲区的数量；
  - 如果被访问的数据对象X不是完整的磁盘块，那么在undo日志与redo日志之间可能产生相互矛盾的请求。
- undo/redo日志可以解决上述矛盾。

undo/redo日志记录的格式

- 与undo日志或redo日志的格式基本一样，区别在于更新记录的格式：<T,X,v,w>
  - 不仅记录更新前的值v (before image)，同时也要记录更新后的新值w (after image)
  - 因此该种类型的日志既能用于未结束事务的撤消(UNDO)，也能用于已提交事务的重做(REDO)

undo/redo日志的记载规则

- UR_1：在由于某个事务T所做的改变而修改磁盘上的数据库元素X之前，更新记录<T,X,v,w>必须出现在磁盘上。

图3 undo/redo日志的例子

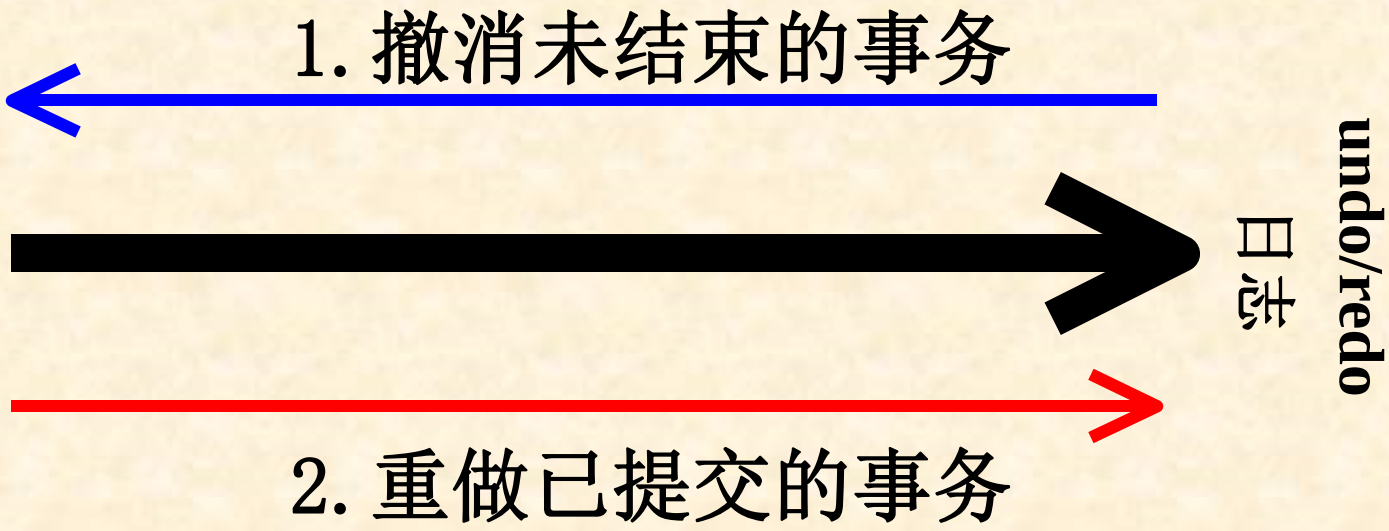
在这里，只要确信更新记录 **<T,A,8,16>** 已经出现在磁盘中（*通过执行‘Flush Log’操作来保证*），还可以在更早的时候执行 **Output(A)**操作，而不管修改**A**的事务**T**是否已经被提交。（规则 UR_1）

							undo/redo日志
							<Start T>
							<T, A, 8, 16>
							<T, B, 8, 16>
8	Flush Log						
9	Output(A)		16	16	16	8	
10							<Commit T>
11	Output(B)		16	16	16	16	

- 第10步也可以出现在第9步之前或第11步之后。

使用undo/redo日志的故障恢复过程

- 根据<Commit T>是否已经出现在磁盘中来决定事务T是否已经被提交，然后：
  1. 先按照从后往前的顺序，撤消(undo)所有未提交的事务
  2. 再按照从前往后的顺序，重做(redo)所有已提交的事务



- 为了确保已经在日志中写入<Commit T>记录的事务T确实被提交，为 undo/redo日志新增加了一条记载规则：
  - UR_2：在每一条 <Commit T> 后面必须紧跟一条 Flush Log 操作