# 深入理解并实现基本的 CRDT（无冲突复制数据类型）数据结构

马浩琨

Oct 23, 2025

在当今分布式系统日益普及的背景下，数据一致性成为开发者必须面对的挑战。尤其是在多用户协作场景中，如在线文档编辑，如何确保所有副本最终一致，而不依赖复杂的冲突解决机制？CRDT（Conflict-free Replicated Data Types）作为一种优雅的解决方案，通过数学原理和巧妙的设计，实现了强最终一致性。本文将带你从理论到实践，深入解析 CRDT 的核心思想，并亲手实现两个基础类型：G-Counter 和 PN-Counter。想象一个多人在线协作文档的场景，比如 Notion 或 Google Docs。当两个用户同时在文档的同一位置插入不同内容时，系统如何保证所有用户的视图最终一致？传统方法如悲观锁会破坏用户体验，无法支持离线工作；乐观锁则将冲突解决的负担抛给用户，导致操作中断；中央仲裁器则存在单点故障风险。CRDT 通过设计可交换、关联和幂等的操作，使得无论操作顺序如何，所有副本都能自动收敛到相同状态，无需人工干预。这不仅提升了系统的可用性，还降低了开发复杂度。

## 1　CRDT 的核心思想：数学的优雅

CRDT 的核心在于其数学基础，特别是半格理论。首先，我们需要区分状态复制和操作复制。状态复制直接同步完整数据状态，简单但数据量大；操作复制同步导致状态变化的操作，高效但需处理顺序问题。CRDT 属于操作复制范畴，但通过设计消除了对顺序的依赖。半格是一种数学结构，定义在偏序集上，能够计算任意两个元素的上确界。在 CRDT 中，状态被组织成半格，合并函数即为计算上确界的操作。例如，求一组数字的最大值，无论比较顺序如何，结果总是相同。这保证了收敛性。CRDT 分为基于状态和基于操作两大流派。基于状态的 CRDT 直接同步状态，合并函数计算上确界，可靠但不依赖可靠传输，缺点是状态可能膨胀；基于操作的 CRDT 同步操作，要求操作满足交换律、结合律和幂等律，传输数据量小，但需可靠广播。

## 2　实战：实现一个 G-Counter（增长计数器）

G-Counter 适用于分布式计数器场景，每个副本只能增加自身计数，但需获取全局总和。其数据结构不使用单一整数，而是采用字典，键为副本 ID，值为该副本的计数值。例如，状态可能表示为 `{ replica_A: 5, replica_B: 3, replica_C: 7 }`。

操作设计包括 increment 和 value 方法。increment 方法只增加当前副本自身的计数。以下是用 JavaScript 伪代码实现的示例：

```
function increment() {
   this.state[this.replicaId] += 1;
}
```

在这段代码中，this.state 是存储计数的字典，this.replicaId 是当前副本的标识符。increment 操作通过简单递增当前副本的计数值，确保每个副本只修改自身部分，从而避免冲突。这种设计的关键在于，操作是局部的，不会影响其他副本。

value 方法用于计算所有副本值的总和：

```
function value() {
   return Object.values(this.state).reduce((sum, count) => sum + count, 0);
}
```

这里，Object.values(this.state) 获取所有计数值，reduce 方法对它们求和，初始值为 0。即使状态为空，也能正确返回结果。这个方法体现了全局视图的生成，而不需要同步所有操作。

合并策略是 CRDT 的精华，它基于半格思想。合并函数 merge 接收另一个状态，并对每个副本的计数取最大值：

```
function merge(otherState) {
   const mergedState = {};
   const allReplicaIds = new Set([...Object.keys(this.state), ...Object.keys(
       ↪ otherState)]);
   for (let id of allReplicaIds) {
      mergedState[id] = Math.max(this.state[id] || 0, otherState[id] || 0);
   }
   this.state = mergedState;
}
```

在这个函数中，首先创建新对象 mergedState，然后获取所有涉及的副本 ID（包括当前状态和另一个状态的键）。对于每个副本 ID，使用 Math.max 取当前状态和另一个状态中该副本计数的最大值；如果某个副本不存在，则用 0 作为默认值。最后，更新当前状态为合并后的状态。为什么这种方法有效？因为 Math.max 操作满足交换律、结合律和幂等律，这正是半格上确界操作的体现。交换律确保顺序无关，结合律允许任意分组合并，幂等律保证重复合并不改变结果。

为了演示，假设副本 A 和 B 初始状态为空。A 执行两次 increment，状态变为 { A: 2 }；B 执行一次 increment，状态变为 { B: 1 }。合并时，对每个副本取最大值，结果状态为 { A: 2, B: 1 }，value 返回 3。即使合并顺序相反，结果相同，证明了收敛性。

## 3 进阶：实现一个 PN-Counter（增减计数器）

G-Counter 只能增加计数，无法处理减少操作，这在现实场景如取消赞或减少库存中不够用。PN-Counter 通过分解为两个 G-Counter 来解决这一问题：P 记录增加操作，N 记录减少操作。最终值等于 P 的值减去 N 的值。例如，状态可能表示为 { P: { A: 3, B: 2 }, N: { A: 1, B: 0 } }。

操作设计包括 increment、decrement 和 value 方法。increment 调用 P 的 increment，decrement 调用 N 的 increment，value 计算差值。以下是伪代码示例：

```
function increment() {
   this.P.increment();
}
```

```
4
function decrement() {
6   this.N.increment();
}
8
function value() {
10  return this.P.value() - this.N.value();
}
```

在这里，increment 和 decrement 分别委托给 P 和 N 的 increment 方法，确保增减操作独立跟踪。value 方法简单计算差值，避免了直接操作全局状态。

合并策略是分别合并 P 和 N 两个 G-Counter：

```
function merge(other) {
   this.P.merge(other.P);
3  this.N.merge(other.N);
}
```

由于 P 和 N 的合并都是收敛的（基于 G-Counter 的合并函数），它们的差值也会收敛。例如，假设副本 A 增加两次后减少一次，状态为 P: { A: 2 }, N: { A: 1 }；副本 B 增加一次，状态为 P: { B: 1 }, N: { B: 0 }。合并后，P 变为 { A: 2, B: 1 }，N 变为 { A: 1, B: 0 }，value 返回 (2+1) - (1+0) = 2。这种设计的正确性源于增减操作的独立性：合并时取最大值，确保不丢失任何操作信息。

## 4 超越计数器：CRDT 的广阔世界

除了计数器，CRDT 还应用于更复杂的数据结构。OR-Set（观察移除集合）解决了朴素集合中「添加」和「移除」操作不满足交换律的问题。在 OR-Set 中，每个元素关联唯一标记（如 UUID）；添加时生成新标记，移除时记录「墓碑」集合；查询时，元素存在当且仅当其标记在添加集且不在移除集。这确保了无论操作顺序如何，最终集合状态一致。其他 CRDT 类型包括寄存器（如 LWW-Register，基于时间戳实现最后写入获胜）、地图（由其他 CRDT 组成的复合结构）和文本序列（如 RGA 或 Logoot，用于协同编辑）。这些领域更复杂，但原理类似，通过设计可交换操作确保收敛。

## 5 CRDT 的优缺点与最佳实践

CRDT 的优势在于提供强最终一致性，保证无冲突收敛；支持低延迟操作，本地立即生效，无需等待服务器；以及高可用性，允许网络分区和离线工作。然而，CRDT 也面临挑战，如状态膨胀（由于墓碑或向量元数据增长，需要压缩机制）、语义限制（并非所有数据结构都能轻松设计成 CRDT）、复杂性（理解和使用比传统结构更复杂）和安全性问题（恶意副本可能污染状态，需额外机制如纯操作 CRDT）。适用场景包括协作文档编辑、分布式计数器和排行榜、购物车、分布式配置和标志管理，以及物联网设备状态同步。在这些场景中，CRDT 能有效提升系统鲁棒性和用户体验。

总之，CRDT 通过数学的优雅解决了分布式数据同步的核心挑战。从 G-Counter 到 PN-Counter 的实现，我们看到了「设计可交换操作」和「利用半格合并」的核心方法论。鼓励读者动手实现这些基础 CRDT，并尝试

更复杂的类型。进一步学习资源包括论文《A comprehensive study of Convergent and Commutative Replicated Data Types》、开源库如 automerge、yjs、delta-crdt，以及网站 crdt.tech。通过实践，你将更深入理解分布式系统的魔法。

## 6 附录：代码仓库链接

本文涉及的 G-Counter 和 PN-Counter 完整实现代码已发布在 GitHub 仓库中，欢迎访问和贡献。