

# 实验一：DHT开发

Kademlia DHT 是分布式哈希表的一种实现，它拥有一些很好的特性，如下：节点 ID 与 KEY 是同样的值域，都是使用 SHA-1 算法生成的 160 位摘要，这样大大简化了查询时的信息量，更便于查询。可以使用 XOR，计算任意两个节点的距离或节点和关键字的距离。查找一条请求路径的时候，每个节点的信息是完备的，只需要进行  $\log(n)$  量级次跳转。可根据查询速度和存储量的需求调整每个节点需要维护的 DHT 大小。

而 Kademlia 中，最为关键的就是 K\_Bucket 算法。

以节点 u 为例，其路由表的生成过程如下：

1. 最初，u 的路由表为一个单独的 K 桶，覆盖了整个 160bit ID 空间
2. 当学习到新的节点信息后，则 u 会尝试把新节点的信息，根据其前缀值插入对应的 K 桶中。
  - a. 该桶没有满，则新节点直接插入这个 K 桶中；
  - b. 该 K 桶已经满了：
    - i. 如果该 K 桶覆盖范围包含了节点的 ID，则把该 K 桶分裂为两个大小相同的新 K 桶，并对原桶内的节点信息按照新的 K 桶前缀值进行重新分配；
    - ii. 如果该 K 桶覆盖范围没有包含节点的 ID，则直接丢弃该新节点信息。
3. 上述过程不断重复，直到满足路由表的要求。达到距离近的节点的信息多、距离远的节点的信息少的结果，这样就保证了路由查询过程能快速收敛。

## 实验目的

通过实现 Kademlia DHT 中的 K\_Bucket 算法，加深对分布式存储中的 Kademlia DHT 算法的理解，并提升其编程能力和分布式系统设计能力。

## 实验前提：

1. 掌握了 Kademlia DHT 算法的基本原理和相关概念；
2. 学生应具备基本的编程知识和对 Golang 或 TypeScript 编程语言的了解；
3. 假设地址长度是 20byte，每个桶中节点的数量为 3 个。

## 实验内容

本次实验主要集中在 Kademlia DHT 中的 K\_Bucket 算法的实现，学生需要使用 Golang 或 TypeScript 完成以下任务：

## 1. K\_Bucket算法实现：

- a. 学生需要实现Kademlia DHT中的K\_Bucket数据结构，包括桶（Bucket）、节点（Node）等相关数据结构。
- b. 学生应能够正确处理节点的插入、删除和更新等操作，根据节点ID将其分配到正确的桶中。

## 2. 接口实现：

需要为K\_Bucket结构提供两个接口：

- insertNode(nodeId string)：将给定的NodeId插入到正确的桶中。
- printBucketContents()：打印每个桶中存在的NodeID。

## 加分项：多节点模拟

1. 实现Peer结构，每个Peer都拥有上述实验中的内容；
2. 为之前的K\_Bucket结构增加一个接口：FindNode（nodeId string）[]string。当别人调用这个接口时，先根据两个节点之间的距离，算出桶的ID，然后查找对应桶中是否有这个节点：
  - a. 如果有这个节点，则返回对应节点；
  - b. 如果没有这个节点，从对应的桶中随机返回2个节点。
3. 当有节点加入时，通过其中一个peer对自己的节点信息进行广播；
4. 在主程序中初始化多个Peer（暂定5个），然后生成200个新的Peer，通过之前的5个初始化的节点，加入这个网络；
5. 打印出这205个节点每个节点的桶的信息。

## 功能测试

1. 需要自行编写测试代码，对实现的K\_Bucket算法进行功能测试，验证插入和打印功能的正确性。
2. 可以设计额外的测试用例，评估算法在不同场景下的性能表现和正确性。

## 实验报告

1. 需要撰写实验报告，记录实现过程中的思考、遇到的问题和解决方案。
2. 报告中应包括实验结果的截图和分析，验证实现的正确性和性能表现。