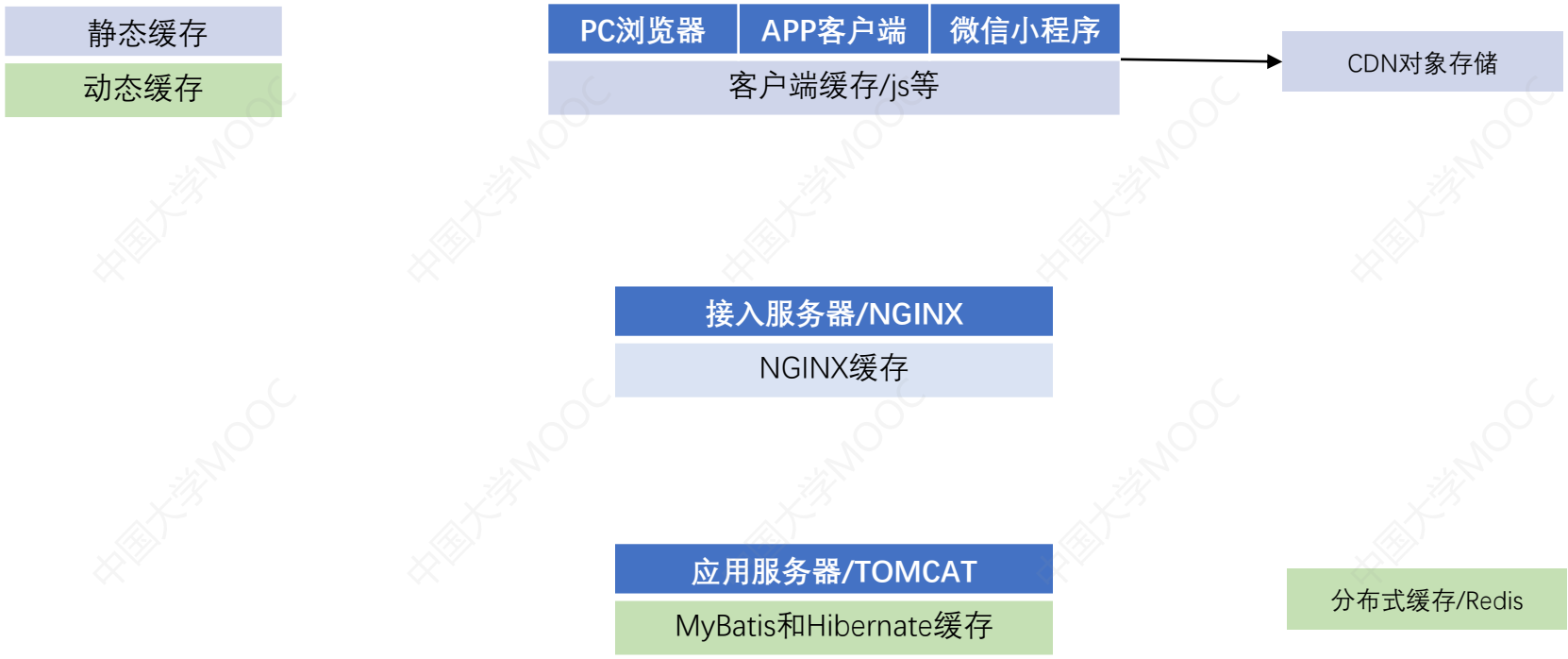
JavaEE平台技术
缓存机制

邱明 博士

厦门大学信息学院

mingqiu@xmu.edu.cn

1 为什么需要缓存

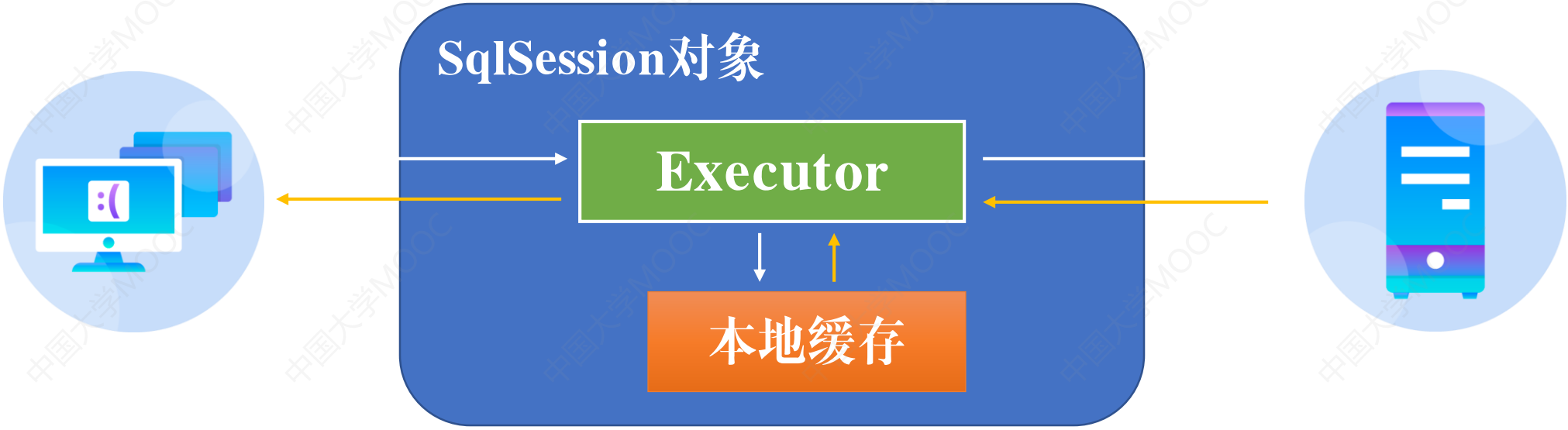

静态缓存
动态缓存
PC浏览器
APP客户端
微信小程序
客户端缓存/js等
CDN对象存储
接入服务器/NGINX
NGINX缓存
应用服务器/TOMCAT
MyBatis和Hibernate缓存
分布式缓存/Redis

2 MyBatis缓存

- 一级缓存和二级缓存。
  - 在参数和 SQL 完全一样的情况下，优先命中一级缓存，避免直接对数据库进行查询，提高性能。
  - 把执行的方法和参数通过算法生成缓存的键值， 将键值和结果存放在一个 Map 中， 如果后续的键值一样， 则直接从 Map 中获取数据；
  - 任何的 UPDATE, INSERT, DELETE 语句都会清空缓存。

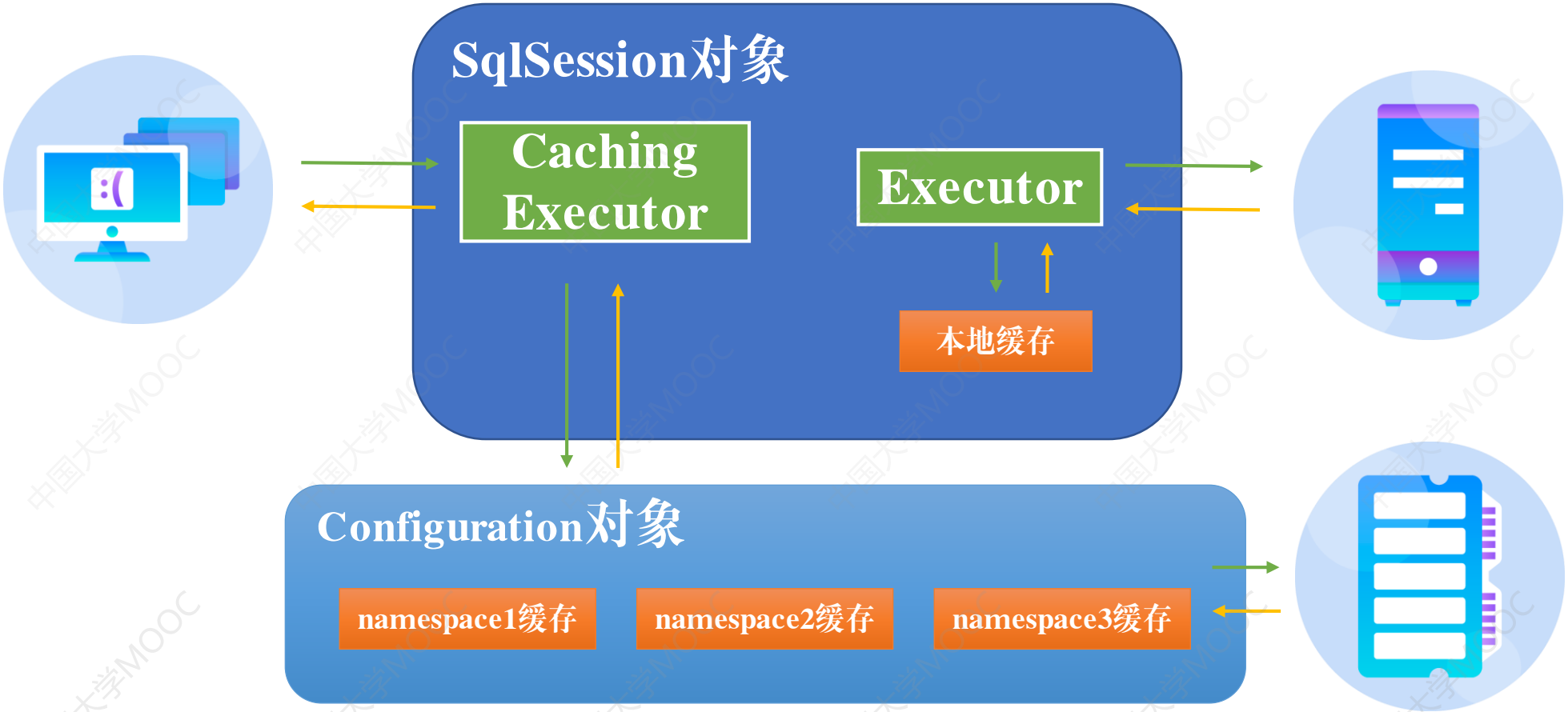
2 MyBatis缓存

- MyBatis 一级缓存



2 MyBatis缓存

- MyBatis 二级缓存



3 Redis

- Redis 是高性能的内存key-value数据库。
  - 支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。
  - 不仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。
  - 支持数据的备份，即master-slave模式的数据备份。
  - Redis采用IO多路复用，单线程结构 – 读的速度是110000次/s, 写的速度是81000次/s 。
  - Redis的所有操作都是原子性的。多个操作也支持事务。

4. Redis的常用数据类型

类型	简介	特性
String(字符串)	是 Redis 最基本的数据类型并且是二进制安全的	可以包含任何数据,比如jpg图片或者序列化的对象,一个键最大能存储512M
Hash(字典)	键值对集合,	适合存储对象,并且可以像数据库中update一个属性一样只修改某一项属性值
List(列表)	链表(双向链表)，按照插入顺序排序。	增删快,提供了操作某一段元素的API
Set(集合)	哈希表实现，元素不重复	添加、删除,查找的复杂度都是O(1) 。为集合提供了求交集、并集、差集等操作
Bitmap	是一种实现对位的操作	借助字符串进行位操作

4. Redis的常用数据类型

- 字符串
  - 可以是字符串、数字、二进制图片、音频等，大小不超过512MB

SET key value [ex seconds] [px milliseconds] [nx|xx]

```
redisTemplate.opsForValue().set(key, value, timeout, TimeUnit.SECONDS)
```

GET key

```
redisTemplate.opsForValue().get(key)
```

INCRBY key delta

```
redisTemplate.opsForValue().increment(key, delta)
```

EXISTS key

```
redisTemplate.hasKey(key)
```

4. Redis的常用数据类型

- Hash
  - 键值对集合,可以像数据库中update一个属性一样只修改某一项属性值

HSET key field

redisTemplate.opsForHash().put(key, field, value)

HGET key field

redisTemplate.opsForHash().get(key, field)

4. Redis的常用数据类型

- List
  - 链表(双向链表)，按照插入顺序排序。

LPOP key

redisTemplate.opsForList().leftPop(key)

LLEN key

redisTemplate.opsForList().size(key)

RPUSHX key value

redisTemplate.opsForList().rightPush(key, value)

4. Redis的常用数据类型

- Set
  - 为集合提供了求交集、并集、差集等操作

SADD key value1 … valueN

redisTemplate..opsForSet().add(key, values)

SMEMBERS key

redisTemplate.opsForSet().members(key)

SUNIONSTORE destination key1 key2

redisTemplate.opsForSet().unionAndStore(key1, key2, destination)

SCARD key

redisTemplate.opsForSet().size(key)

SREM key value1 … valueN

redisTemplate.opsForSet().remove(key, value1, …, valueN)

4. Redis的常用数据类型

- Bitmap
  - 进行位操作

SETBIT key offset value

```
redisTemplate.opsForValue().setBit(key, offset, value)
```

GETBIT key offset

```
redisTemplate.opsForValue().getBit(key, offset)
```

BITCOUNT key start end

```
redisTemplate.execute((RedisCallback<Long>) connection -> connection.bitCount(key.getBytes(), start, end))
```

4. Redis的常用数据类型

- Redis内置 Lua 解释器
- 使用 EVAL 命令对 Lua 脚本进行执行，并获取脚本的返回值

```lua
-- 减库存
local setKey = KEYS[1]
local quantity = tonumber(ARGV[1])
math.randomseed(ARGV[2])

if (redis.call('exists', setKey) == 1) then
    local num = redis.call('scard', setKey)
    local keys = redis.call('smembers', setKey)
    local init = math.random(num)

    for i = 1, num do
        local curr = math.fmod(init + i, num) + 1;
        local stock = tonumber(redis.call('get', keys[curr]))
        if (stock >= quantity) then
            local res = redis.call('incrBy', keys[curr], 0 - quantity)
            if (res == 0) then
                redis.call('srem', setKey, keys[curr])
            end
            return res
        end
    end
end

return -1
```

5. Redis的应用场景

- 利用缓存存储对象，加快访问的速度

```
public ShopChannel findObjById(Long id) throws RuntimeException {
    ShopChannel ret = null;
    if (null != id) {
        String key = String.format(KEY, id);
        if (redisUtil.hasKey(key)) {
            ret = (ShopChannel) redisUtil.get(key);
        } else {
            ShopChannelPo po = shopChannelPoMapper.selectByPrimaryKey(id);
            ret = cloneObj(po, ShopChannel.class);
            redisUtil.set(key, ret, timeout);
        }
        ret.setChannelDao(this.channelDao);
    }
    return ret;
}
```

```
public void updateObjById(ShopChannel shopChannel) throws RuntimeException {
    if (null != shopChannel && null != shopChannel.getId()){
        String key = String.format(KEY, shopChannel.getId());
        ShopChannelPo po = cloneObj(shopChannel, ShopChannelPo.class);
        redisUtil.del(key);
        shopChannelPoMapper.updateByPrimaryKeySelective(po);
    }
}
```

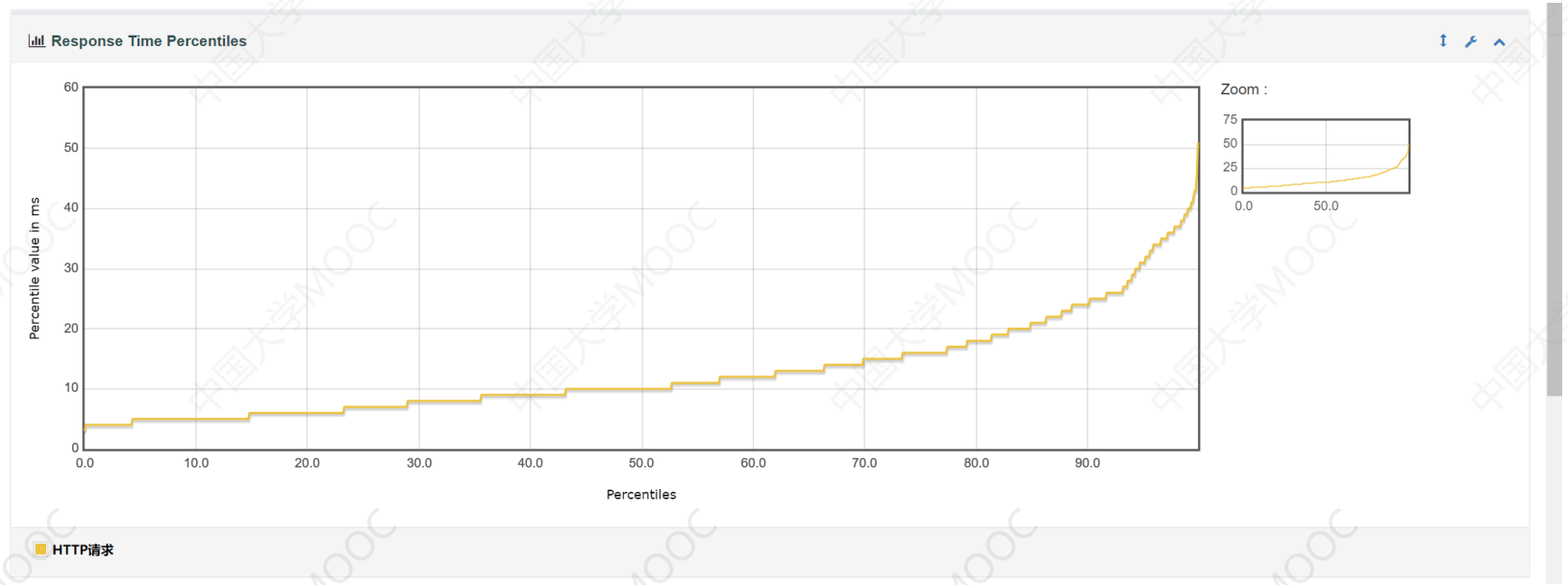
5. Redis的应用场景

- 利用Set，计算权限的并集

```
List<Long> roleIds = (List<Long>) returnObject.getData();
String key = String.format(USERKEY, id);
Set<String> roleKeys = new HashSet<>(roleIds.size());
for (Long roleId : roleIds) {
    String roleKey = String.format(ROLEKEY, roleId);
    roleKeys.add(roleKey);
    if (!redisUtil.hasKey(roleKey)) {
        ReturnObject returnObject1 = roleDao.loadRole(roleId);
        if (returnObject1.getCode() != ReturnNo.OK) {
            return returnObject1;
        }
    }
}
if (roleKeys.size() > 0) {
    redisUtil.unionAndStoreSet(key,roleKeys, key);
}
```

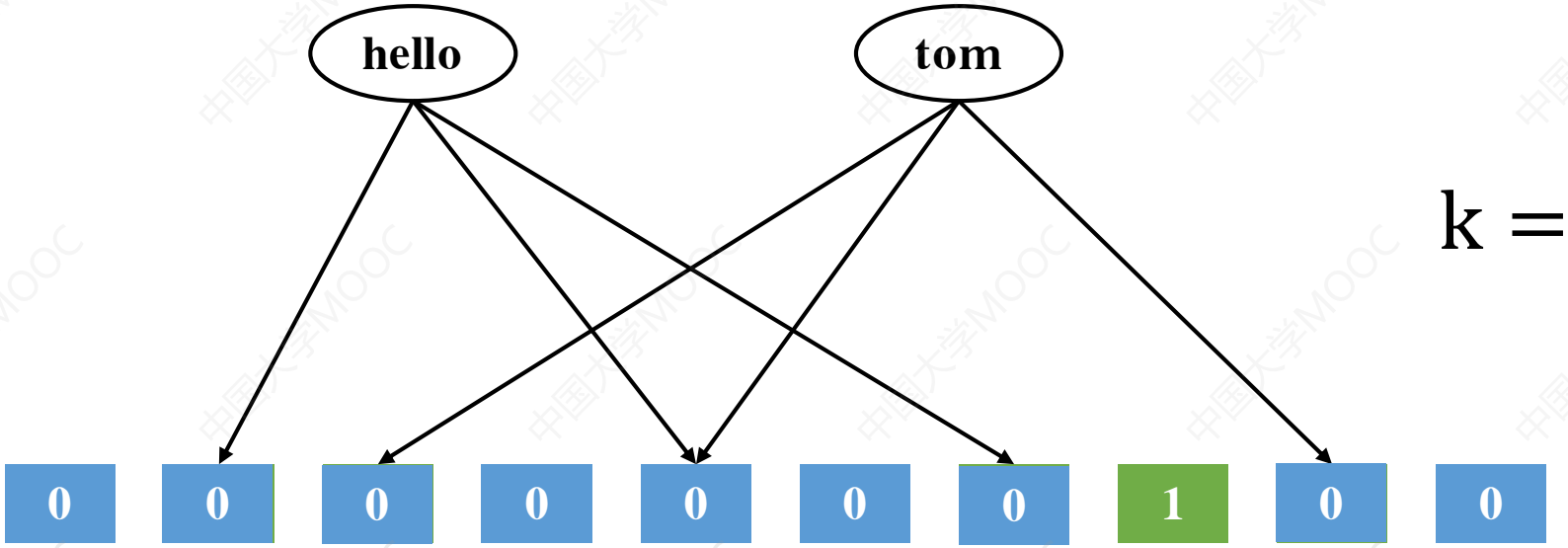
5. Redis的应用场景

- 利用Redis实现库存的高并发扣减
  - 将库存量读入Redis中，利用Redis的原子性操作实现库存的高并发扣减
  - 不更新数据库，用带事务的消息队列将库存量写回。



5. Redis的应用场景

- Bloom过滤器
  - 是1970年由布隆提出的，用以判断一个元素是否在一个集合里。



$$k = \frac{m}{n} ln2$$

6. 为啥不用Spring Cache

- Spring Cache 是Spring 提供的一整套的缓存解决方案（JSR-107），
- 提供一整套的接口和代码规范、配置、注解等，用于整合各种缓存方案，如Redis、Caffeine、Guava Cache、Ehcache。

```
@Cacheable(cacheNames = {"stu"}, keyGenerator = "MyKeyGenerator", condition = "#a0 == 1", unless = "#a0 == 1")
@Override
public Student getStu(Integer id) {
        Student student = studentMapper.selectByPrimaryKey(id);
        System.out.println(student);
        return student;
}
```

```
@CachePut(value = "stu", key = "#student.id")
@Override
public Student updateStu(Student student){
        System.out.println(student);
        studentMapper.updateByPrimaryKey(student);
        return student;
}
```

7.Redis的内存管理

- 惰性删除
  - 当访问到该键时，如果该键值已过期，则返回空，并删除该键。
- 定期删除
  - 定期运行，随机中选取一些键并删除其中过期的键

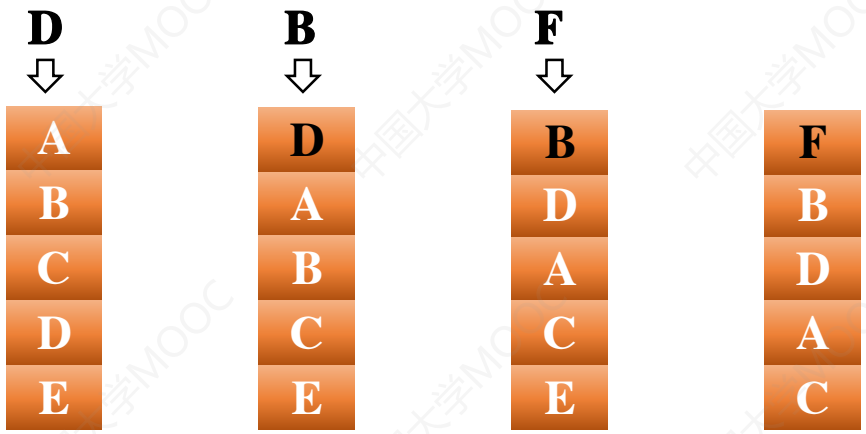
7.Redis的内存管理

- 内存管理机制
  - volatile-lru：利用LRU算法移除有过期时间的key。
  - volatile-random：随机移除有过期时间的key。
  - volatile-ttl：移除即将过期的key，根据最近过期时间来删除
  - allkeys-lru：利用LRU算法移除任何key。
  - allkeys-random：随机移除任何key。
  - noeviction：不移除任何key，只是返回一个写错误。

maxmemory-policy noeviction

7.Redis的内存管理

- LRU（Least recently used，最近最少使用）
  - 随机采样一部分键值，淘汰其中最久没有访问的键。



maxmemory-samples 5

7.Redis的内存管理

- 缓存穿透
  - 故意去请求缓存中不存在的数据，导致所有的请求都穿透到数据库访问上，造成数据库的负载增加。
- 缓存雪崩
  - 缓存同一时间大面积的失效，导致所有请求同时到达数据库，造成数据库负载增加。