《Java性能优化与面试21讲》

李国

— 拉勾教育出品 —

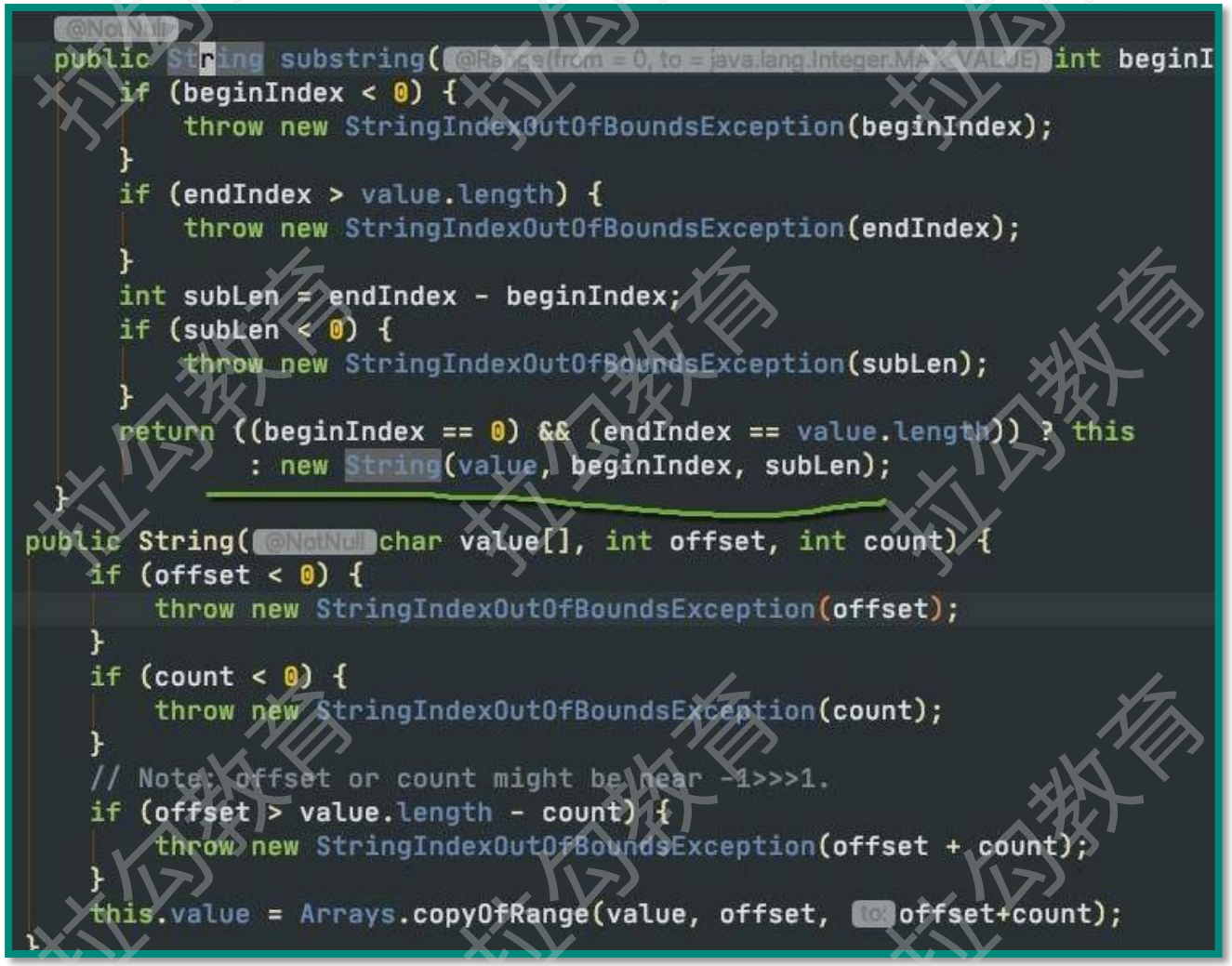
10 | 案例分析：大对象复用的目标和注意点

前言

为什么大对象会影响我们的应用性能呢？

1. 大对象占用的资源多，垃圾回收器要花一部分精力去对它进行回收
2. 大对象在不同的设备之间交换，会耗费网络流量，以及昂贵的 I/O
3. 对大对象的解析和处理操作是耗时的，对象职责不聚焦，就会承担额外的性能开销

String 的 substring 方法



```
@NotNull
public String substring(@Range(from = 0, to = java.lang.Integer.MAX_VALUE) int beginI
    if (beginIndex < 0) {
        throw new StringIndexOutOfBoundsException(beginIndex);
    }
    if (endIndex > value.length) {
        throw new StringIndexOutOfBoundsException(endIndex);
    }
    int subLen = endIndex - beginIndex;
    if (subLen < 0) {
        throw new StringIndexOutOfBoundsException(subLen);
    }
    return ((beginIndex == 0) && (endIndex == value.length)) ? this
            : new String(value, beginIndex, subLen);
}
public String(@NotNull char value[], int offset, int count) {
    if (offset < 0) {
        throw new StringIndexOutOfBoundsException(offset);
    }
    if (count < 0) {
        throw new StringIndexOutOfBoundsException(count);
    }
    // Note: offset or count might be near -1>>>1.
    if (offset > value.length - count) {
        throw new StringIndexOutOfBoundsException(offset + count);
    }
    this.value = Arrays.copyOfRange(value, offset, to: offset+count);
}
```

String 的 substring 方法


```
public String substring(int beginIndex, int endIndex) {
    if (beginIndex < 0) {
        throw new StringIndexOutOfBoundsException(beginIndex);
    }
    if (endIndex > count) {
        throw new StringIndexOutOfBoundsException(endIndex);
    }
    if (beginIndex > endIndex) {
        throw new StringIndexOutOfBoundsException(endIndex - beginIndex);
    }
    return ((beginIndex == 0) && (endIndex == count)) ? this :
        new String(offset + beginIndex, endIndex - beginIndex, value);
}


// Package private constructor which shares value array for speed.
String(int offset, int count, char value[]) {
    this.value = value;
    this.offset = offset;
    this.count = count;
}
```

String 的 substring 方法

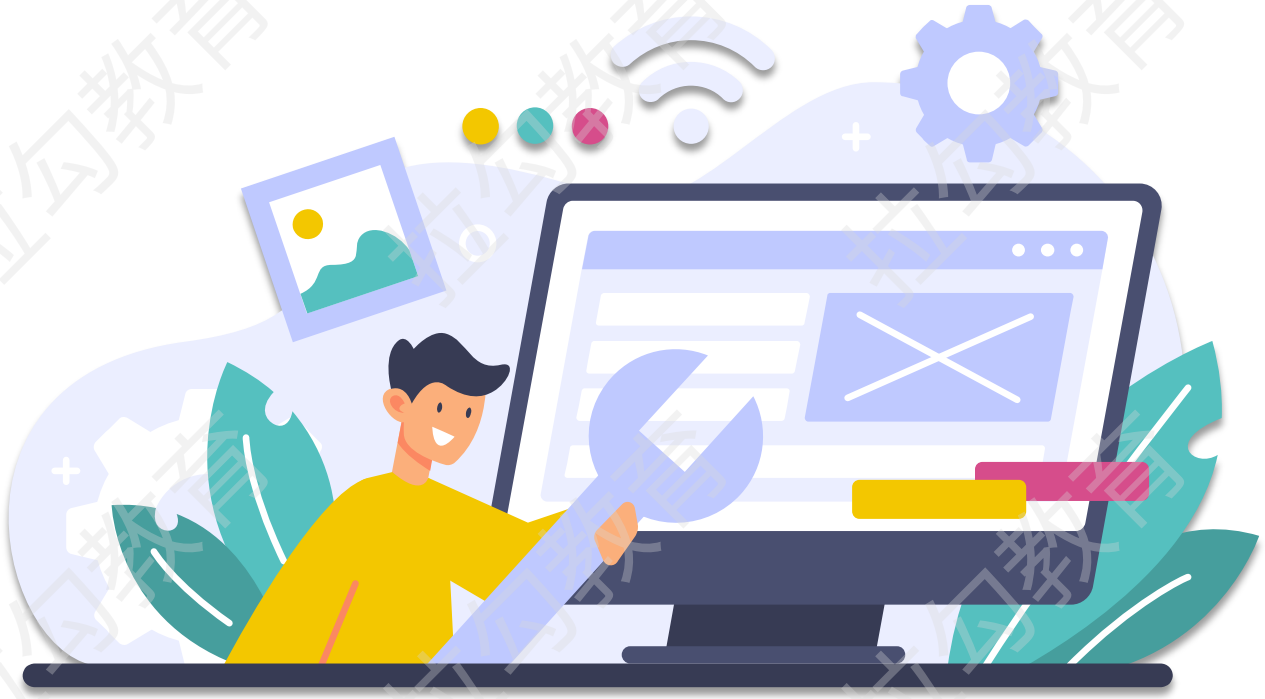

```
String content = dao.getArticle(id);
String summary=content.substring(0,100);
articles.put(id,summary);
```

集合大对象扩容

拉勾教育
互联网人实战大学

对象扩容——在 Java 中是司空见惯的现象

比如 StringBuilder、StringBuffer、HashMap，ArrayList 等



集合大对象扩容

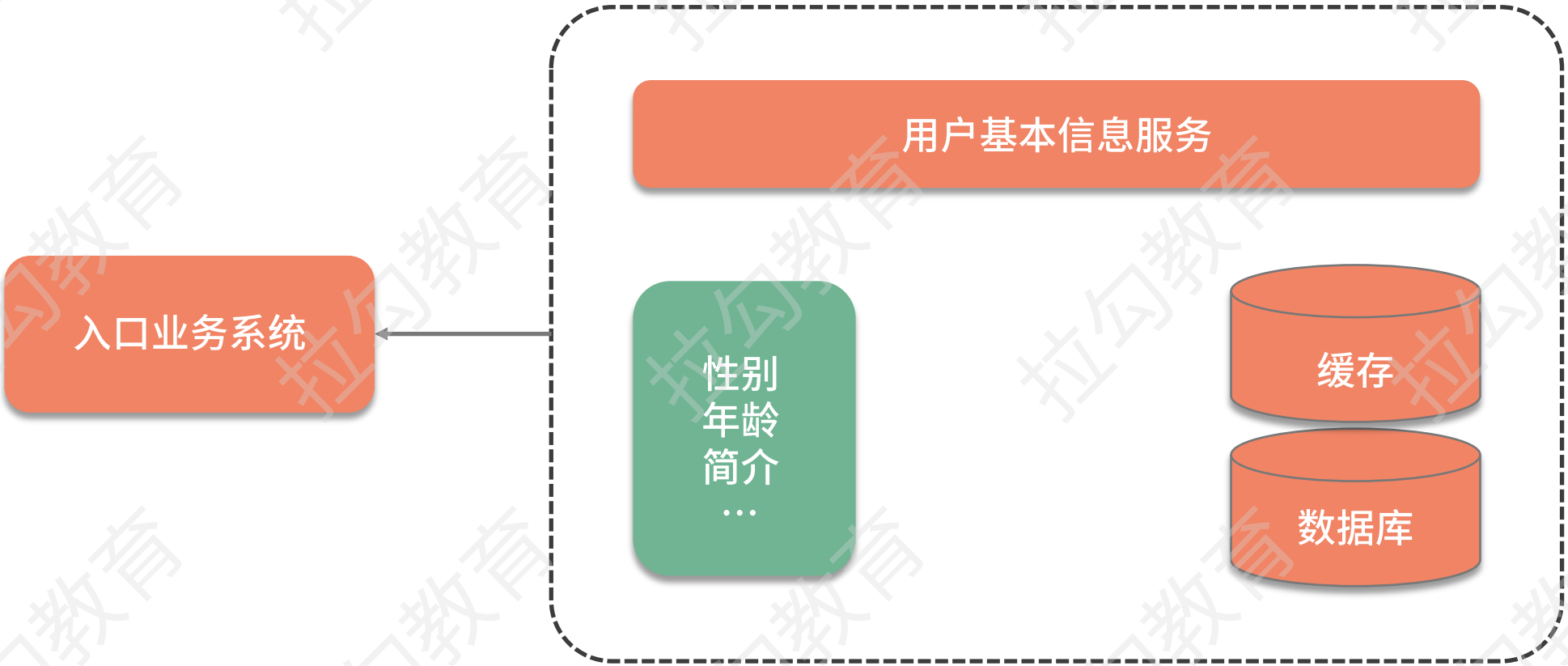

```
void expandCapacity(int minimumCapacity) {
    int newCapacity = value.length * 2 + 2;
    if (newCapacity - minimumCapacity < 0)
        newCapacity = minimumCapacity;
    if (newCapacity < 0) {
        if (minimumCapacity < 0) // overflow
            throw new OutOfMemoryError();
        newCapacity = Integer.MAX_VALUE;
    }
    value = Arrays.copyOf(value, newCapacity);
}
```

集合大对象扩容


```
void addEntry(int hash, K key, V value, int bucketIndex) {
    if ((size >= threshold) && (null != table[bucketIndex])) {
        resize(2 * table.length);
        hash = (null != key) ? hash(key) : 0;
        bucketIndex = indexFor(hash, table.length);
    }
    createEntry(hash, key, value, bucketIndex);
}

void resize(int newCapacity) {
    Entry[] oldTable = table;
    int oldCapacity = oldTable.length;
    if (oldCapacity == MAXIMUM_CAPACITY) {
        threshold = Integer.MAX_VALUE;
        return;
    }
    Entry[] newTable = new Entry[newCapacity];
    transfer(newTable, initHashSeedAsNeeded(newCapacity));
    table = newTable;
    threshold = (int)Math.min(newCapacity * loadFactor, MAXIMUM_CAPACITY + 1);
}
```

保持合适的对象粒度



保持合适的对象粒度

原始的 redis key

```
type: string
key: user_${userid}
value: json
```

保持合适的对象粒度

原始的 redis key

```
type: string
key: user_${userid}
value: json
```

两个问题

- 查询其中某个字段的值，需要把所有 json 数据查询出来，并自行解析
- 更新其中某个字段的值，需要更新整个 json 串，代价较高

保持合适的对象粒度

对 Redis 中的数据进行了以下设计，采用 hash 结构而不是 json 结构

```
type: hash
key: user_${userid}
value: {sex:f, id:1223, age:23}
```

Bitmap 把对象变小

系统中就频繁用到了用户的性别数据，用来发放一些礼品

推荐一些异性的好友，定时循环用户做一些清理动作等

是否在线，是否签到，最近是否发送信息等，从而统计一下活跃用户等

Bitmap 把对象变小

Bitmap 把对象变小

在 Java 虚拟机规范里，描述是：将 Boolean 类型映射成的是 1 和 0 两个数字，它占用的空间是和 int 相同的 32 位

```
int a= 0b0001_0001_1111_1101_1001_0001_1111_1101;
```

Bitmap 把对象变小

拉勾教育
互联网人实战大学

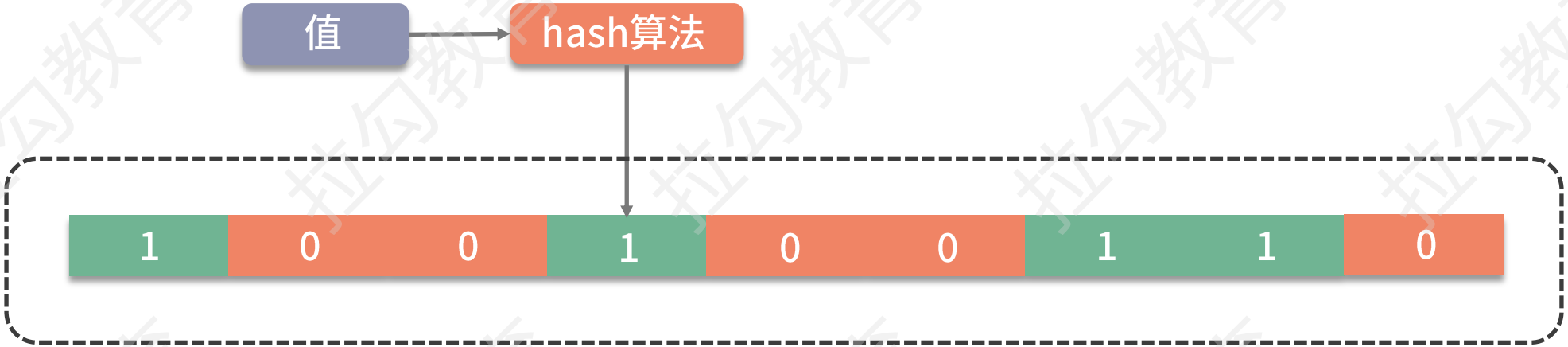
```
static BitSet missSet = new BitSet(010_000_000_000);
static BitSet sexSet = new BitSet(010_000_000_000);
String getSex(int userId) {
  boolean notMiss = missSet.get(userId);
  if (!notMiss) {
    //lazy fetch
    String lazySex = dao.getSex(userId);
    missSet.set(userId, true);
    sexSet.set(userId, "female".equals(lazySex));
  }
  return sexSet.get(userId) ? "female" : "male";
}
```

Bitmap 把对象变小

给出一个 1GB 内存的机器，提供 60亿 int 数据，如何快速判断有哪些数据是重复的?

Bitmap 把对象变小

给出一个 1GB 内存的机器，提供 60亿 int 数据，如何快速判断有哪些数据是重复的？



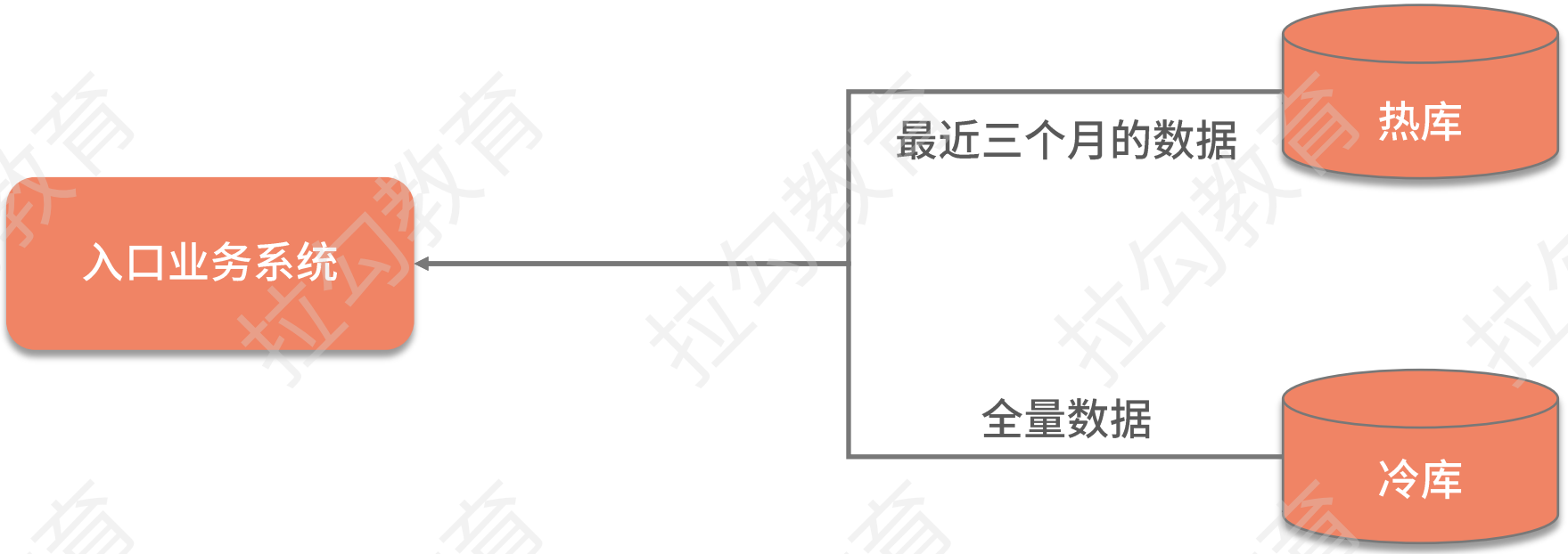
数据的冷热分离

数据除了横向的结构纬度，还有一个纵向的时间维度，对时间维度的优化，最有效的方式就是冷热分离

热数据是靠近用户的，被频繁使用的数据

冷数据是那些访问频率非常低，年代非常久远的数据

数据的冷热分离

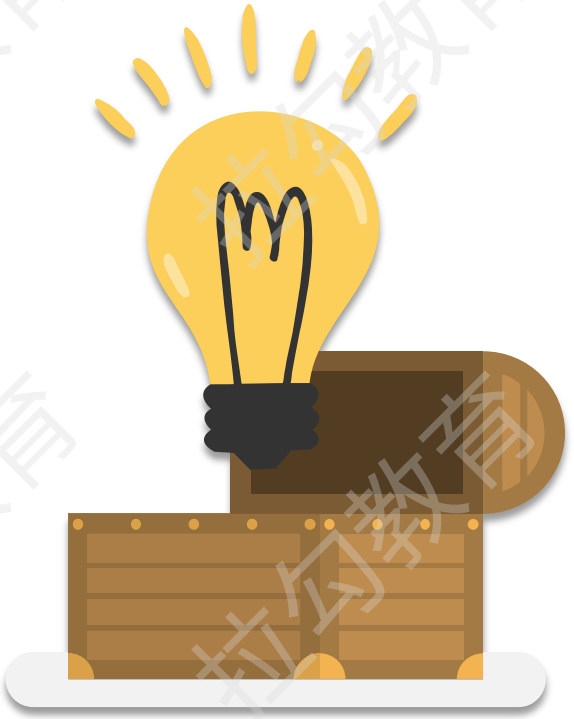


数据的冷热分离

数据双写

把对冷热库的插入、更新、删除操作，全部放在一个统一的事务里面

由于热库（比如 MySQL）和冷库（比如 Hbase）的类型不同，这个事务大概率会是分布式事务



数据的冷热分离

写入MQ分发

通过 MQ 的发布订阅功能，在进行数据操作的时候，先不落库，而是发送到 MQ 中

单独启动消费进程，将 MQ 中的数据分别落到冷库、热库中

使用这种方式改造的业务，逻辑非常清晰，结构也比较优雅

数据的冷热分离

使用 Binlog 同步

针对 MySQL，就可以采用 Binlog 的方式进行同步

使用 Canal 组件，可持续获取最新的 Binlog 数据，结合 MQ，可以将数据同步到其他的数据源中

思维发散

- **常用的数据库索引，就是一种对数据的重新组织、加速**

  B+ tree 可以有效地减少数据库与磁盘交互的次数，通过类似 B+ tree 的数据结构

  将最常用的数据进行索引，存储在有限的存储空间中

- **在 RPC 中常用的序列化**

  有的服务是采用的 SOAP 协议的 WebService，它是基于 XML 的一种协议，内容大传输慢，效率低下

  现在的 Web 服务中，大多数是使用 json 数据进行交互的，json 的效率相比 SOAP 就更高一些

小结

比较老的 JDK 版本中，String 为了复用引起的内容泄漏问题

平常的编码中，一定要注意大对象的回收，及时切断与它的联系

小结

结构纬度的优化和时间维度的优化两种方法：

结构纬度——通过把对象切分成合适的粒度，可以把操作集中在小数据结构上，减少时间处理成本

时间纬度——通过冷热分离的手段，将常用的数据存放在高速设备中，减少数据处理的集合，加快处理速度

Next：第11讲《案例分析：如何用设计模式优化性能》

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容

L / A / G / O / U