week 2(date:20190902-20190908)

Algorithm

leetcode 初级算法-数组篇

1. 两个数组的交集 II：https://leetcode-cn.com/explore/interview/card/top-interview-questions-easy/1/array/26/

   题目描述:给定两个数组，编写一个函数来计算它们的交集。

   思路 (时间复杂度：n*log(n))：

   将两个数组进行排序，分别使用指针i、j指向两个数组，当nums1[i]==nums2[j],将数据保存，否则让较小的数的指针+1，直到i、j不小于数组的长度。

```
public int[] intersect(int[] nums1, int[] nums2) {
        Arrays.sort(nums1);
        Arrays.sort(nums2);
        int i = 0;
        int j = 0;
        List<Integer> tmp=new ArrayList<Integer>();
        while(i<nums1.length && j<nums2.length){
            if(nums1[i]>nums2[j]){
                j++;
            }else if(nums1[i]<nums2[j]){
                i++;
          }else{
              tmp.add(nums1[i]);
              i++;
                j++;

            }
      }
        int[] res=new int[tmp.size()];
        for(int k=0;k<tmp.size();k++){
            res[k]=tmp.get(k);
        }

        return res;
}
```

2. 两数之和:https://leetcode-cn.com/explore/interview/card/top-interview-questions-easy/1/array/29/

   题目描述:给定一个整数数组 `nums` 和一个目标值 `target`，请你在该数组中找出和为目标值的那 **两个** 整数，并返回他们的数组下标。

   思路1 (时间复杂度：O(n^2)) ： 两次遍历数组，判断nums[i] + nums[k] ==target如果等于记录下，两个数据的下标.时间复杂度

   代码1：

```
    public int[] twoSum(int[] nums, int target) {
        int[] res=new int[2];
        for(int i=0;i<nums.length;i++){
            for(int j=i+1;j<nums.length;j++){
                if(nums[i]+nums[j]==target){
                    res[0]=i;
                    res[1]=j;
                }
            }
        }
        return res;
    }
```

思路2（时间复杂度:O(n) ）：

1. 遍历数组保存，数据与下标的映射关系到map中
2. 遍历数组，保存数据与满足条件的数据到map1中
3. 遍历数组，当该数据对应满足条件的数据也存在map1中，且下标不等于自身，说明找到满足条件的数据。记录该数据的下标。

代码2:

```
public int[] twoSum(int[] nums, int target) {
        Map<Integer,Integer> map=new HashMap<Integer,Integer>();
        Map<Integer,Integer> map1=new HashMap<Integer,Integer>();
        int[] res=new int[2];
        for(int i=0;i<nums.length;i++){
            map.put(nums[i],i);
        }
        for(int i=0;i<nums.length;i++){
            map1.put(nums[i],target-nums[i]);
        }
        for(int i=0;i<nums.length;i++){
            if(map1.containsKey(map1.get(nums[i]))&&(i!=map.get(target-
nums[i]))){
                res[0]=i;
                res[1]=map.get(target-nums[i]);
            }
        }
        return res;
    }
```

Review

SLOG: serializable, low-latency, geo-replicated transactions

：https://blog.acolyer.org/2019/09/04/slog/

单词:

1. transactions 事务
2. low-latency 低时延
3. complexity 复杂的
4. sequentially 顺序
5. serializability 串行的
6. guarantees 保证
7. coordination 协调

8. performance 性能
9. throughput 吞吐量
10. maintained 维护
11. respect 尊重
12. involved 涉及
13. mix 混合
14. slight 轻微
15. latency figures 延迟数据
16. implemented 实施
17. leverages 影响力
18. workload 工作量
19. patterns 模式

文章大意：

SLOG是一种可序列化，低延迟，地理复制的事务。SLOG使用严格的可序列化，在此基础上优化提升处理事务的性能。

严格的序列化导致的问题：严格的序列化需要协调开销，协调开销会导致系统性能下降。

SLOG核心思想是：利用*区域亲和力* - 例如，与用户相关的数据可能在其本地区域中访问 - 以仅使用区域内协调来处理尽可能多的事务。（降低协调的范围）

难题：LOG设计中技术上最具挑战性的问题是多宿主数据交易，多宿主交易都需要相互订购，如何产生全局可序列化时间表。（多宿主：数据在不同的区域）（ps:理解不了）

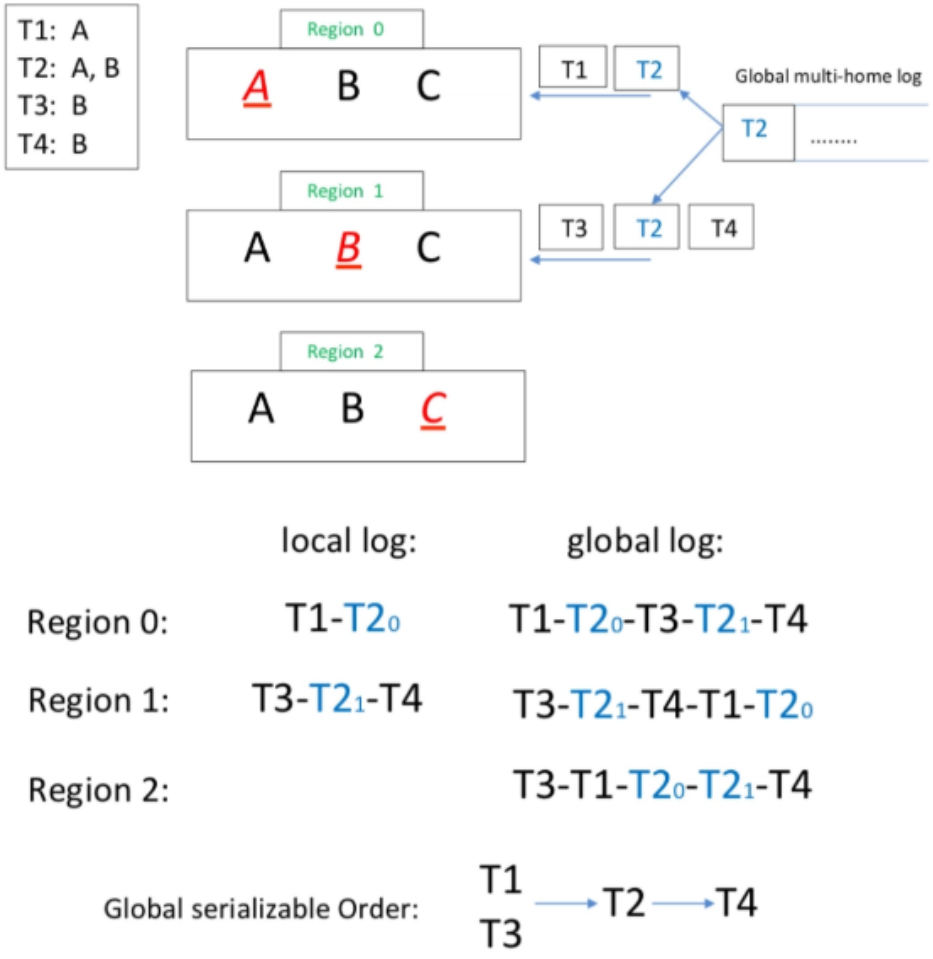

Figure 4: Multi-home transactions in SLOG.

后续：学习下事务与分布式事务。

Tips

- **easypoi：**
  - 当需要导出xlsx文件时，ExportParams exportParams=new ExportParams();

    需要设置exportParams导出类型为xssf，只设置

    ExcelExportUtil.exportExcel(list, ExcelType.XSSF),会导致导出失败，原因是：ExportParams参数初始化时，默认设置ExcelType为HSSF。
  - `@Excel(name = "学生性别", replace = { "男_1", "女_2" }, suffix = "生")`replace可以将数据字典替换文本。
  - `@Excel(name = "进校日期", format = "yyyy-MM-dd")`：format用于自定义格式展现时间类型数据。
  - 继承ExceptionHandlerExceptionResolver类，重写方法可以处理异常，自定义返回前端的数据格式。

Share

Easypoi:

1. easypoi官方文档：http://easypoi.mydoc.io：

easypoi功能是简化Excel导出,Excel模板导出,Excel导入,Word模板导出,通过简单的注解和模板语言(熟悉的表达式语法),完成以前复杂的写法。

2. easypoi的对象定义

```
@ExcelTarget("courseEntity")
public class CourseEntity implements java.io.Serializable {
   /** 主键 */
   private String        id;
   /** 课程名称 */
   @Excel(name = "课程名称", orderNum = "1", width = 25)
   private String        name;
   /** 老师主键 */
   @ExcelEntity(id = "absent")
   private TeacherEntity mathTeacher;

   @ExcelCollection(name = "学生", orderNum = "4")
   private List<StudentEntity> students;
}
```

- 需要注意的点：
  - 实现`Serializable`接口
  - 类上需要注解@ExcelTarget()
  - 字段注解@Excel()，没加注释的内容不会导出。

3. easypoi导出：

```
ExportParams empExportParams = new ExportParams();
// 设置sheet的名称
empExportParams.setSheetName("课程报表1");
Map<String, Object> empExportMap = new HashMap<>();
empExportMap.put("title", empExportParams);
empExportMap.put("entity", CourseEntity.class);
empExportMap.put("data", exportList);
workBook = ExcelExportUtil.exportExcel(empExportMap, ExcelType.HSSF);
```

4. 多sheet页导出：

   注意点:需要定义 `List<Map<String, Object>>` 对象用于接收多个sheet的数据，然后使用 `ExcelExportUtil.exportExcel(sheetsList, ExcelType.HSSF);` 进行导出

```
            ExportParams deptExportParams = new ExportParams();
            // 设置sheet得名称
            deptExportParams.setSheetName("员工报表1");
            // 创建sheet1使用得map
            Map<String, Object> deptExportMap = new HashMap<>();
            // title的参数为ExportParams类型，目前仅仅在ExportParams中设置了
sheetName
            deptExportMap.put("title", deptExportParams);
            // 模版导出对应得实体类型
            deptExportMap.put("entity", DeptUtil.class);
            // sheet中要填充得数据
            deptExportMap.put("data", exportList);
            ExportParams empExportParams = new ExportParams();
            empExportParams.setSheetName("员工报表2");
            // 创建sheet2使用得map
            Map<String, Object> empExportMap = new HashMap<>();
            empExportMap.put("title", empExportParams);
            empExportMap.put("entity", DeptUtil.class);
            empExportMap.put("data", exportList);
            // 将sheet1、sheet2使用得map进行包装
            List<Map<String, Object>> sheetsList = new ArrayList<>();
            sheetsList.add(deptExportMap);
            sheetsList.add(empExportMap);
            // 执行方法
            workBook = ExcelExportUtil.exportExcel(sheetsList,
ExcelType.HSSF);
```

   - 需要定义 `List<Map<String, Object>>` 对象用于接收多个sheet的数据，然后使用 `ExcelExportUtil.exportExcel(sheetsList, ExcelType.HSSF);` 进行导出。