```
if (level < lastLevel) { // 非最后一层需要与下一层进行合并
    List<FileMeta> basicFiles = new ArrayList<>();
    for (FileMeta fileMeta : fileMetaList) {
        if (fileMeta.getLevel() == level) {
            basicFiles.add(fileMeta);
        }
    }

    if (level == 0) { // 第 0 层的数据必须先对旧的数据（fileId 小的）做 compaction
        if (basicFiles.size() > levelMinSize[0]) {
            basicFiles.sort(Comparator.naturalOrder());
            upperLevelFiles = basicFiles.subList(0, compactionMaxPickCount[0]);
        }
    } else { // 第 1、2 层则优先选择尺寸小的文件，减少零碎
        … //省略
    }

    if (upperLevelFiles != null) { //
        byte[] leftEnd = CommonUtils.smallestStartKey(upperLevelFiles);
        byte[] rightEnd = CommonUtils.largestEndKey(upperLevelFiles);
        //跟据上层文件的记录 rk 范围选取下层文件
        lowerLevelFiles = new ArrayList<>();
        for (FileMeta fileMeta : fileMetaList) {
            if (fileMeta.getLevel() == level + 1 && CommonUtils.overlapInRange(fileMeta, leftEnd,
rightEnd)) {
                lowerLevelFiles.add(fileMeta);
            }
        }
        lowerLevelFiles.sort(Comparator.naturalOrder());
    }
} else { // 最后一层与自己合并
    … //省略
}

//使用 upperLevelFiles 与 lowerLevelFiles 调度任务
if (upperLevelFiles != null && !upperLevelFiles.isEmpty()) {
    LOG.info("Scheduled a compaction task on level " + level);
    compactionExecutorV1.doCompactionV1(upperLevelFiles, lowerLevelFiles, level);
}
```