✅ 如何处理 Kafka 消费者遭遇的 Offset 问题?

扩展与回答：

Kafka 的 Offset 问题一般包括以下几种情况：

1. **重复 offset**：可通过配置 `enable.auto.commit=false`，在处理完消息后**手动提交 offset** 保证“至少一次”语义。
2. **offset 丢失**：使用消费者组的 `group.id` 配置确保 offset 被 Kafka 自动管理或保存在外部（如数据库）。
3. **offset 被清除（过期）**：消费者需要指定 `auto.offset.reset=earliest` 或 `latest` 来决定是从头读还是跳过。

✅ 当在一个分支上工作，但突然需要切换到另一个分支...

解决办法：

1. **使用 Git Stash 暂存修改：**

```
git stash
git checkout other-branch
```

2. **修改完切回原分支并恢复修改：**

```
git checkout original-branch
git stash pop
```

也可用 `git stash -u` 暂存包括未追踪的文件。

✅ 取消暂停容器中所有的进程的命令：

```
kill -CONT -1
```

说明：

- `-CONT` 信号用于恢复被 `SIGSTOP` 暂停的进程。
- `-1` 表示对容器内所有进程发送该信号。

也可使用 `docker container unpause <container_id>` 取消暂停整个容器。

✅ 从 "file.txt" 中查找 "error" 开头的所有行：

使用 `grep` 命令：

```
grep "^error" file.txt
```

说明：

- `^` 表示行首，确保只匹配以 `error` 开头的行。

✅ 查看与关键词 "httpd" 相关的所有进程：

```
ps aux | grep httpd
```

或者使用更精确匹配（排除 grep 本身）：

```
pgrep -af httpd
```

✅ 将当前目录中所有 ".txt" 文件扩展名更改为 ".md"：

```
for file in *.txt; do mv "$file" "${file%.txt}.md"; done
```

✅ 设计一个数据表来存储学生的成绩信息：

扩展与回答：

可以设计如下表结构：

```
CREATE TABLE student_scores (
    id INT PRIMARY KEY AUTO_INCREMENT,
    student_id INT NOT NULL,
    course_id INT NOT NULL,
    score DECIMAL(5,2) NOT NULL,
    exam_date DATE,
    FOREIGN KEY (student_id) REFERENCES students(id),
    FOREIGN KEY (course_id) REFERENCES courses(id)
);
```

表结构包含：学生 ID、课程 ID、分数、考试日期，可支持多次考试记录。

✅ 数据库查询优化方法的应用：

1. 创建索引（特别是联合索引）
2. 避免 `SELECT *`
3. 使用 LIMIT 分页
4. 减少子查询和 JOIN 层数
5. 使用 EXPLAIN 分析 SQL 执行计划
6. 分区分表策略（大数据量）

✅ Kafka 在建筑监控系统中出现的超时异常如何解决?

可能原因与处理方式:

1. **消费端处理慢** → 优化业务逻辑，增加消费者并发。
2. **Kafka 服务响应慢** → 增加 partition、broker，优化磁盘性能。
3. **配置不合理** → 如 `request.timeout.ms` 太短，可调整。

✅ 优化 SQL 查询以提高电商平台商品搜索性能:

- 建立商品名称、分类、品牌等字段索引
- 引入全文索引（如使用 MySQL 的 `FULLTEXT` 或接入 Elasticsearch）
- 使用缓存（如 Redis）缓存热搜商品
- 减少 JOIN、聚合，采用异步统计

✅ 根据应用场景设计索引以优化 MySQL 查询性能:

- 精确查询使用 B+ 树索引
- 范围查询放在索引末尾
- 多条件查询考虑联合索引顺序
- 高频更新字段慎用索引
- 查询分析 `EXPLAIN` 结果命中率

✅ Kubernetes 中如何回滚 Deployment:

```
kubectl rollout undo deployment <deployment-name>
```

可以加上 `--to-revision=n` 回滚到指定版本。

✅ Golang 代码段输出结果分析:

文档未给出代码内容，请补充代码段以便分析输出。

✅ 查看正在运行的后台进程命令:

```
jobs
```

查看后台任务。
或者:

```
ps -ef
```

查看所有后台进程。

✅ 限制 goroutine 的并发执行数量：

可以使用带缓冲的 channel 作为并发控制器：

```
semaphore := make(chan struct{}, 10) // 限制最大10个并发

for _, task := range tasks {
    semaphore <- struct{}{}
    go func(task Task) {
        defer func() { <-semaphore }()
        doWork(task)
    }(task)
}
```

✅ 查询 2023 年 8 月销售额最高的三个商品：

```
SELECT product_id, SUM(sale_amount) AS total
FROM sales
WHERE sale_date BETWEEN '2023-08-01' AND '2023-08-31'
GROUP BY product_id
ORDER BY total DESC
LIMIT 3;
```

✅ Git 变基操作中如何解决冲突？

1. `git rebase branch`
2. 若出现冲突，手动修改冲突文件
3. 使用 `git add <file>` 标记已解决
4. `git rebase --continue`
5. 若想放弃变基：`git rebase --abort`

✅ 在交互式 rebase 中删除某个提交：

```
git rebase -i HEAD~n
```

将需要删除的提交前的 `pick` 改为 `drop` 或直接删除该行。

✅ 当前分支 rebase 到远程 master：

```
git fetch origin
git rebase origin/master
```