链滴

关于 Java Web 项目性能提升的一些思路

作者：huihui

原文链接：https://ld246.com/article/1487946252902

来源网站：链滴

- **使用 Nginx 作为前端接入**

用 Nginx 进行动静分离。这个不用多讲，新浪、网易、淘宝、腾讯等巨头的使用已经说明了一切。

- **保持最简单的架构**

遵守 KISS 原则(Keep it simple and stupid)。尽量不要考虑项目外的重用。过多的考虑项目外的重，必然会增加项目的复杂度。避免过度集成，让每个模块只做自己的事，这对于日后的维护和模块复都有好处。

- **精心设计缓存处理、毫不吝啬代码(对象、列表、片段)**

对于门户网站的首页来说，往往可能会有近百个 SQL。用户并发上去以后，光首页就足以让服务器 dwn 掉。缓存不但有利于降低负载，而且还能提高响应速度。

- **调整使用聚集索引**

对于每个表来讲，聚集索引只有一个，利用好了，查询速度会有意想不到的提升效果。
以 MySql 为例，InnoDB选取聚集索引参照列的顺序是
1\. 如果声声明了主键(primary key)，则这个列会被做为聚集索引；
2\. 如果没有声明主键，则会用一个唯一且不为空的索引列做为主键，成为此表的聚集索引；
3\. 上面二个条件都不满足，InnoDB会自己产生一个虚拟的聚集索引。

```
1. CREATE TABLE     timeline_raw (
2. rawId bigint(20) NOT NULL AUTO_INCREMENT,
3. uid bigint(20) DEFAULT NULL,
4. did bigint(20) DEFAULT NULL,
5. channelId char(1) NOT NULL DEFAULT '1' COMMENT '1:qvga; 2:720p',
6. fileId bigint(20) DEFAULT NULL,
7. sectionId bigint(20) DEFAULT NULL,
8. headerFilePath varchar(120) DEFAULT NULL,
9. startTime bigint(20) DEFAULT NULL,
10. endTime bigint(20) DEFAULT NULL,
11. updateTime datetime DEFAULT NULL,
12. createTime datetime DEFAULT NULL,
13. PRIMARY KEY (      rawId),
14. KEY       index_uid_did_startTime (uid,did,startTime) USING BTREE,
15. KEY       index_uid_did_endTime (uid,did,endTime) USING BTREE,
16. KEY       index_time (startTime) USING BTREE,
17. KEY       index_uid_did_fileId (uid,did,sectionId) USING BTREE,
18. KEY       index_sectionId (sectionId)
19. ) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

这个表有四个索引：主键 rawId、sectionId、uid,did、startTime。

项目的 iBatis2 中有这样一条查询语句：

```
1. <selectid="getRawFileList" parameterClass="java.util.HashMap" resultClass="com.defonds
mysql.raw.entity.TimelineRaw">
2.
 SELECT * FROM timeline_raw_
3.
 WHERE uid=#uid#
4.
 AND did=#did#
5.
 AND channelId=#channelId#
6.
 < isNotNull  property="sectionId"> AND sectionId = #sectionId# < isNotNull>
7.
 AND
8.
 (
9.
        (startTime BETWEEN #startTime# and #endTime#)
10.
            OR
11.
        (endTime BETWEEN #startTime# and #endTime#)
12.
                        OR
13.
```

```
    (
14.
15.

    startTime<=#startTime#
16.

     ]]>
17.

    AND
18.
19.

    endTime>=#endTime#
20.

    ]]>
21.

    )
22.

 )
23.

 ORDER BY startTime;
24. select>
```

根据实际业务向 timeline_raw 表注入一千万条数据，进行模拟测试，发现 getRawFileList 的执行平时间为 160 ms 以上。这是不能接受的。

考虑到实际业务中对于主键 rawId 查询条件甚少，我们把rawId主键索引取消掉，改为唯一约束，却把ectionId+startTime+endTime作为主键(业务上能够保证其唯一性，根据InnoDB索引规则，这个索将成为我们新表的聚集索引)。然后把sectionId、startTime两个索引也取消掉，仅保留uid,did索引。

这样子，我们新表的索引实际上只有两个了：一个聚集索引(sectionId+startTime+endTime)一个非集索引(uid,did)。

再次进行模拟测试，同样的数据、数据量，同样的查询结果集，getRawFileList 执行平均时间已经降了 11 ms。结果是令人振奋的，不是么?

- **使用 /dev/shm 来存储缓存的磁盘文件**

在网站运维中，利用好了这一点，往往有意想不到的收获。以 tomcat 为例，可以通过修改 catalina.s 中的 CATALINA_TMPDIR 值的路径来将缓存设置为 /dev/shm。
以 OSC 为例，他们就是纯 Java 写的，部署在 tomcat 下。在长时间的在线运行之后，管理员发现网响应速度奇慢，服务器负载正常，又找不出是哪里的问题。后来 df 一下，发现 tomcat 临时目录下文件足足有 8G 之多，原来是 CPU 等待磁盘操作造成响应速度加长。于是他们将临时目录映射到 /de /shm，网站响应速度从此奇快。

- **分析系统中每一个 SQL 的执行效率**

以 MySql 为例，对于每个 SQL 最好都 explain 一下。对于有明显效率问题的，通过 sql 优化、调索等方法进行改进。

- **健康慢查询日志，检查所有执行超过 100 毫秒的 SQL**

对于上线了的项目，健康慢查询日志，检查所有执行超过 100 毫秒的 SQL，看看有没有优化余地。于没有上线的项目，可以进行场景模拟对嫌疑 SQL，或者对频繁使用的 SQL 进行性能测试，统计它执行时间，得出平均值，画出曲线分析图，对于单表千万数据，执行时间超过 50ms 的 SQL 要重点注。参考《[sql 性能测试例子](http://blog.csdn.net/defonds/article/details/16832081)》。