1. TCP与UDP的区别及使用场景

**区别：**

* **TCP（传输控制协议）**：面向连接，提供可靠的数据传输，保证数据包的顺序和完整性，适合对数据准确性要求高的场景。
* **UDP（用户数据报协议）**：无连接，传输速度快但不保证数据包的顺序和完整性，适合对实时性要求高的场景。

**使用场景：**

* **TCP**：网页浏览、文件传输、邮件发送等。
* **UDP**：视频流、在线游戏、语音通话等。

2. ArrayList与LinkedList的区别

**区别：**

* **ArrayList**：基于动态数组，随机访问速度快，但插入和删除元素较慢。
* **LinkedList**：基于双向链表，插入和删除元素快，但随机访问速度慢。

**使用场景：**

* **ArrayList**：频繁随机访问时使用。
* **LinkedList**：频繁插入和删除时使用。

3. SSH协议

**SSH（Secure Shell）**：用于安全远程登录和其他网络服务的协议，通过加密保护数据传输，常用于远程服务器管理和文件传输。

4. 正常环境如何保证数据库缓存数据一致性

**方法：**

* **读写策略**：使用读写分离或缓存更新策略。
* **失效机制**：设置缓存失效时间或手动删除缓存。
* **事务处理**：通过数据库事务确保数据一致性。

5. 高并发环境数据库缓存一致性

**方法：**

* **分布式锁**：防止并发写操作。
* **消息队列**：异步处理缓存更新。
* **一致性哈希**：优化缓存分布，减少数据不一致。

6. ThreadLocal

**ThreadLocal**：为每个线程提供独立的变量副本，避免线程间共享变量的并发问题，常用于线程上下文信息传递。

7. 项目难点及两次追问

**难点**：高并发系统的性能优化。

**追问1**：如何优化数据库查询？

* 使用索引、优化SQL语句、分库分表等。

**追问2**：如何处理缓存雪崩？

* 设置不同的缓存失效时间、使用多级缓存等。

8. MySQL查询分析工具的使用及追问

**工具**：EXPLAIN用于分析SQL语句的执行计划。

**追问**：如何优化EXPLAIN显示的全表扫描？

* 添加索引、优化查询条件等。

9. 三年职业规划

**目标**：

* **第一年**：深入掌握当前技术栈，提升项目经验。
* **第二年**：学习新技术，承担更多责任。
* **第三年**：成为技术专家或团队负责人，参与架构设计和技术决策。

TCP重传机制有哪些触发条件？超时重传和快速重传的区别？

TCP 重传机制的触发条件主要有以下两种：

**超时重传**：发送方在发送数据后，会启动一个定时器。如果在定时器超时之前没有收到接收方对该数据的确认（ACK），就会认为数据丢失或损坏，从而触发重传。

1. 由定时器超时触发，即经过了一定的时间间隔仍未收到确认
2. 通常等待时间较长，要等到定时器超时才进行重传，这可能导致较长的延迟
3. 一般会重传定时器超时对应的整个数据包或数据包组

**快速重传**：当接收方收到乱序的数据包时，会发送重复的确认给发送方。如果发送方连续收到多个（通常是 3 个）相同的重复确认，就会认为该确认号之后的数据可能丢失了，进而触发快速重传，而不需要等待超时。

1. 由收到多个重复确认触发，不需要等待定时器超时。
2. 能在较早阶段就发现数据丢失并进行重传，能更快地恢复丢失的数据，减少传输延迟。
3. 通常只重传收到重复确认所指示的丢失数据包，而不是重传整个窗口或更大范围的数据，更有针对性。

解释Linux管道的概念，给出一个使用管道的例子

在 Linux 中，管道（Pipe）是一种进程间通信（IPC）机制，它使用竖线 | 作为操作符。管道的作用是把一个命令的输出直接作为另一个命令的输入，这样就能让多个命令协同工作，完成复杂的任务。

举例：ls | grep ".txt$" | wc -l

ls 命令的输出被当作 grep 命令的输入，而 grep 命令的输出又成为了 wc -l 命令的输入，最终得出 .txt 文件的数量。

* **ls 命令**：ls 是 “list” 的缩写，用于列出指定目录中的文件和子目录。当不指定目录时，默认列出当前工作目录的内容。例如，在终端输入 ls 后，会显示当前目录下的所有文件和文件夹名称。此外，它还支持很多选项，如 ls -l 可以以长格式显示文件和目录的详细信息，包括权限、所有者、大小、修改时间等。
* **grep 命令**：grep 是 “Global Regular Expression Print” 的缩写，用于在文本中搜索匹配指定模式的行，并将匹配的行打印出来。在 grep ".txt$" 中，".txt$" 是一个正则表达式，其中 "txt" 代表要匹配的文本内容，$ 是正则表达式的元字符，表示行的结尾，所以 ".txt$" 表示匹配以 .txt 结尾的行。因此，grep ".txt$" 会从输入中筛选出以 .txt 结尾的行。
* **wc -l 命令**：wc 是 “word count” 的缩写，用于统计文件的行数、单词数和字节数。-l 是 wc 命令的一个选项，它表示只统计行数。因此，wc -l 会对输入的文本进行处理，输出其中的行数。

对explain关键字的理解，包括它的用途如何使用典型输出信息的含义，以及如何利用这些信息进行查询优化

对EXPLAIN关键字的理解

EXPLAIN 用于分析查询语句的执行计划，让开发者了解数据库如何执行查询，帮助找出查询性能瓶颈。

使用方法

* 定位潜在性能问题，为优化查询提供依据。
* **使用方法**： EXPLAIN SELECT \* FROM users WHERE age > 20;。

典型输出信息含义

type

连接类型，反映了查询查找数据的方式，是衡量查询效率的重要指标。不同的 type 值对应的查询效率差异很大，从优到劣的常见类型排序为：system > const > eq\_ref > ref > range > index > ALL。

* ALL 代表全表扫描，意味着数据库要遍历表中的每一行数据，查询效率最差。
* range 表示使用索引进行范围扫描，效率较好。
* const 则表示通过索引一次就能找到数据，效率最高。

key

实际使用的索引。索引在数据库查询中起着关键作用，能够显著提升查询速度。若 key 显示为 NULL，则表明查询未使用索引，可能会引发全表扫描，导致查询效率低下。因此，要保证查询语句能利用合适的索引，可通过 CREATE INDEX 语句创建索引，或对查询语句进行调整，以让其使用已有的索引。

rows

它是数据库估算的查询需要扫描的行数。该值并非精确值，但能反映查询的大致成本。一般来说，rows 值越小，查询所需扫描的数据量就越少，查询效率也就越高。你可以借助优化查询条件、添加合适的索引等方法来减少 rows 值。

* **id**：查询的序列号，反映查询中各个子查询或操作的执行顺序，数值越大越先执行。
* **select\_type**：查询类型，常见有 SIMPLE（简单查询）、PRIMARY（主查询）、SUBQUERY（子查询）等。
* **table**：查询涉及的表名。
* **type**：连接类型，反映查询的效率，从好到差常见有 system、const、eq\_ref、ref、range、index、ALL 等，ALL 表示全表扫描，效率最低。
* **possible\_keys**：可能使用的索引。
* **key**：实际使用的索引。
* **key\_len**：使用的索引长度，越短通常效率越高。
* **ref**：哪些列或常量被用于查找索引列上的值。
* **rows**：估算查询需要扫描的行数，值越小性能越好。
* **Extra**：额外信息，如 Using index 表示使用了覆盖索引，Using where 表示使用了 WHERE 子句过滤数据。

利用信息进行查询优化

* **索引优化**：若 possible\_keys 有索引但 key 为 NULL，考虑创建合适的索引；若 type 为 ALL，可尝试添加索引提高查询效率。
* **减少扫描行数**：通过 rows 估算值，优化查询条件，缩小扫描范围。
* **利用覆盖索引**：若 Extra 未显示 Using index，可调整查询语句或索引，让查询仅通过索引就能获取所需数据。

自动装箱和拆箱？给出一个例子

* **自动装箱**：在 Java 里，自动装箱是把基本数据类型自动转换为**对应的**包装类对象。例如，将 int 类型转换为 Integer 对象。
* **自动拆箱**：与自动装箱相反，自动拆箱是将包装类对象自动转换为对应的基本数据类型。例如，把 Integer 对象转换为 int 类型。

hashCode和equals的关系？为什么要重写hashcode和equals

hashCode 和 equals 的关系

在 Java 里，hashCode 和 equals 方法有着紧密联系，需遵循以下规则：

* 若两个对象 equals 比较结果为 true，那么它们的 hashCode 值必须相同。
* 若两个对象的 hashCode 值相同，它们的 equals 比较结果不一定为 true，即可能是不同对象产生了哈希冲突。

重写 hashCode 和 equals 的原因

* **equals 方法**：默认的 equals 方法比较的是对象的引用地址，也就是判断两个对象是否为同一个实例。但在很多情况下，我们希望比较的是对象的内容是否相等，所以需要重写 equals 方法来实现自定义的相等逻辑。
* **hashCode 方法**：在使用 HashMap、HashSet 等基于哈希表实现的集合类时，这些类会先通过 hashCode 值来确定对象在哈希表中的位置，再用 equals 方法判断对象是否相等。如果只重写了 equals 方法而不重写 hashCode 方法，就可能出现两个 equals 比较为 true 的对象，其 hashCode 值却不同，从而导致在集合中出现重复元素或者无法正确查找元素的问题。所以，当重写 equals 方法时，通常也需要重写 hashCode 方法，以保证它们的一致性。

解释Java中的线程，如何创建和启动一个线程

**线程**是Java中实现并发编程的基本单元。线程允许程序同时执行多个任务，从而提高效率。每个线程都有自己的执行路径，可以独立运行。

**创建和启动线程的两种主要方式：**

* **继承Thread类：**
* **实现Runnable接口：**

设计一个简单的文章热度计算系统考虑浏览量评论数和分享数等因素

**设计思路：**

文章热度可以通过浏览量、评论数和分享数等指标计算。假设热度公式为： 热度=浏览量×0.5+评论数×1.0+分享数×1.5热度=浏览量×0.5+评论数×1.0+分享数×1.5

追问：是否考虑过时间因素对文章热度的影响？比如，一篇文章中发布后的不同时间段内热度的变化情况，如果考虑的话，你会如何将时间因素整合到热度计算中呢？

**考虑时间因素：**

文章的热度会随时间衰减，可以通过指数衰减模型整合时间因素。公式为： 热度=(浏览量×0.5+评论数×1.0+分享数×1.5)×e−λt热度=(浏览量×0.5+评论数×1.0+分享数×1.5)×*e*−*λt*

其中，λ*λ* 是衰减率，t*t* 是文章发布后的时间。

在编程学习中如何处理挫折感和持续保持学习动力的？举一个具体的例子

* **设定小目标：** 将大任务分解为小目标，逐步完成。
* **寻求帮助：** 通过社区、论坛或向有经验的人请教。
* **记录进展：** 记录学习过程，看到进步时会有成就感。
* **保持兴趣：** 选择感兴趣的项目，通过实践学习。

1. TCP重传机制，超时重传和快速重传区别

* **超时重传（Timeout Retransmission）**：当发送方发送一个数据包后，启动一个计时器。如果在计时器超时前未收到接收方的确认（ACK），发送方会重新发送该数据包。超时重传的时间通常基于RTT（Round-Trip Time）动态调整。
* **快速重传（Fast Retransmit）**：当接收方收到一个失序的数据包时，会立即发送一个重复的ACK。如果发送方连续收到三个相同的ACK，它会认为该数据包丢失，并立即重传该数据包，而不等待超时。

**区别**：

* 超时重传依赖于计时器超时，而快速重传依赖于重复ACK。
* 快速重传比超时重传更快，因为它不需要等待计时器超时。

2. Linux中管道的概念，使用场景

* **概念**：管道是一种进程间通信（IPC）机制，允许一个进程的输出直接作为另一个进程的输入。管道分为匿名管道和命名管道。
  + **匿名管道**：用于父子进程之间的通信，使用|符号创建。
  + **命名管道（FIFO）**：可以在无关进程之间使用，通过mkfifo命令创建。
* **使用场景**：
  + 命令行中组合多个命令，例如ls | grep .txt。
  + 进程间传递数据，特别是在脚本中。

3. EXPLAIN的作用以及有哪些字段

* **作用**：EXPLAIN用于分析SQL查询的执行计划，帮助开发者理解查询是如何执行的，从而优化查询性能。
* **常见字段**：
  + id：查询的标识符。
  + select\_type：查询的类型（如SIMPLE、SUBQUERY等）。
  + table：查询涉及的表。
  + type：访问类型（如ALL、INDEX、RANGE等）。
  + possible\_keys：可能使用的索引。
  + key：实际使用的索引。
  + rows：预计扫描的行数。
  + Extra：额外信息（如Using where、Using index等）。

6. 如何设计校园论坛帖子点赞接口的防刷机制？追问：如何防止误伤正常用户

* **防刷机制**：
  + **频率限制**：限制每个用户在一定时间内的点赞次数。
  + **IP限制**：限制同一IP地址的点赞频率。
  + **行为分析**：通过机器学习或规则引擎识别异常行为。
  + **验证码**：在频繁操作时要求用户输入验证码。
* **防止误伤正常用户**：
  + **动态调整阈值**：根据用户历史行为动态调整限制阈值。
  + **白名单机制**：将可信用户加入白名单，不受限制。
  + **用户反馈**：提供申诉渠道，允许用户反馈误判情况。

7. 实习或项目中，快速学习的情况如何处理？具体描述一个案例

* **案例**：在一次实习中，我需要快速学习并使用一个全新的框架来完成项目。
* **处理方式**：
  + **查阅文档**：首先阅读官方文档，了解框架的基本概念和使用方法。
  + **实践项目**：通过一个小型项目进行实践，边学边做。
  + **请教同事**：遇到问题时，及时向有经验的同事请教。
  + **总结笔记**：将学习过程中的关键点记录下来，便于复习和分享。

8. 高性能服务器中如何部署协程调度器，详细描述策略

* **策略**：
  + **轻量级协程**：使用轻量级的协程库（如libco、Boost.Coroutine）来减少上下文切换开销。
  + **多核调度**：将协程调度器与CPU核心绑定，利用多核优势。
  + **事件驱动**：基于事件驱动的模型，使用epoll或kqueue等机制高效处理I/O事件。
  + **负载均衡**：动态分配协程到不同的核心，确保负载均衡。
  + **优先级调度**：根据任务优先级调度协程，确保高优先级任务优先执行。

9. 如何快速识别项目中的关键问题，如何分析重点原因？追问：如何系统验证可靠性，采用什么方法确定不会再出现问题

* **识别关键问题**：
  + **日志分析**：通过日志快速定位异常。
  + **监控系统**：使用监控工具（如Prometheus、Grafana）实时跟踪系统状态。
  + **用户反馈**：收集用户反馈，识别常见问题。
* **分析重点原因**：
  + **根因分析**：使用5 Whys或鱼骨图等工具深入分析问题的根本原因。
  + **代码审查**：检查相关代码，寻找潜在问题。
* **系统验证可靠性**：
  + **单元测试**：编写单元测试，确保代码的正确性。
  + **集成测试**：进行集成测试，验证模块间的交互。
  + **回归测试**：在修复问题后，进行回归测试，确保问题不再出现。

10. 压测时是否设计测试用例，边界情况问题？

* **设计测试用例**：是的，压测时需要设计覆盖各种场景的测试用例，包括正常情况、边界情况和异常情况。
* **边界情况**：特别关注系统在极限条件下的表现，例如高并发、大数据量、网络延迟等。

2. Linux中文件、文件夹复制，以及权限控制

* **文件复制**：
  + 使用cp命令复制文件。例如：cp source.txt destination.txt。
  + 如果需要保留文件属性（如权限、时间戳），可以使用-p选项：cp -p source.txt destination.txt。
* **文件夹复制**：
  + 使用cp -r命令递归复制文件夹及其内容。例如：cp -r sourcedir/ destinationdir/。
* **权限控制**：
  + 使用chmod命令修改文件或文件夹的权限。例如：chmod 755 file.txt（设置文件权限为rwxr-xr-x）。
  + 使用chown命令修改文件或文件夹的所有者和组。例如：chown user:group file.txt。

3. 什么是聚合函数，举例

* **聚合函数**：聚合函数用于对一组值执行计算，并返回单个值。常见的聚合函数包括：
  + COUNT()：计算行数。例如：SELECT COUNT(\*) FROM employees;。
  + SUM()：计算数值列的总和。例如：SELECT SUM(salary) FROM employees;。
  + AVG()：计算数值列的平均值。例如：SELECT AVG(salary) FROM employees;。
  + MIN()：找到列中的最小值。例如：SELECT MIN(salary) FROM employees;。
  + MAX()：找到列中的最大值。例如：SELECT MAX(salary) FROM employees;。

4. try catch finally 执行顺序

* **执行顺序**：
  + **try**：首先执行try块中的代码。
  + **catch**：如果try块中抛出异常，则执行相应的catch块。
  + **finally**：无论是否抛出异常，finally块中的代码都会执行。

5. finally一定会执行吗

* **一般情况下**：finally块中的代码在大多数情况下都会执行，无论try块中是否抛出异常。
* **特殊情况**：
  + **程序退出**：如果try或catch块中调用System.exit()导致程序退出，finally块不会执行。
  + **无限循环**：如果try或catch块中进入无限循环，finally块不会执行。
  + **系统崩溃**：如果系统崩溃或断电，finally块也不会执行。

1. HTTPS和HTTP的区别

| **特性** | **HTTP** | **HTTPS** |
| --- | --- | --- |
| **安全性** | 不加密，数据明文传输 | 加密，数据通过SSL/TLS加密传输 |
| **端口** | 默认端口80 | 默认端口443 |
| **性能** | 较快，无加密开销 | 较慢，加密解密过程增加开销 |
| **证书** | 不需要证书 | 需要SSL/TLS证书 |

2. String、StringBuilder、StringBuffer的区别

* **String**：不可变，适合存储固定字符串。
* **StringBuilder**：可变，性能高，适合单线程场景。
* **StringBuffer**：可变，线程安全，适合多线程场景。

3. Linux管道，举例

**管道（Pipe）**：管道是Linux中用于进程间通信的一种机制，将一个命令的输出作为另一个命令的输入。使用符号|表示。

**举例**：

1. **统计文件行数**：

cat file.txt | wc -l

* + cat file.txt：输出文件内容。
  + wc -l：统计输入的行数。

1. **查找特定进程**：

ps aux | grep nginx

* + ps aux：列出所有进程。
  + grep nginx：筛选包含“nginx”的行。

1. **排序并去重**：

cat file.txt | sort | uniq

* + cat file.txt：输出文件内容。
  + sort：对内容排序。
  + uniq：去除重复行。

1. **统计日志中错误次数**：

grep "ERROR" app.log | wc -l

* + grep "ERROR" app.log：筛选包含“ERROR”的行。
  + wc -l：统计行数。

HTTP状态码301与302区别

| **特性** | **301 Moved Permanently** | **302 Found (Moved Temporarily)** |
| --- | --- | --- |
| **含义** | 永久重定向 | 临时重定向 |
| **浏览器行为** | 缓存重定向结果，后续直接访问新URL | 不缓存，每次请求原URL都会重定向 |
| **SEO影响** | 搜索引擎将权重转移到新URL | 搜索引擎保留原URL的权重 |
| **使用场景** | 网站永久迁移 | 临时维护或A/B测试 |

**查看系统的日志文件，日志文件有哪些**

* **Linux系统常见日志文件**：
  + /var/log/messages：系统全局日志。
  + /var/log/syslog：系统日志。
  + /var/log/auth.log：认证日志（如登录、sudo）。
  + /var/log/kern.log：内核日志。
  + /var/log/dmesg：启动日志。
  + /var/log/nginx/access.log：Nginx访问日志。
  + /var/log/mysql/error.log：MySQL错误日志。
* **查看方法**：
  + tail -f /var/log/syslog：实时查看日志。
  + cat /var/log/messages：查看完整日志。

关系型数据库与非关系型数据库的区别

| **特性** | **关系型数据库（RDBMS）** | **非关系型数据库（NoSQL）** |
| --- | --- | --- |
| **数据模型** | 表结构，行列形式 | 文档、键值对、图等 |
| **扩展性** | 垂直扩展 | 水平扩展 |
| **事务支持** | ACID事务 | 部分支持，最终一致性 |
| **查询语言** | SQL | 无统一查询语言 |
| **性能** | 适合复杂查询 | 适合高并发、大规模数据 |
| **典型代表** | MySQL、PostgreSQL | MongoDB、Redis、Cassandra |

Java封装的目的，为什么要封装

* **目的**：
  + **隐藏实现细节**：保护内部数据不被外部直接访问。
  + **提高安全性**：通过方法控制数据的访问和修改。
  + **增强可维护性**：修改内部实现不影响外部调用。
  + **简化使用**：提供清晰的接口，降低使用复杂度。
* **示例**：

Java

public class Person {

private String name; *// 私有属性*

public String getName() { *// 公共方法*

return name;

}

public void setName(String name) { *// 公共方法*

this.name = name;

}

}

如何避免深度分页的性能问题

* **问题**：深度分页（如LIMIT 100000, 10）会导致数据库扫描大量数据，性能下降。
* **解决方案**：
  + **基于游标的分页**：使用WHERE id > last\_id LIMIT 10代替LIMIT offset, 10。
  + **索引优化**：确保分页字段有索引。
  + **缓存分页结果**：将热门页面的结果缓存。
  + **分批加载**：前端分批请求数据，减少单次查询量。

如何提高自己的代码质量和编程技巧

1. **代码规范**：遵循编码规范。
2. **代码复用**：提取公共方法，减少重复代码。
3. **单元测试**：编写单元测试，确保代码正确性。
4. **设计模式**：学习并应用设计模式。
5. **Code Review**：参与代码评审，学习他人经验。
6. **持续学习**：阅读技术书籍、博客，关注新技术。
7. **工具使用**：使用静态分析工具（如SonarQube）检查代码质量。

1. 网络延迟包含的部分

1. **传输延迟**：数据从发送方到接收方的传输时间。
2. **处理延迟**：路由器、交换机等设备的处理时间。
3. **排队延迟**：数据在网络设备中排队等待的时间。
4. **传播延迟**：信号在介质中传播的时间。
5. **序列化延迟**：数据转换为网络格式的时间。

2. Java函数重载和函数重写

| **特性** | **函数重载（Overload）** | **函数重写（Override）** |
| --- | --- | --- |
| **定义** | 同一类中方法名相同，参数不同 | 子类中方法名、参数与父类相同 |
| **调用** | 编译时决定 | 运行时决定 |
| **访问修饰符** | 可以不同 | 不能比父类更严格 |
| **返回值** | 可以不同 | 必须相同 |

3. Linux查看内存占用情况

* **命令**：
  + free -h：查看内存使用情况。
  + top：实时查看内存占用。
  + vmstat：查看虚拟内存统计。
  + ps aux --sort=-%mem：按内存占用排序进程。

4. Java垃圾回收

* **机制**：自动回收不再使用的对象，释放内存。
* **算法**：
  + **标记-清除**：标记无用对象，清除。
  + **复制算法**：将存活对象复制到新空间。
  + **标记-整理**：标记无用对象，整理内存。
  + **分代收集**：将堆分为新生代和老年代，分别回收。

5. 介绍一下索引和优缺点

* **索引**：数据结构，用于加速数据库查询。
* **优点**：
  + 加速查询。
  + 提高数据检索效率。
* **缺点**：
  + 占用存储空间。
  + 降低写操作（插入、更新、删除）性能。

6. 高并发的计数器怎么设计

* **方案**：
  + **分布式缓存**：使用Redis等缓存系统实现原子操作。
  + **分片计数**：将计数器分片，分散压力。
  + **异步更新**：将计数操作异步化，减少实时压力。
* **数据一致性**：通过分布式锁或CAS操作保证一致性。
* **性能优化**：使用批量提交和缓存减少数据库压力。

1. 遇到的挫折和学习动力

* **挫折**：在项目中遇到技术瓶颈，无法解决。
* **学习动力**：通过查阅文档、请教同事、实践解决问题，提升技术能力。

2. 遇到不会的技术领域问题怎么解决

1. **查阅文档**：阅读官方文档或技术博客。
2. **搜索解决方案**：使用Google、Stack Overflow等平台。
3. **请教同事或社区**：向有经验的人请教。
4. **实践验证**：通过小项目验证学习成果。

3. 未来三年的职业规划

1. **短期（1年）**：深入学习当前技术栈，提升开发效率。
2. **中期（2年）**：参与架构设计，积累项目经验。
3. **长期（3年）**：成为技术专家或团队负责人。

DNS是什么？网络通信中作用

* **DNS**：域名系统（Domain Name System），将域名转换为IP地址。
* **作用**：简化用户访问，无需记住复杂的IP地址。

设计一个高效的缓存系统？要求数据一致性和故障恢复

* **设计要点**：
  + **缓存策略**：使用LRU或LFU算法淘汰数据。
  + **数据一致性**：通过写穿透或写回策略保证一致性。
  + **故障恢复**：定期备份缓存数据，使用持久化存储（如Redis AOF）。

平常是怎么学习新技术的

1. **阅读文档**：学习官方文档。
2. **实践项目**：通过小项目应用新技术。
3. **技术社区**：参与技术论坛或开源项目。
4. **总结分享**：撰写博客或技术文章。