

现在已经到了面试招聘比较火热的时候，后续会分享一些面试真题供大家复习参考。准备面试的过程中，一定要多看面经，多自测！

今天分享一位四川大学的同学分享的阿里淘天一面的面经。

淘天一面的问题比较随意，主要分为下面几类问题：

1. 非技术问题比如自我介绍。
2. 笔试题比如手写单例模式。
3. 技术问题也就是常规八股比如 GET 和 POST 的区别。

淘天一面的笔试题共有三道，不是那种纯粹的 Leetcode 问题，我个人觉得笔试相对比较简单了。



我要当卷王

大橘已定

08-25 11:17 美团_到家事业群/到家研发平台_后端(实... N

+ 关注

淘天一面，秒挂

电话面,一个半小时

1. 自我介绍
2. 讲一下实习经历，以及遇到的难点
3. 说一下自己以后的发展发向(就像更倾向于技术还是业务理解)
4. 进入笔试环节
 - a. 写三种单例模式的实现方式(只知道饿汉式和懒汉式，懒汉式写了线程安全和不安全两种，面试官只给算一种)
 - b. 编号为1-n的循环报1-3，报道3的出列，求最后一人的编号。(面试官说暴力解也可以，然后自己就模拟着去解，结果中途还是出了很多问题🤔)
 - c. 写两个线程打印1-n，一个线程打印奇数，一个线程打印偶数，顺序不能乱(一开始想复杂了，做成了生产者消费值)
5. 八股文环节，GET和POST的区别
6. 讲一下反射以及应用场景
7. 说一下如何优化mysql查询

还有一些内容记不清了，总结一下，这次面试花了很多时间在笔试上面，感觉自己没有保持刷题，脑袋变迟钝了，做题的时候出了很多问题，然后就是感觉实习经历这一块还是得提前想好怎么说，可以打一下草稿。看来实习的时候得抽点时间刷刷题，不能光顾着做需求了

面经原贴地址：<https://www.nowcoder.com/discuss/524545734834728960>。

非技术问题

自我介绍

自我介绍一般是你和面试官的第一次面对面正式交流，换位思考一下，假如你是面试官的话，你想听到被你面试的人如何介绍自己呢？一定不是客套地说说自己喜欢编程、平时花了很多时间来学习、自己的兴趣爱好是打球吧？

我觉得一个好的自我介绍应该包含这几点要素：

1. 用简单的话说清楚自己主要的技术栈于擅长的领域；
2. 把重点放在自己在行的地方以及自己的优势之处；
3. 重点突出自己的能力比如自己的定位的 bug 的能力特别厉害；

自我介绍并不需要死记硬背，记住要说的要点，面试的时候根据公司的情况临场发挥也是没问题的。另外，网上一般建议的是准备好两份自我介绍：一份对 hr 说的，主要讲能突出自己的经历，会的编程技术一语带过；另一份对技术面试官说的，主要讲自己会的技术细节和项目经验。

自我介绍模板：

社招：

面试官，您好！我叫独秀儿。我目前有 1 年半的工作经验，熟练使用 Spring、MyBatis 等框架、了解 Java 底层原理比如 JVM 调优并且有着丰富的分布式开发经验。离开上一家公司是因为我想在技术上得到更多的锻炼。在上一个公司我参与了一个分布式电子交易系统的开发，负责搭建了整个项目的基础架构并且通过分库分表解决了原始数据库以及一些相关表过于庞大的问题，目前这个网站最高支持 10 万人同时访问。工作之余，我利用自己的业余时间写了一个简单的 RPC 框架，这个框架用到了 Netty 进行网络通信，目前我已经将这个项目开源，在 GitHub 上收获了 2k 的 Star! 说到业余爱好的话，我比较喜欢通过博客整理分享自己所学知识，现在已经是多个博客平台的认证作者。生活中我是一个比较积极乐观的人，一般会通过运动打球的方式来放松。我一直都非常想加入贵公司，我觉得贵公司的文化和技术氛围我都非常喜欢，期待能与你共事！

校招：

面试官，您好！我叫秀儿。大学时间我主要利用课外时间学习了 Java 以及 Spring、MyBatis 等框架。在校期间参与过一个考试系统的开发，这个系统的主要用了 Spring、MyBatis 和 shiro 这三种框架。我在其中主要担任后端开发，主要负责了权限管理功能模块的搭建。另外，我在大学的时候参加过一次软件编程大赛，我和我的团队做的在线订餐系统成功获得了第二名的成绩。我还利用自己的业余时间写了一个简单的 RPC 框架，这个框架用到了 Netty 进行网络通信，目前我已经将这个项目开源，在 GitHub 上收获了 2k 的 Star! 说到业余爱好的话，我比较喜欢通过博客整理分享自己所学知识，现在已经是多个博客平台的认证作者。生活中我是一个比较积极乐观的人，一般会通过运动打球的方式来放松。我一直都非常想加入贵公司，我觉得贵公司的文化和技术氛围我都非常喜欢，期待能与你共事！

讲一下实习经历以及遇到的难点

实习经历的描述一定要避免空谈，尽量列举出你在实习期间取得的成就和具体贡献，使用具体的数据和指标来量化你的工作成果。

示例（这里假设项目细节放在实习经历这里介绍，你也可以选择将实习经历参与的项目放到项目经历中）：

1. 参与项目订单模块的开发，负责订单创建、删除、查询等功能。
2. 排查并解决扣费模块由于扣费父任务和反作弊子任务使用同一个线程池导致的死锁问题。
3. 使用 `CompletableFuture` 并行加载后台用户统计模块的数据信息，平均相应时间从 3.5s 降低到 1s。
4. 使用 Redis+Caffeine 多级缓存优化热门数据（如首页、热门商品）的访问，解决了缓存击穿和穿透问题，查询速度毫秒级，QPS 30w+。
5. 在实习期间，共完成了 10 个需求开发和 5 个问题修复，编写了 2000 行代码和 100 个测试用例，通过了代码评审和测试验收，上线运行稳定无故障。

关于实习经历这块再多提一点。很多同学实习期间可能接触不到什么实际的开发任务，大部分时间可能都是在熟悉和维护项目。对于这种情况，你可以适当润色这段实习经历，找一些简单的功能研究透，包装成自己做的，很多同学都是这么做的。不过，我更建议你在实习期间主动去承担一些开发任务，甚至说对原系统进行优化改造。常见的性能优化方向实践（涉及到多线程、JVM、数据库/缓存、数据结构优化这 4 个常见的性能优化方向）总结请看：<https://t.zsxq.com/0c1uS7q2Y>（这块内容分享在 [知识星球](#) 里了，你也可以自己按照我的思路总结，效果是一样的）。

很多球友不知道如何从性能角度优化项目，这里推荐几个比较容易实现的点，单体项目和分布式项目都适用，涉及到多线程、JVM、数据库/缓存、数据结构优化这4个常见的性能优化方向：

多线程方向优化：

- 1、如何在 SpringBoot 中使用异步方法优化 Service 逻辑提高接口响应速度？- 2022: [如何在SpringBoot中使用异步方法优化Service逻辑提高接口响应速度?_springboo...](#)
- 2、asyncTool: [asyncTool: 解决任意的多线程并行、串行、阻塞、依赖、回调的并行框架，可以任意组合各线程的执...](#)（京东零售开源的一个并行框架，里面大量使用到了 CompletableFuture，可以学习其精华来运用在自己的项目上）
- 3、CompletableFuture 原理与实践-外卖商家端 API 的异步化 - 美团技术团队- 2022: [CompletableFuture原理与实践-外卖商家端API的异步化 - 知乎](#)
- 4、简述 CompletableFuture 异步任务编排 - 掘金 - 2022: [简述CompletableFuture异步任务编排 - 掘金](#)

JVM方向优化：

- 5、JVM 调优实战 - 掘金 - 2022: [JVM调优-JVM调优实践一 - 掘金](#)

数据库/缓存方向：

- 6、MySQL 索引与查询优化 - 西召 - 2019: [MySQL索引与查询优化 - 掘金](#)
- 7、基于 Spring 接口，集成 Caffeine+Redis 两级缓存 - 码农参上 - 2022: [基于Spring接口，集成Caffeine Redis两级缓存 - 掘金](#)
- 8、J2Cache: [J2Cache: Java 两级缓存框架，可以让应用支持两级缓存框架 ehcache\(Caffein...](#)（基于内存和 Redis 的开源两级 Java 缓存框架）
- 9、MySQL 读写分离实战 - 遇见0和1 - 2023: [MySQL8读写分离集群](#)
- 10、MaxScale 实现 MySQL读写分离 - 爱可生开源社区 - 2022: [技术分享 | MaxScale 实现 MySQL读写分离](#)

数据结构结构方向优化：

- 11、换个数据结构，一不小心节约了 591 台机器！ - why 技术 - 2022: [换个数据结构，一不小心节约了 591 台机器！ - why技术 - 博客园](#)（将系统中的本地缓存实现由 HashMap 替换为了 IntObjectHashMap(这个类出自 Netty)节约了2364C 的服务器资源。）
- 12、这个队列的思路是真的好，现在它是我简历上的亮点了。 - why 技术 - 2022: [java - 这个队列的思路是真的好，现在它是我简历上的亮点了。 - 个人文章 - SegmentF...](#)

对于如何回答项目遇到什么困难，如何解决这类问题，可以参考这篇文章：[面试被挂，项目太简单！](#)。

说一下自己以后的发展发向(就像更倾向于技术还是业务理解)

技术本身往往不会产生价值，必须依托于产品/业务才能体现。比如你是一个提供技术服务的公司，你创造的技术产品有人买单或者有人使用。再比如你是一个普通的互联网公司，你们通过技术创造了某个热门 App 为公司创造了营收。

程序员不能只关注技术本身，更要站在业务的角度去思考问题，解决业务的痛点，提高自己的业务能力。

工作初期，由于接触到的业务场景比较少，我们对于一些常见业务的理解可能会不太全面，导致实现的功能总有一些细节没有考虑到。这是很正常的，我想几乎所有同学都会经历这个阶段。随着工作时间的增加，接触到的业务场景也会越来越多，我们的业务能力也会在潜移默化中提高。

我们还要养成系统思考和复盘总结的习惯，这对于提高业务能力非常重要！大淘宝技术的[如何快速理解复杂业务，系统思考问题？](#)这篇文章，详细介绍了系统思考的方法和意义，非常值得一读。特别是在面对复杂的业务场景时，系统思考能够帮助我们把握业务的全貌和关键点，避免陷入细节和局部的思维。

另外，[工作五年之后，对技术和业务的思考](#) 这篇文章是我在两年前看到的一篇对我触动比较深的文章，介绍了作者工作五年之后，对于技术和业务的深度思考。

笔试题

笔试的形式是给你的邮箱发个链接，点进去就是一个在线的编辑器。

写三种单例模式的实现方式

枚举（推荐）：

```
public enum Singleton {  
    INSTANCE;  
    public void doSomething(String str) {  
        System.out.println(str);  
    }  
}
```

《Effective Java》作者推荐的一种单例实现方式，简单高效，无需加锁，线程安全，可以避免通过反射破坏枚举单例。

静态内部类（推荐）：

```
public class Singleton {  
    // 私有化构造方法  
    private Singleton() {
```

```

    }

    // 对外提供获取实例的公共方法
    public static Singleton getInstance() {
        return SingletonInner.INSTANCE;
    }

    // 定义静态内部类
    private static class SingletonInner{
        private final static Singleton INSTANCE = new Singleton();
    }
}

```

当外部类 `Singleton` 被加载的时候，并不会创建静态内部类 `SingletonInner` 的实例对象。只有当调用 `getInstance()` 方法时，`SingletonInner` 才会被加载，这个时候才会创建单例对象 `INSTANCE`。`INSTANCE` 的唯一性、创建过程的线程安全性，都由 JVM 来保证。

这种方式同样简单高效，无需加锁，线程安全，并且支持延时加载。

双重校验锁：

```

public class Singleton {

    private volatile static Singleton uniqueInstance;

    // 私有化构造方法
    private Singleton() {
    }

    public static Singleton getUniqueInstance() {
        //先判断对象是否已经实例化过，没有实例化过才进入加锁代码
        if (uniqueInstance == null) {
            //类对象加锁
            synchronized (Singleton.class) {
                if (uniqueInstance == null) {
                    uniqueInstance = new Singleton();
                }
            }
        }
        return uniqueInstance;
    }
}

```

`uniqueInstance` 采用 `volatile` 关键字修饰也是很有必要的，`uniqueInstance = new Singleton();` 这段代码其实是分为三步执行：

1. 为 `uniqueInstance` 分配内存空间
2. 初始化 `uniqueInstance`
3. 将 `uniqueInstance` 指向分配的内存地址

但是由于 JVM 具有指令重排的特性，执行顺序有可能变成 1->3->2。指令重排在单线程环境下不会出现问题，但是在多线程环境下会导致一个线程获得还没有初始化的实例。例如，线程 T1 执行了 1 和 3，此时 T2 调用 `getUniqueInstance ()` 后发现 `uniqueInstance` 不为空，因此返回 `uniqueInstance`，但此时 `uniqueInstance` 还未被初始化。

这种方式实现起来较麻烦，但同样线程安全，支持延时加载。

推荐阅读：[Java 并发常见面试题总结（中）](#)。

编号为 1-n 的循环报 1-3，报道 3 的出列，求最后一人的编号

标准的约瑟夫环问题。有 n 个人围成一个圈，从某个人开始报数，报到某个特定数字（本题中为 3）时该人出圈，直到只剩下一个人为止。

解决约瑟夫环问题，可以分两种情况：

1. 我们要求出最后留下的那个人的编号（本题要求）。
2. 求全过程，即要算出每轮出局的人。

有多种方法可以解决约瑟夫环问题，其中一种是使用递归的方式。

本题的约瑟夫环问题的公式为： $(f(n-1, k) + k - 1) \% n + 1$ 。f(n,k) 表示 n 个人报数，每次报数报到 k 的人出局，最终最后一个人的编号。

假设 n 为 10，k 为 3，逆推过程如下：

- $f(1, 3) = 1$ （当 $n = 1$ 时，只有一个人，最后一人的编号就为 1）；
- $f(2, 3) = (f(1, 3) + 3 - 1) \% 2 + 1 = 3 \% 2 + 1 = 2$ （当 $n = 2$ 时，最后一人的编号为 2）；
- $f(3, 3) = (f(2, 3) + 3 - 1) \% 3 + 1 = 4 \% 3 + 1 = 2$ （当 $n = 3$ 时，最后一人的编号为 2）；
- $f(4, 3) = (f(3, 3) + 3 - 1) \% 4 + 1 = 4 \% 4 + 1 = 1$ （当 $n = 4$ 时，最后一人的编号为 1）；
- ...
- $f(10, 3) = 3$ （当 $n = 10$ 时，最后一人的编号为 4）；

这个问题对应剑指 Offer 62. 圆圈中最后剩下的数字，两者意思是类似的，比较简单。


```

public class Josephus {

    // 定义递归函数
    public static int f(int n, int k) {
        // 如果只有一个人，则返回 1
        if (n == 1) {
            return 1;
        }
        return (f(n - 1, k) + k - 1) % n + 1;
    }

    public static void main(String[] args) {
        int n = 10;
        int k = 3;
        System.out.println("最后留下的那个人的编号是: " + f(n, k));
    }
}

```

输出：

```
最后留下的那个人的编号是：4
```

写两个线程打印 1-n，一个线程打印奇数，一个线程打印偶数

这道题的实现方式还是挺多的，线程的等待/通知机制（`wait()` 和 `notify()`）、信号量 `Semaphore` 等都可以实现。

这里以 `wait()` 和 `notify()` 为例进行介绍。

我们先定义一个类用于打印奇数和偶数。

```

class ParityPrinter {
    // 线程共享变量，总打印次数
    private int max;
    // 线程共享变量，表示打印次数
    private int number = 1;
    // 线程共享变量，表示是否应该打印奇数
    private boolean odd;

    public ParityPrinter(int max) {
        this.max = max;
    }
}

```



```

}

/**
 * 打印奇数
 */
public synchronized void printOdd() {
    while (number < 100) {
        // 如果当前应该打印偶数，就等待
        while (odd) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        System.out.println(Thread.currentThread().getName() + " : " + number);
        number++;
        // 设置为应该打印偶数
        odd = true;
        // 唤醒另一个线程
        notify();
    }
}

/**
 * 打印偶数
 */
public synchronized void printEven() {
    while (number < 100) {
        // 如果当前应该打印奇数，就等待
        while (!odd) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
        System.out.println(Thread.currentThread().getName() + " : " + number);
        number++;
        // 设置为应该打印奇数
        odd = false;
        // 唤醒另一个线程
        notify();
    }
}

```

```

    }

}

}

```

在这个类中，有 3 个线程共享变量，`max` 表示总打印次数，`number` 表示已经打印的次数，`odd` 表示是否应该打印奇数。并且，有两个 `synchronized` 关键字修饰的方法，`printOdd()` 用于打印奇数，`printEven()` 用于打印偶数。使用 `synchronized` 关键字可以保证同一时刻只有一个线程能够访问共享变量，线程间使用 `wait()` 和 `notify()` 方法来实现线程间的通信。

接着，我们创建两个线程，一个负责打印奇数，一个负责打印偶数。

```

// 打印 1-100
ParityPrinter printer = new ParityPrinter(100);
// 创建两个线程，分别传入不同的 Runnable 对象
Thread t1 = new Thread(printer::printOdd);
Thread t2 = new Thread(printer::printEven);
// 启动两个线程
t1.start();
t2.start();

```

输出如下：

```

Thread-0 : 1
Thread-1 : 2
Thread-0 : 3
Thread-1 : 4
...
Thread-0 : 63
Thread-1 : 64
Thread-0 : 65
Thread-1 : 66
...
Thread-1 : 96
Thread-0 : 97
Thread-1 : 98
Thread-0 : 99

```

技术问题

八股文资料首推 [《Java 面试指北》](#) (质量很高，专为面试打造，配合 JavaGuide 食用)。和 [JavaGuide](#)。里面不仅仅是原创八股文，还有很多对实际开发有帮助的干货。除了这两份资料之外，你还可以去网上找一些其他的优质的文章、视频来看。

《Java面试指北》

星球内部的《Java面试进阶指北》属于是 JavaGuide 的补充完善，两者配合使用！

...

☆ 收藏

目录

全部文档 目录管理

介绍	05-26 20:58
▸ 面试准备篇	10-14 15:11
▾ 技术面试题篇	03-21 19:03
▸ 系统设计	05-29 15:10
▸ Java	
▸ 数据库	
▸ 常见框架	
▸ 分布式	05-29 15:12
▸ 高并发	05-29 15:13
▸ 服务器	05-29 15:13
▸ Devops	05-29 15:15
▸ 技术面试题自测篇	05-29 15:16
▸ 面经篇	05-29 15:24
▸ 练级攻略篇	05-28 15:58
▸ 工作篇	08-14 19:12

一定不要抱着一种思想，觉得八股文或者基础问题的考查意义不大。如果你抱着这种思想复习的话，那效果可能不会太好。实际上，个人认为还是很有意义的，八股文或者基础性的知识在日常开发中也会需要经常用到。例如，线程池这块的拒绝策略、核心参数配置什么的，如果你不了解，实际项目中使用线程池可能就用的不是很明白，容易出现问。而且，其实这种基础性的问题是最容易准备的，像各种底层原理、系统设计、场景题以及深挖你的项目这类才是最难的！

✂️ JavaGuide 官方网站地址：[javaguide.cn](#)。

✂️ [知识星球](#)（点击链接即可查看星球的详细介绍）包含的大部分资料都整理在了 [星球使用指南](#) 中（一定要看！！！！）。

星球专属资料概览：

🌟 知识星球的使用指南（必看）：

👉 专属资料：

星球目前一共有 6 个专栏（见图1《Java 面试指北》就在其中），均为星球专属，📌 阅读地址：

🔗 下面是星球提供的一些专栏（目前...（会定期修改密码，避免盗版传播。只要购买过一次星球，即可永久找我获取最新密码）。

项目资料合集（持续更新中）：🔗 实战项目资料合集。

除了这些专栏之外，星球还有下面这些常用的优质 PDF 技术资源：

- 📌 原创 PDF 面试资料（选择自己需要的即可）：🔗 原创PDF面试资料（内容很全面...。
- 📌 Java 面试常见问题总结（2024 最新版，用于自测）：🔗 Java面试常见问题总结（20...
- 高频笔试题（非常规Leetcode类型）PDF：🔗 非常规 Leetcode 类型的高频笔试题
- 20 道 HR 面常见问题：🔗 20道HR面常见问题
- 线上常见问题案例和排查工具：🔗 进一步完善了之前整理的线上常见...

星球提供的资料可能无法满足每一位球友的期待，如果你有其他迫切需要的内容，欢迎微信联系我，给我留言，我会尽力为你提供帮助！




1、《Java面试指北》(配合 JavaGuide 使用, 会根据每一年的面试情况对内容进行更新完善, 故不提供 PDF 版本): <https://www.yuque.com/books/share/04ac99ea-7726-4a...> (密码: )

JavaGuide 地址: <https://javaguide.cn/>, 《Java 面试指北》的学习建议在这里: <https://t.zsxq.com/QNFMFAU>。如果不知道《Java面试指北》和开源版的关系, 可以看看这份建议。

2、《后端面试高频系统设计&场景题》: <https://www.yuque.com/snailclimb/tangw3> 密码: 

这部分内容本身是属于《Java面试指北》的, 后面由于内容篇幅较多, 因此被单独提了出来。

3、《Java 必读源码系列》(目前已经整理了 Dubbo 2.6.x、Netty 4.x、SpringBoot2.1 的源码): <https://www.yuque.com/books/share/7f846c65-f32e-41...> (密码: )

欢迎在评论区说出你们想要看的框架/中间件的源码!


4. 《从零开始写一个RPC框架》: <https://www.yuque.com/books/share/b7a2512c-6f7a-4a...> (密码: )

RPC 框架地址: <https://gitee.com/SnailClimb/guide-rpc-framework>。

5、《分布式、高并发、Devops 面试扫盲》: 已经并入《Java面试指北》中。

6、《Kafka常见面试题/知识点总结》: <https://www.yuque.com/books/share/dd07d89b-9437-4f...> (密码: )

7、《程序员副业赚钱之路》 <https://www.yuque.com/books/share/1bd77211-f7e0-41...> (密码: )

8、《Guide的读书笔记与文章精选集》(经典书籍精读笔记分享) <https://www.yuque.com/books/share/f63faff5-53f9-41...> (密码: )



星球部分面试资料介绍：

- 《Java 面试指北》（面试专版，Java面试必备）
- 《后端面试高频系统设计&场景题》（20+高频系统设计&场景面试题）
- 《Java 必读源码系列》（目前已经整理了 Dubbo 2.6.x、Netty 4.x、SpringBoot2.1 的源码）
- 《后端高频笔试题（非常规Leetcode类型）》（新增多线程相关的手撕题）
- 《Java 面试常见问题总结（2024 最新版）》（350+ 道超高频面试题总结）
- 20 道 HR 面的常见问题（HR 面必备）

星球项目资源：

- 网盘项目（网盘项目通用的介绍模板、优化思路以及面试知识点考察分析）
- 手写 RPC 框架（如何从零开始基于 Netty+Kyro+Zookeeper 实现一个简易的 RPC 框架）

GET 和 POST 的区别

这个问题在知乎上被讨论的挺火热的，地址：<https://www.zhihu.com/question/28586791>。

GET 和 POST 到底有什么区别？

"get是从服务器上获取数据，post是向服务器传送数据"（网上的回答），这句话post请求就不从服务器获取数据了？现在我是这么理解get和post： ...显示全部

关注问题

写回答

邀请回答

好问题 62

2 条评论

分享

78 个回答

默认排序



大宽宽

飞书任务诚招人才中-有意者私信

+ 关注

沉默王二、Asterisk、sowhat1412 等 6,185 人赞同了该回答

收藏数4000，赞数2000，没天理啊～

这个问题虽然看上去很初级，但实际上却涉及到方方面面，这也就是为啥面试里老爱问这个的原因之一。

HTTP最早被用来做浏览器与服务器之间交互HTML和表单的通讯协议；后来又被被广泛的扩充到接口格式的定义上。所以在讨论GET和POST区别的时候，需要现确定下到底是浏览器使用的GET/POST还是用HTTP作为接口传输协议的场景。

相关问题

- get 和 post 有哪些区别？ 0 个回答
- Get与post的区别是什么？ 1 个回答
- get与post有什么区别？ 3 个回答
- get 和post的区别在哪里啊？ 0 个回答
- 想知道get和post的区别到底是什么，网上说法貌似大多错误的？ 0 个回答

帮助中心

知乎隐私保护指引 申请开通机构号 联系我们

举报中心

涉未成年举报 网络谣言举报 涉企虚假举报 更多

GET 和 POST 是 HTTP 协议中两种常用的请求方法，它们在不同的场景和目的下有不同的特点和用法。一般来说，可以从以下几个方面来区分它们（重点搞清两者在语义上的区别即可）：

- 语义（主要区别）：GET 通常用于获取或查询资源，而 POST 通常用于创建或修改资源。
- 幂等：GET 请求是幂等的，即多次重复执行不会改变资源的状态，而 POST 请求是不幂等的，即每次执行可能会产生不同的结果或影响资源的状态。
- 格式：GET 请求的参数通常放在 URL 中，形成查询字符串（querystring），而 POST 请求的参数通常放在请求体（body）中，可以有多种编码格式，如 application/x-www-form-urlencoded、multipart/form-data、application/json 等。GET 请求的 URL 长度受到浏览器和服务器的限制，而 POST 请求的 body 大小则没有明确的限制。不过，实际上 GET 请求也可以用 body 传输数据，只是并不推荐这样做，因为这样可能会导致一些兼容性或者语义上的问题。
- 缓存：由于 GET 请求是幂等的，它可以被浏览器或其他中间节点（如代理、网关）缓存起来，以提高性能和效率。而 POST 请求则不适合被缓存，因为它可能有副作用，每次执行可能需要实时的响应。
- 安全性：GET 请求和 POST 请求如果使用 HTTP 协议的话，那都不安全，因为 HTTP 协议本身是明文传输的，必须使用 HTTPS 协议来加密传输数据。另外，GET 请求相比 POST 请求更容易泄露敏感数据，因为 GET 请求的参数通常放在 URL 中。

再次提示，重点搞清两者在语义上的区别即可，实际使用过程中，也是通过语义来区分使用 GET 还是 POST。不过，也有一些项目所有的请求都用 POST，这个并不是固定的，项目组达成共识即可。

如何优化 MySQL 查询

这个问题涉及可以聊的东西有点多，可以看看下面这两篇文章：

- MySQL 高性能优化规范建议总结
- MySQL 执行计划分析

另外，《Java 面试指北》的技术面试题篇有详细总结归纳 SQL 优化手段。

1 首页

2 目录

3 介绍

更新记录

面试准备篇（必看）

技术面试题篇

- 系统设计&场景题
- Java
- 数据库&缓存
- 常见框架
- 分布式&微服务
- 高并发
 - 高可用：如何设计一个高可用系统？
 - 高性能：负载均衡的常见算法有哪...
 - 高性能：池化技术的应用场景
 - 高性能：零拷贝为什么能提升性能？
 - 高性能：有哪些常见的 SQL 优化...
 - 高可用：如何避免微服务中的雪崩...
 - 高可用：降级和熔断有什么区别？
 - 高可用：灰度发布和回滚有什么用？
- 服务器

技术面试题自测篇

面经篇

练级攻略篇

工作篇

高性能：有哪些常见的 SQL 优化手段

避免使用 SELECT *

- SELECT * 会消耗更多的 CPU。
- SELECT * 无用字段增加网络带宽资源消耗，增加数据传输时间，尤其是 blob、text）。
- SELECT * 无法使用 MySQL 优化器覆盖索引的优化（基于 MySQL 优化速度极快，效率极高，业界极为推荐的查询优化方式）
- SELECT <字段列表> 可减少表结构变更带来的影响。

分页优化

普通的分页在数据量小的时候耗费时间还是比较短的。

1 SELECT `score`,`name` FROM `cus_order` ORDER BY `score` DESC

如果数据量变大，达到百万甚至是千万级别，普通的分页耗费的时间就非常长了

1 SELECT `score`,`name` FROM `cus_order` ORDER BY `score` DESC

2 SELECT `score`,`name` FROM `cus_order` ORDER BY `score` DESC

如何优化呢？ 可以将上述 SQL 语句修改为子查询。

大纲

避免使用 SELECT *

分页优化

尽量避免多表做 join

建议不要使用外键与级联

选择合适的字段类型

尽量用 UNION ALL 代替 UNION

批量操作

Show Profile 分析 SQL 执行性能

优化慢 SQL

正确使用索引

选择合适的字段创建索引

被频繁更新的字段应该慎重建立索引

尽可能的考虑建立联合索引而不是单

注意避免冗余索引

考虑在字符串类型的字段上使用前缀

避免索引失效

删除长期未使用的索引

参考

重点关注慢查询、EXPLAIN 命令、SHOW PROFILE 和 SHOW PROFILES 命令即可。

反射及应用场景

如果说大家研究过框架的底层原理或者咱们自己写过框架的话，一定对反射这个概念不陌生。反射之所以被称为框架的灵魂，主要是因为它赋予了我们在运行时分析类以及执行类中方法的能力。通过反射你可以获取任意一个类的所有属性和方法，你还可以调用这些方法和属性。

反射可以让我们的代码更加灵活、为各种框架提供开箱即用的功能提供了便利。

不过，反射让我们在运行时有了分析操作类的能力的同时，也增加了安全问题，比如可以无视泛型参数的安全检查（泛型参数的安全检查发生在编译时）。另外，反射的性能也要稍差点，不过，对于框架来说实际是影响不大的。

像咱们平时大部分时候都是在写业务代码，很少会接触到直接使用反射机制的场景。但是！这并不代表反射没有用。相反，正是因为反射，你才能这么轻松地使用各种框架。像 Spring/Spring Boot、MyBatis 等等框架中都大量使用了反射机制。

这些框架中也大量使用了动态代理，而动态代理的实现也依赖反射。

比如下面是通过 JDK 实现动态代理的示例代码，其中就使用了反射类 `Method` 来调用指定的方法。

```
public class DebugInvocationHandler implements InvocationHandler {  
    /**  
     * 代理类中的真实对象  
     */  
    private final Object target;  
  
    public DebugInvocationHandler(Object target) {  
        this.target = target;  
    }  
  
    public Object invoke(Object proxy, Method method, Object[] args) throws  
    InvocationTargetException, IllegalAccessException {  
        System.out.println("before method " + method.getName());  
        Object result = method.invoke(target, args);  
        System.out.println("after method " + method.getName());  
        return result;  
    }  
}
```

另外，像 Java 中的一大利器 **注解** 的实现也用到了反射。

为什么你使用 Spring 的时候，一个 `@Component` 注解就声明了一个类为 Spring Bean 呢？为什么你通过一个 `@Value` 注解就读取到配置文件中的值呢？究竟是怎么起作用的呢？

这些都是因为你可以基于反射分析类，然后获取到类/属性/方法/方法的参数上的注解。你获取到注解之后，就可以做进一步的处理。

更多关于 Java 基础的常见面试问题，参考 [JavaGuide](http://javaguide.cn)（javaguide.cn）的总结即可。

JavaGuide(Java面试 + 学习指南)

面试指南 开源项目 技术书籍 程序人生 知识星球 网站相关

搜索

必看

面试准备

Java

基础

Java基础常见面试题总结(上)

Java基础常见面试题总结(中)

Java基础常见面试题总结(下)

重要知识点

集合

Java集合常见面试题总结(上)

Java集合常见面试题总结(下)

Java集合使用注意事项总结

源码分析

并发编程

Java并发常见面试题总结 (上)

Java并发常见面试题总结 (中)

Java并发常见面试题总结 (下)

重要知识点

IO

JVM

新特性

计算机基础

数据库

```
1 public class DebugInvocationHandler implements InvocationHandler {
2     /**
3      * 代理类中的真实对象
4      */
5     private final Object target;
6
7     public DebugInvocationHandler(Object target) {
8         this.target = target;
9     }
10
11    public Object invoke(Object proxy, Method method, Object[] args) throws InvocationTargetException {
12        System.out.println("before method " + method.getName());
13        Object result = method.invoke(target, args);
14        System.out.println("after method " + method.getName());
15        return result;
16    }
17 }
18
```

此外，像 Java 中的一大利器 注解 的实现也用到了反射。

为什么你使用 Spring 的时候，一个 @Component 注解就声明了一个类为 Spring Bean 呢？为什么你通过一个 @Value 注解就读取到配置文件中的值呢？究竟是怎么起作用的呢？

这些都是因为你可以基于反射分析类，然后获取到类/属性/方法/方法的参数上的注解。你获取到注解之后，就可以做进一步的处理。

注解

何谓注解？

Annotation（注解）是 Java5 开始引入的新特性，可以看作是一种特殊的注释，主要用于修饰类、方法或者变量，提供某些信息供程序在编译或者运行时使用。

此页内容

异常

Exception 和 Error 有什么区...

Checked Exception 和 Unch...

Throwable 类常用方法有哪些？

try-catch-finally 如何使用？

finally 中的代码一定会执行吗？

如何使用 try-with-resources ...

异常使用有哪些需要注意的地...

泛型

什么是泛型？有什么作用？

泛型的使用方式有哪几种？

项目中哪里用到了泛型？

反射

何谓反射？

反射的优缺点？

反射的应用场景？

注解

何谓注解？

注解的解析方法有哪几种？

SPI

何谓 SPI？

SPI 和 API 有什么区别？

SPI 的优缺点？

序列化和反序列化

什么是序列化？什么是反序列化？

如果有些字段不想进行序列化...