1. **加密解密了解么？几种算法，讲一下你了解的**

不懂，我搞后台的，不搞算法。

1. **多线程了解么？什么是线程安全？**

可以举个例子，hashmap的扩容阿，get null那些线程不安全的例子。

1. **说一个你最熟悉的设计模式**

工厂、享元、单例，随便说。

1. **讲一下你项目中用到了哪些设计模式**

雷电游戏走起。

1. **Java的hashmap的原理、线程安全性，什么是线程安全的？如何实现线程安全。**

可以秀一波了。Concurrenthashmmap。

1. **数据库的索引、数据库引擎了解么？数据库有哪些优化的方法？讲你自己知道的。**

秀起。

1. **写单例模式**

答主写的是双检查锁单例，问了为什么用Volatile，synchronize

1. **数据连接池采用了什么设计模式？意义是什么？**

享元模式， 数据库连接池来说，url、driverClassName、username、password及dbname，这些属性对于每个连接来说都是一样的，所以就适合用享元模式来处理，建一个工厂类，将上述类似属性作为内部数据，其它的作为外部数据，在方法调用时，当做参数传进来，这样就节省了空间，减少了实例的数量。

1. **为什么请求过来时可以找到在注解中的URL路径**

通过配置的[web.xml](https://www.baidu.com/s?wd=web.xml&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)的servlet(DispatcherServlet)，拦截所有请求，然后就通过SpringMVC的流程，找到URL对应的Handler。

1. **自己写注解实现增强的思路**

第一种方法：其实就是使用拦截，用一个切面注解@PointCut，把需要拦截的地方给拦截下来，然后在用@before在前后进行方法增强。

还有第二种方法，就是配置好拦截器。实现了Spring 的HandlerInterceptor 接口，然后重写3个方法，preHandle 、postHandle和afterCompletion；最后在在SpringMVC的配置文件中加上支持MVC的配置说明。就可以拦截所需的类并且实现增强。

1. **TCP粘包拆包解决**

在进行Java NIO学习时，发现，**如果客户端连续不断的向服务端发送数据包时，服务端接收的数据会出现两个数据包粘在一起的情况**，这就是TCP协议中经常会遇到的粘包以及拆包的问题。

我们都知道TCP属于传输层的协议，传输层除了有TCP协议外还有UDP协议。那么UDP是否会发生粘包或拆包的现象呢？答案是不会。**UDP是基于报文发送的**，从UDP的帧结构可以看出，在UDP首部采用了16bit来指示UDP数据报文的长度，因此在**应用层能很好的将不同的数据报文区分开，从而避免粘包和拆包的问题。**而**TCP是基于字节流的，**虽然应用层和TCP传输层之间的数据交互是大小不等的数据块，但是**TCP把这些数据块仅仅看成一连串无结构的字节流，没有边界**；另外**从TCP的帧结构也可以看出，在TCP的首部没有表示数据长度的字段**，基于上面两点，在使用TCP传输数据时，才有粘包或者拆包现象发生的可能。

**粘包、拆包表现形式**

现在假设客户端向服务端连续发送了两个数据包，用packet1和packet2来表示，那么服务端收到的数据可以分为三种，现列举如下：

第一种情况，接收端正常收到两个数据包，即没有发生拆包和粘包的现象，此种情况不在本文的讨论范围内。

第二种情况，接收端只收到一个数据包，由于TCP是不会出现丢包的，所以这一个数据包中包含了发送端发送的两个数据包的信息，这种现象即为粘包。这种情况由于接收端不知道这两个数据包的界限，所以对于接收端来说很难处理。

第三种情况，这种情况有两种表现形式，如下图。接收端收到了两个数据包，但是这两个数据包要么是不完整的，要么就是多出来一块，这种情况即发生了拆包和粘包。这两种情况如果不加特殊处理，对于接收端同样是不好处理的。

**粘包、拆包发生原因**

发生TCP粘包或拆包有很多原因，现列出常见的几点，可能不全面，欢迎补充，

1、**要发送的数据大于TCP发送缓冲区剩余空间大小，将会发生拆包。**

2、**待发送数据大于MSS（最大报文长度），TCP在传输前将进行拆包。**

3、**要发送的数据小于TCP发送缓冲区的大小，TCP将多次写入缓冲区的数据一次发送出去，将会发生粘包。**

4、**接收数据端的应用层没有及时读取接收缓冲区中的数据，将发生粘包。**

**等等。**

**粘包、拆包解决办法**

通过以上分析，我们清楚了粘包或拆包发生的原因，那么如何解决这个问题呢？解决问题的关键在于如何给每个数据包添加边界信息，常用的方法有如下几个：

1、发送端给**每个数据包添加包首部**，首部中应该**至少包含数据包的长度**，这样接收端在接收到数据后，通过读取包首部的长度字段，便知道每一个数据包的实际长度了。

2、发送端将**每个数据包封装为固定长度**（不够的可以通过补0填充），这样接收端**每次从接收缓冲区中读取固定长度的数据**就自然而然的**把每个数据包拆分开来。**

3、可以在**数据包之间设置边界，如添加特殊符号**，这样，接收端通过这个边界就可以将不同的数据包拆分开。

等等。

1. **MyBatis分页如何做**

**引入依赖，配置xml，调用函数。**

1. **ArrayList和LinkedList有什么区别？**

1.ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。   
2.对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。   
3.对于新增和删除操作add和remove，LinedList比较占优势，因为ArrayList要移动数据。 （但是其实我觉得是不一定的。）

1. **http有哪几种方式**

根据HTTP标准，HTTP请求可以使用多种请求方法。

HTTP1.0定义了三种请求方法： GET, POST 和 HEAD方法。

HTTP1.1新增了五种请求方法：OPTIONS, PUT, DELETE, TRACE 和 CONNECT 方法。

|  |  |  |
| --- | --- | --- |
| **序号** | **方法** | **描述** |
| 1 | GET | **请求指定的页面信息**，并返回实体主体。 |
| 2 | HEAD | 类似于get请求，只不过返回的响应中没有具体的内容，用于获取报头 |
| 3 | POST | **向指定资源提交数据进行处理请求**（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。 |
| 4 | PUT | 从客户端向服务器传送的数据取代指定的文档的内容。 |
| 5 | DELETE | 请求服务器删除指定的页面。 |
| 6 | CONNECT | HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。 |
| 7 | OPTIONS | 允许客户端查看服务器的性能。 |
| 8 | TRACE | 回显服务器收到的请求，主要用于测试或诊断。 |

1. **get方式有没body**

# Get请求

GET /hello/index.html HTTP/1.1

#【↑】request line

#【↓】request headers

Accept: \*/\*

Accept-Language: zh-cn

Accept-Encoding: gzip, deflate

Host: localhost:8000

#发送完关闭连接 or 等待

Connection: Keep-Alive

Cookie: JSESSIONID=BBBA54D519F7A320A54211F0107F5EA6

**get请求是不存在request body的。**

# Post请求

#url的部分仍然是urlencoded

POST /hello/checkUser.html?opt=xxx HTTP/1.1

Referer: http://localhost:8000/hello/index.html

Accept: \*/\*

Accept-Language: zh-cn

#纯文本的编码：不编码

Content-Type: text/plain

Accept-Encoding: gzip, deflate

Host: localhost:8000

Content-Length: 20

Connection: Keep-Alive

Cache-Control: no-cache

Cookie: JSESSIONID=BBBA54D519F7A320A54211F0107F5EA6

#【↑】/r/n

#【↓】request body

12345678901234567890

1. **LinkedList尾端插入数据的时间复杂度，LinkedList在给定位置插入数据的时间复杂度。**

## LinkedList双向链表，还有自动优化，从位置判定由左边还是右边出发。

1. **error和exception区别，异常的名称，产生原因，解决方法, 异常分为哪几类**

Error类和Exception类的父类都是throwable类，他们的区别是：

**Error类一般是指与虚拟机相关的问题，如系统崩溃，虚拟机错误，内存空间不足，方法调用栈溢等。**对于这类**错误的导致的应用程序中断**，仅靠程序本身无法恢复和和预防，遇到这样的错误，建议让程序终止。

Exception类**表示程序可以处理的异常，可以捕获且可能恢复。**遇到这类异常，**应该尽可能处理异常，使程序恢复运行，而不应该随意终止异常。**

Exception类又分为**运行时异常**（Runtime Exception）和**受检查的异常**(Checked Exception )，运行时异常;ArithmaticException,IllegalArgumentException，编译能通过，但是一运行就终止了，**程序不会处理运行时异常，出现这类异常，程序会终止。**而受检查的异常，要么用try。。。catch捕获，要么用throws字句声明抛出，交给它的父类处理，否则编译不会通过。

**常见的运行时异常;**

ArrayIndexOutOfBoundsException **数组下标越界异常，**

ArithmaticException **算数异常 如除数为零**

NullPointerException **空指针异常**

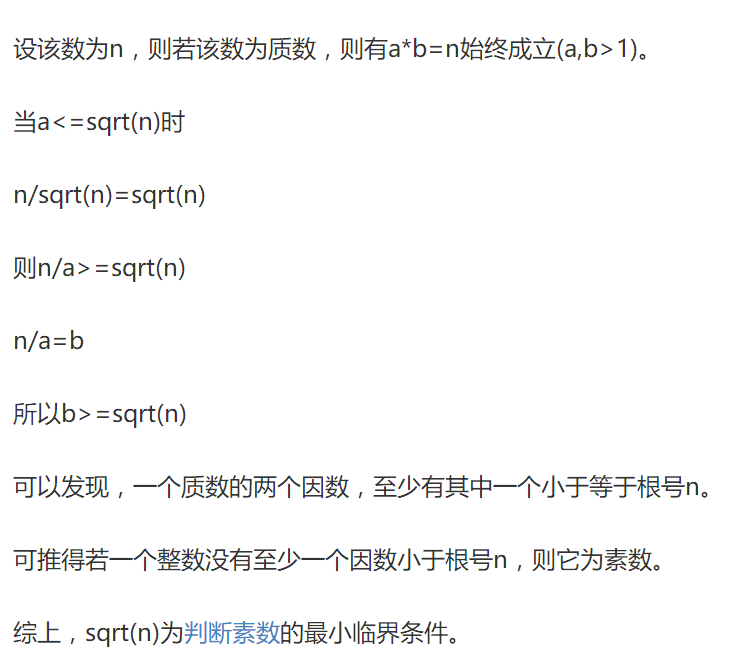
IllegalArgumentException **不合法参数异常**

1. **写出1-1000之间的素数。题目不是很难，主要代码风格。**

**1.简单的遍历，就不说了。**

**2.遍历前半部分。**

**3.遍历math.sqrt**

**优化：**

**代码如下：import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** sushu {  
 **public static void** main(String[] args) {  
 **int** count = 0;  
 List<Integer> list = **new** ArrayList<>();  
 **for** (**int** i = 2; i < 1000; i++) {  
 **boolean** isTrue = **true**;  
 **for** (**int** j = 2; j <= Math.*sqrt*((**double**) i); j++) {  
 **if** (i % j == 0) {  
 isTrue = **false**;  
 **break**;  
 }  
 }  
 **if** (isTrue == **true**) {  
 list.add(i);  
 count++;  
 }  
 }  
 System.***out***.println(count);  
 System.***out***.println(list.toString());  
 }  
}

1. **二分法**
2. **package** yilaidaozhi;  
     
   **public class** erfen {  
    **public static void** main(String[] args) {  
    **int**[] arrays = {0, 3, 3, 3, 4, 5, 6, 7};  
   *// int[] arrays = {};* **int** key = 3;  
    **int** c = *BinaryChop*(arrays, key);  
    System.***out***.println(c);  
    }  
     
    **public static int** BinaryChop(**int**[] arrays, **int** key) {  
    **int** length = arrays.**length**;  
    **int** min = 0;  
    **int** max = length - 1;  
     
    **if** (arrays.**length** == 0) {  
    **return** 0;  
    }  
    **if** (arrays.**length** == 1) {  
    **if** (arrays[0] == key) {  
    **return** 1;  
    } **else** {  
    **return** 0;  
    }  
    }  
    **if** (key > arrays[max] || key < arrays[0]) {  
    **return** 0;  
    }  
    **int** first = *getFirst*(arrays, key, min, max);  
    **int** last = *getLast*(arrays, key, min, max);  
    **if** (last == 0 && first == 0) {  
    **return** 0;  
    }  
    **return** last - first + 1;  
    }  
     
    **public static int** getFirst(**int**[] arrays, **int** key, **int** min, **int** max) {  
    **while** (min <= max) {  
    **int** mid = (min + max) / 2;  
    **if** (arrays[mid] < key && min <= max) {  
    min = mid+1;  
    **continue**;  
    }  
    **if** (arrays[mid] > key && min <= max) {  
    max = mid-1;  
    **continue**;  
    }  
    **if** (arrays[mid] == key && min <= max) {  
    **if** (mid - 1 >= 0 && arrays[mid - 1] == key) {  
    max = mid - 1;  
    **continue**;  
    } **else** {  
    **return** mid;  
    }  
    }  
    }  
    **return** 0;  
    }  
     
    **public static int** getLast(**int**[] arrays, **int** key, **int** min, **int** max) {  
    **while** (min <= max) {  
    **int** mid;  
    **if** (min == max) {  
    mid = max;  
    } **else** {  
    mid = (min + max) / 2+1;  
    }  
    **if** (arrays[mid] < key && min <= max) {  
    min = mid+1;  
    **continue**;  
    }  
    **if** (arrays[mid] > key && min <= max) {  
    max = mid-1;  
    **continue**;  
    }  
    **if** (arrays[mid] == key && min <= max) {  
    **if** (mid + 1 <= max && arrays[mid + 1] == key) {  
    min = mid + 1;  
    } **else** {  
    **return** mid;  
    }  
    }  
    }  
    **return** 0;  
    }  
     
   }