本文主要从面试角度讲述泛型问题在面试中该如何回答，如果只想看问题答案，可以重点留意灰色背景的问题，其他的文字主要是扩展内容。
关于泛型，Jdk1.5 以后推出来的特性，一经推出就收到 Java 程序员大力赞扬，因为真的解决一些编码问题。下面我们来看看如此重要的泛型，在面试中会怎么考察呢。
面试问题概览
下面我罗列一些大厂面试中，对于泛型一些常见问题。
泛型了解吗？介绍一下泛型？【快手】
泛型实现的原理，为什么要实现泛型【阿里】
泛型实现原理 【去哪儿】
java 的泛型，super 和 extend 的区别？泛型擦除的原理【华为】
什么是泛型，什么时候需要利用泛型【字节跳动】
java 的泛型，有什么缺点【腾讯】
可以看到泛型在各个大厂面试中已经成为了笔考题目，回想自己当初校招，因为泛型回答不好而错失阿里 Offer，至今仍然悔恨不已。
真实面试回顾
一个身着灰色格子衬衫，拿着闪着硕大的
大黄同学是吧，我看你简历上面写熟悉 Java 基础以及常见的 Java 特性。那你先说说你对泛型的理解吧
此刻咱们得淡定，虽然自己准备过，也得慢慢道来。
面试官您好，泛型是 JDK1.5 之后提出的新概念，本意是让同一套代码可以适应更多的类型。
大家都写过议论文对吧，对于一个问题需要正反两面说，优点、缺点，一一道来。
缺点：在没有泛型之前一旦某个接口定义了参数为某个类型，则实现了该接口的方法必须采用同样类型参数，不利于程序的扩展。
优点：
比如在创建容器的时候，指定容器持有何种类型，以便在编译期间就能够保证类型的正确性，而不是将错误留到运行的时候。
无论是创建类、方法的时候都可以泛化参数，可以做到代码的复用。
泛型的出现最主要目的为了更好的创建容器类
比如下面这个例子中，在定义 Dog 类的时候不需要指定其属性 a 的含义，只需要利用 T 来表示，在 new 该对象的时候指定对应的类型即可。

```
1  /**
2   *
3   * @param <T>
4   */
5  public class Dog<T> {
6      private T a;
7      public Dog(T a) {
8          this.a = a;
9      }
10
11     public void set(T a) {
12         this.a = a;
13     }
14     public T get() {
```

```
15      return a;
16    }
17
18    public static void main(String[] args) {
19      // 在实例化的时候才确定类型
20      Dog<Integer> h3 = new Dog<Integer>(new Integer(3));
21      // 然后就可以不用强制类型转化了
22      Integer a = h3.get();
23      // h3.set("Not an Automobile"); // Error
24      // h3.set(1.11); // Error
25    }
26 }
```

面试官摸了摸笔记本，心想，回答的还可以，继续追问。

你刚才说泛型消除了类型之间的差异，那你知道 jdk 底层是如何做到的吗？

jdk 是通过类型擦除来实现泛型的，编译器在编译之后会擦除了所有类型相关的信息，所以在运行时不存在任何类型相关的信息。比如如果编写 List<String> names = new Arraylist<>(); 在运行的时候，仅用 List 来表示。

先不要高兴，面试官可能会继续追问。

面试官：既然你说编译时就已经擦除了，为什么要用类型擦除呢？

答：这么做是为了向前兼容，为了兼容 jdk1.5 以前的程序，确保能和 Java 5 之前的版本开发二进制类库进行兼容。

好小伙，这都知道，看来之前做足了功课，还得继续问问，试试深浅。

面试官：你知道泛型中的通配符吧，那你说一下什么是限定通配符和非限定通配符。

答：限定通配符，是限定类型必须是某一个类型的子类。

比如：List<? extends Fruit> flist 表示具有任何从 Fruit 继承的类型的列表。

而限定通配符：<? super Fruit> 确保类型必须是 Fruit 的父类来设定类型的下界。

不要意味 new 一个 List<? extends Fruit>的 flist 就可以把任何的水果塞到 flist 里面，很遗憾的告诉你，下面的程序编译不通过的。

```
1  /**
2   * 泛型中通配符的应用
3   */
4  public class GenericsAndCovariance {
5    public static void main(String[] args) {
6      // new 一个水果篮子
7      List<? extends Fruit> flist = new ArrayList<Apple>();
8      // 下面编译不会通过
9      flist.add(new Apple());
10     // 下面编译不会通过
11     flist.add(new Orange());
12     // 下面编译同样不会通过
13     // flist.add(new Fruit());
14   }
15 }
```

看到这个程序是否感觉到有点懵，我自己创建的水果篮子，我无法往里面放苹果、橙子，甚

至连篮子对象也不能放。那我这个篮子还有屁用哦。
是的，确实没有什么用，似乎和预期不太一样。
但是想想也合乎道理。如果不知道 list 持有什么对象，那么怎么样才能安全地向其中添加对象呢？如果允许了，则获取元素的时候，就会出错。
下面面试继续：
面试官：那你再说说泛型可以用到什么地方？
大黄：泛型即可用用于类定义中、接口的定义、方法的定义、同时也广泛用于容器中。
在类中使用泛型，可以让类中的属性由泛型定义，而不需要提前知道属性的类型。
在接口中使用泛型，可以让接口方法的返回值按照各自实现自由定义。这也是常见的工厂设计模式的一种应用。
在方法中利用泛型，可以做到方法的复用，同一个方法可以传入不同类型参数。实现时只需要将泛型参数列表置于返回值之前即可，JDK1.8 中 Stream 中很多流式方法的定义采用了泛型。
记得加上这句话
我觉得，使用泛型机制最吸引人的地方，在使用容器的地方，比如 List、Set、Map，因为在 1.5 以前，在泛型出现之前，当将一个对象放到容器中，这个对象会默认的转化为 Object 类型，因此会丢失掉类型信息，可能将一个 Apple 对象放到 Orange 容器中，然后试图从 Orange 容器中获取对象时，得到的是一个 Apple，强制转化必然出问题，会抛出 RuntimeException，但是有了泛型之后，这种问题在写代码，也就是说在编译期间就会暴露，而不是在运行时暴露。
比如下面定义接口用于生成不同的对象：

```
1  /**
2   * 定义一个生成器
3   * @param <T>
4   */
5  public interface Generator<T> {
6      /**
7       * 定义生成下一个对象
8       * @return
9       */
10     T next();
11 }
12
13 /**
14  * 利用泛型生成器来生成费波列切数
15  */
16 public class Fibonacci implements Generator<Integer> {
17     private int count = 0;
18
19     public static void main(String[] args) {
20         Fibonacci gen = new Fibonacci();
21         for (int i = 0; i < 18; i++)
22             System.out.print(gen.next() + " ");
23     }
```

```
24    /**
25     * 实现接口的方法，返回值为 Integer
26     * @return
27     */
28    public Integer next() {
29        return fib(count++);
30    }
31    /**
32     * 定义费波列切数
33     * @param n
34     * @return
35     */
36    private int fib(int n) {
37        if (n < 2) return 1;
38        return fib(n - 2) + fib(n - 1);
39    }
40 }
```

泛型方法的应用：

```
1  public class GenericMethods {
2
3      public static void main(String[] args) {
4          GenericMethods gm = new GenericMethods();
5          gm.f("");
6          gm.f(1);
7          gm.f(gm);
8      }
9
10     /**
11      * 定义泛型方法
12      * @param x        方法的参数为泛型
13      * @param <T>
14      */
15     public <T> void f(T x) {
16         System.out.println(x.getClass().getName());
17     }
18 }
```

回答到这里，关于泛型的问题基本上总结的差不多了。

总结

本身主要围绕开头的几个真正的面试题展开，简单来说，泛型是什么？为什么要有泛型？泛型如何实现的？泛型有哪些用处。