Java泛型

Java泛型概念

Java泛型是一种在编译时进行类型检查和类型推断的机制，它可以让我们编写更加通用、可重用的代码，提高了代码的可读性和可维护性，同时保证了类型安全。

Java泛型的核心思想是类型参数化，即在类、接口或方法的定义中使用类型参数来代替具体的类型，这些类型参数在实例化时被具体类型替换，从而实现了通用性和类型安全。

主要特性包括

1. 类型参数：在类、接口或方法的定义中使用类型参数来代替具体的类型。例如，List<E>中的E就是类型参数。
2. 类型擦除：Java泛型在编译时实现类型安全检查，但在运行时会将泛型类型的信息擦除，转换为原始类型。这是为了保持与Java早期版本的兼容性，同时减少运行时的开销。例如，List<String>在运行时会被擦除为List。
3. 上限和下限：使用通配符（?）可以指定泛型类型的上限或下限，从而限制可用的类型范围。例如，<? extends Number>表示只能使用Number及其子类类型。
4. 泛型类和泛型接口：在类或接口的定义中使用类型参数，从而实现类或接口的通用性。例如，List<E>就是一个泛型类，Comparable<T>就是一个泛型接口。
5. 泛型方法：在方法的定义中使用类型参数，从而实现方法的通用性。例如，Collections.sort(List<T>)就是一个泛型方法。

Java泛型的优点

可以提高代码的可读性和可重用性，同时保证类型安全。它可以在编译时检查类型错误，避免了运行时出现类型转换异常等问题。但是，由于Java泛型的类型擦除机制，会导致一些限制，如无法使用基本类型作为类型参数、

无法获取泛型类型的具体类型等。因此，在使用Java泛型时需要注意一些细节和限制。

示例

定义了一个Pair类，它有两个类型参数T和U，分别表示第一个元素和第二个元素的类型。Pair类有一个构造方法，可以用来创建Pair对象，并提供了获取和设置元素的方法。

Java | 复制代码

```java
public class Pair<T, U> {
    private T first;
    private U second;

    public Pair(T first, U second) {
        this.first = first;
        this.second = second;
    }

    public T getFirst() {
        return first;
    }

    public U getSecond() {
        return second;
    }

    public void setFirst(T first) {
        this.first = first;
    }

    public void setSecond(U second) {
        this.second = second;
    }

    public static <T> Pair<T, T> createPair(T first, T second) {
        return new Pair<T, T>(first, second);
    }

    public static <T extends Comparable<T>> T max(T[] array) {
        if (array == null || array.length == 0) {
            return null;
        }
        T max = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[i].compareTo(max) > 0) {
                max = array[i];
            }
        }
        return max;
    }
}
```