# 泛型到底是什么？

泛型的本质 → 类型参数化 → 要操作的数据类型 可以通过参数的形式来指定。

说人话：把数据类型变成参数。

类比函数/方法，定义时指定参数类型（形参），调用时传入具体参数实例（实参）：



# 泛型的作用：

语法糖：

可以将容易运行时出错的地方使用泛型限制变成在编译期间就可以发现的错误

如果等待在运行时出错再改

1、可能耗费更多时间和精力

1. 如果测试时没有发现错误等到上线了运行再发现就已经晚了

# 泛型规则：(Article extends Entity)

1. **边界限制**

 <T extends Entity>泛型上界

好处：可以直接调用父类或父接口的方法

1. **不变、协变、逆变**

编译器直接认为 Response<Entity> 和 Response<Article> 不存在继承关系，无法相互替代，即只能识别具体的类型，这就是泛型的不变性。

而在有些场景，这样的特性会给我们带来一些不便，可以通过型变来扩展参数的类型范围，有下面两种形式：

**① 协变：父子关系一致 → 子类也可以作为参数传进来 → <? extends Entity> → 上界通配符**  
**Tips：Kotlin中使用 out 关键字表示协变 → Response<out Entity>**

**② 逆变：父子关系颠倒 → 父类也可以作为参数传进来 → <? super Article> → 下界通配符**



**Tips：Kotlin中使用 in 关键字表示逆变 → Response<in Entity>**

**3**无限定通配符<?>

不关心泛型的具体类型时可用

**Tips：Kotlin 中使用星投影 <\*> 表示，等价于 <out Any>**

根据定义型变的位置，分为使用处型变（对象定义）和声明处型变（类定义）。

Java 只有使用处型变（例子就是），而 Kotlin 两种都有，示例如下：

// Kotlin 使用处型变

fun printArticleResponse(response: Response<in Article>) {

response.parseResponse("开始请求接口")

}

// Kotlin 声明处型变

class KtResponse<in T>(private val entity: T){

fun getOut(): T = t

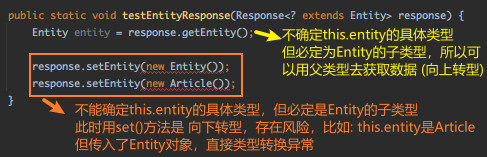
}

1. **何时用协变？何时用逆变？**

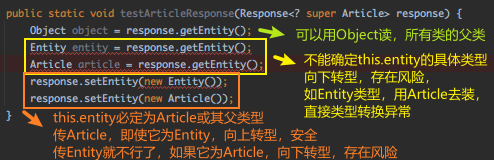
先提一嘴介个：

* 向上转型 → 子类转换成父类（隐式），安全，可以访问父类成员；
* 向下转型 → 父类转换成子类（显式），存在安全隐患，子类可能有一些父类没有的方法；

协变 → 能读不能写（能用父类型去获取数据，不确定具体类型，不能传）。



逆变 → 能写不能读（能传入子类型，不确定具体类型，不能读，但可以用Object读）。



Tips：

能向上转型但不能向下转型：

response.setEntity(new Entity)将接收的Entity对象转为response的对象

Entity entity = response.getEntity()将获取到的Entity对象转为entity对象

实在不行，那就背：PECS法则(Producer Extends，Consumer Super)

* 生产者 → extends/out → 协变 → 对象只作为返回值传出
* 消费者 → super/in → 逆变 → 对象只作为参数传入

**Tips：Kotlin 官方文档写的 Consumer in, Producer out!，好像更容易理解和记忆~**

总结：