**泛型数组**

 @author ixenos

* **先给结论**
  + 不能(直接)创建泛型数组
  + 实际的运行时对象数组只能是Object[]类型，而实际的运行时数组对象可能是T类型(虽然运行时会擦除成原始类型)
  + 一般解决方案：(泛型数组包装器):使用ArrayList收集泛型数组对象的对象元素，如ArrayList<T>、ArrayList<Pair<String>>
    - 将获得数组的行为，以及由泛型提供的编译期的类型安全

* **直接创建泛型数组不能通过编译，而转型对象数组通过编译但是不能在JVM运行**
  + - [复制代码](javascript:void(0);)
    - public class ArrayOfGeneric{
    - static Generic<Integer>[] gia;
    - @SupperssWarnings("unchecked")
    - public static void main(String[] args){
    - gia = (Generic<Integer>[])new Generic[100]; // 通过类型转换匿名对象
    - //! gia[0] = new Object(); //编译不通过，不能(直接)创建泛型数组实例
    - }

}

* + 问题在于数组将跟踪他们的实际类型，而这个类型是在数组被创建时确定的，因此，即使gia已经被转型为Generic<Integer>[]，但这个信息只存在于**编译期**(并且如果没有@SuppressWarning("unchecked")注解，将得到这个转型的警告)。在运行时，它仍旧是Object数组
  + 因此，成功创建泛型数组的唯一方式就是创建一个被擦出类型的新数组，然后对其转型（而且是在运行时转型）
    - 直接对整个数组强制转型，在编译时依旧会被擦除掉类型！所以应该在运行时转型，而这时最好的办法就是使用一个**泛型数组包装器**，维护一个原始类型的数组，通过数组入口方法进行元素**编译期的类型安全检测**(对应返回值)和强制类型转换(对于运行时不重要)，从而保证类型安全。

* + 对整个数组强制转型的例子(错误方法)
  + public class GenericArray<T> {
  + private T[] array;
  + @SupperessWarning("unchecked")
  + public GenericArray(int sz) {
  + array = (T[]) new Object[sz];
  + }
  + public void put(int index, T item) {
  + array[index] = item;
  + }
  + public T get(int index) { return array[index]; }
  + public T[] rep() { return array; } //应该在运行时出口做文章
  + public static void main (String[] args){
  + GenericArray<Integer> gai = new GenericArray<Integer>(10);
  + // Integer[] ia = gai.rep(); //ClassCastException

Object[] oa = gai.rep(); //只能返回对象数组类型为Object[]

* + - 实际的运行时对象数组是Object[]，而实际的运行时数组对象可能是T类型。

* + 因此，应该在运行时，数组对象的出口做转型输出，入口方法在编译期已实现类型安全，所以出口方法可以放心强制类型转换，保证成功。如下
    - public class GenericArray2<T> {
    - private Object[] array; //维护Object[]类型数组
    - @SupperessWarning("unchecked")
    - public GenericArray2(int sz) {
    - array = new Object[sz];
    - }
    - public void put(int index, T item) {
    - array[index] = item;
    - }
    - public T get(int index) { return (T)array[index]; }//数组对象出口强转
    - public T[] rep() { return (T[])array; } //运行时无论怎样都是Object[]类型
    - public static void main (String[] args){
    - GenericArray<Integer> gai = new GenericArray<Integer>(10);
    - // Integer[] ia = gai.rep(); //依旧ClassCastException
    - Object[] oa = gai.rep(); //只能返回对象数组类型为Object[]
    - gai.put(0,11);
    - System.out.println(gai.get(0)); // 11 ,出口成功转型
    - }

}

* **通过反射在运行时构出实际类型为type[]的对象数组，避免了类型擦除，从而转换成功，无ClassCastException**

import java.lang.reflect.\*;

public class GenericArrayWithTypeToken<T> {

private T[] array;

@SuppressWarning("unchecked")

public GenericArrayWithTypeToken(Class<T> type, int sz) {

array = (T[]) Array.newInstance(type, sz);//通过反射在运行时构出实际类型为type[]的对象数组，避免了类型擦除，从而转换成功，无ClassCastException

}

public void put(int index, T item){

array[index] = item;

}

public T get(int index) { return array[index]; }

public T[] rep() { return array; } //能成功返回了~

public static void main(String[] args) {

GenericArrayWithTypeToken<Integer> gawtt = new GenericArrayWithTypeToken<>(Integer.class, 10);

Integer[] ia = gawtt.rep(); //能成功返回了！

}

}

* ArrayList类中对象数组的运行时类是Object[]，是历史遗留问题，能做得更好，但是Java类库开发人员囿于现有接口的限定，无法修改某些Java类库代码