泛型

引入泛型

思考问题

> 如果我们需要产生多个对象，每个对象的逻辑完全一样，只是对象内成员变量的类型不同，应该怎么去实现？

方案一

创建多个类，为每个类的成员变量设置不同的类型

```
class MyClass1 {
    private int data;

    public void setData(int data) {
        this.data = data;
    }

    public int getData() {
        return data;
    }
}

class MyClass2 {
    private String data;

    public void setData(String data) {
        this.data = data;
    }

    public String getData() {
        return data;
    }
}
```

例子

示例代码

```
MyClass1 myClass1 = new MyClass1();
myClass1.setData(10);
int data = myClass1.getData();
System.out.println(data);

MyClass2 myClass2 = new MyClass2();
myClass2.setData("字符串");
String data2 = myClass2.getData();
System.out.println(data2);
```

执行结果

```
10
字符串
```

此时，我们发现如果需要增加一个不同类型的对象，就需要增加一个类。会导致类的膨胀，代码重用性太差。

方案二

创建一个类文件，给这个类中的成员变量设置Object数据类型

```
class MyClass {
    private Object data;

    public void setData(Object data) {
        this.data = data;
    }

    public Object getData() {
        return data;
    }
}
```

缺点： 编译的时候没有问题，但是运行的时候肯出现异常

例子

示例代码

```
MyClass myClass = new MyClass();
myClass.setData(10);
Object data = myClass.getData();
String str = (String) data;
System.out.println(str);
```

执行结果

```
Exception in thread "main" java.lang.ClassCastException: java.lang.Integer cannot be cast to
java.lang.String
        at com.oaec.genericdemo.Demo2.main(Demo2.java:9)
```

泛型（JDK1.5开始添加此特性）

概述

参数化数据类型，在声明类的时候，不明确指出需要什么样的类型，而是加一个标记，该标记代表类型，使用此类的时候才有具体类型

第一个泛型类

语法

```
class 类名<泛型参数[,泛型参数2，泛型参数3]>{
 //使用T作为类型
}
```

例子

示例代码

```
class GenClass1<T>{//T-->Type
    private T data;

    public void setData(T data) {
        this.data = data;
    }
    public T getData() {
        return data;
    }
}
```

使用

```
GenClass1<String> gc1 = new GenClass1<String>();
gc1.setData("字符串");
String data = gc1.getData();
System.out.println(data.substring(0, 2));
```

执行结果

```
字符
```

分析

此时发现，使用泛型之后，泛型参数一旦被设置为具体类型，就只能使用该类型，不需要类型转换，也就避免了 `ClassCastException`

泛型的类型参数可以是泛型类

示例代码

```java
/**
 * 泛型的类型参数可以是泛型类
 *
 * @author Kevin
 *
 */
public class GenericDemo3 {

    public static void main(String[] args) {
        A<String> a = new A<String>();
        a.setData("字符串");
        //B类的类型参数是A类，A类是泛型类
        B<A<String>> b = new B<A<String>>();
        b.setData(a);
        A<String> data = b.getData();
        String data2 = data.getData();
        System.out.println(data2.length());
    }
}

class A<T> {
    private T data;

    public void setData(T data) {
        this.data = data;
    }

    public T getData() {
        return data;
    }
}

class B<T> {
    private T data;

    public void setData(T data) {
        this.data = data;
    }

    public T getData() {
        return data;
    }
}
```

泛型类可以同时设置多个类型参数

示例代码

```java
/**
 * 泛型类可以同时设置多个类型参数
 *
 * @author Kevin
 *
 */
public class GenericDemo4 {

    public static void main(String[] args) {
        //从JDK1.7开始，只设置声明类时的类型参数就可以
        GenClass3<String,Integer> gc3 = new GenClass3<String,Integer>();
        gc3.setData1("字符串");
        gc3.setData2(10);
        System.out.println(gc3.getData1());
        System.out.println(gc3.getData2());
    }
}

class GenClass3<T1, T2> {
    private T1 data1;
    private T2 data2;

    public void setData1(T1 data1) {
        this.data1 = data1;
    }

    public void setData2(T2 data2) {
        this.data2 = data2;
    }

    public T1 getData1() {
        return data1;
    }

    public T2 getData2() {
        return data2;
    }
}
```

执行结果

```
字符串
10
```

泛型类可以继承泛型类

示例代码

```java
/**
 * 泛型类可以继承泛型类
 *
 * @author Kevin
 *
 */
public class GenericDemo5 {

    public static void main(String[] args) {
        D<String> d = new D<String>();
        d.setData("字符串");

        E e = new E();
        e.setData("字符串");
    }
}
class C<T>{
    private T data;

    public void setData(T data) {
        this.data = data;
    }
    public T getData() {
        return data;
    }
}
class D<T> extends C<T>{
    //当子类还不确定需要什么类型的数据，可以将子类也定义为泛型类
}
class E extends C<String>{
    //当子类已经确定所需要的数据类型，可以为父类指定该具体类型，子类无需定义为泛型类
}
```

执行结果

```
字符串
10
```

泛型类可以实现泛型接口

示例代码

```java
/**
 * 泛型类可以实现泛型接口
 * @author Kevin
 */
public class GenericDemo6 {

    public static void main(String[] args) {
        H<Integer> h = new H<Integer>();
        h.show(10);

        I i = new I();
        i.show("字符串");
    }
}
interface G<T>{
    void show(T data);
}
class H<T> implements G<T>{

    @Override
    public void show(T data) {
        System.out.println(data);
    }

}
class I implements G<String>{

    @Override
    public void show(String data) {
        System.out.println(data);
    }

}
```

执行结果

```
10
字符串
```

限制泛型类型

示例代码

```java
/**
 * 限制泛型类型
 *
 * @author Kevin
 *
 */
public class GenericDemo2 {

    public static void main(String[] args) {
        GenClass2<Dog> gc2 = new GenClass2<Dog>();
        Dog data = new Dog();
        gc2.setData(data);
        gc2.show();

        GenClass2<Cat> gcCat = new GenClass2<Cat>();
        Cat cat = new Cat();
        gcCat.setData(cat);
        gcCat.show();
    }

}
/**
 * 此类中类型参数只能给Animal类型或者其子类类型
 * @author Kevin
 *
 * @param <T>
 */
class GenClass2<T extends Animal>{
    private T data;
    public void setData(T data) {
        this.data = data;
    }

    public void show(){
        data.eat();
    }
    /*public T getData() {
        return data;
    }*/
}
abstract class Animal{
    public abstract void eat();
}
class Dog extends Animal{

    @Override
    public void eat() {
        System.out.println("啃骨头");
    }

}
class Cat extends Animal{
```

```
    @Override
    public void eat() {
        System.out.println("猫吃鱼");
    }

}
```

执行结果

```
啃骨头
猫吃鱼
```

通配符

概述

1. 同一泛型类，如果实例化时给定的实际类型不同，则这些实例的类型是不兼容的，不能相互赋值。
2. 泛型类实例之间的不兼容性会带来使用的不便。我们可以使用泛型通配符(?)声明泛型类的变量就可以解决这个问题。
3. ? 通配符，只能取值，不可以修改值

例子

示例代码

```java
/**
 * 通配符
 *
 * @author Kevin
 *
 */
public class GenericDemo7 {

    public static void main(String[] args) {
        GenClass4<String> genClass1 = new GenClass4<String>();
        genClass1.setData("字符串");
        show(genClass1);
        GenClass4<Integer> genClass2 = new GenClass4<Integer>();
        genClass2.setData(10);
        show(genClass2);
        GenClass4<Double> genClass3 = new GenClass4<Double>();
        genClass3.setData(10.5);
        show(genClass3);
    }

    public static void show(GenClass4<?> genClass4){
        // ? 通配符，只能取值，不可以修改值
//      genClass4.setData(10.6);
        System.out.println(genClass4.getData());
    }

}

class GenClass4<T> {
    private T data;

    public void setData(T data) {
        this.data = data;
    }

    public T getData() {
        return data;
    }
}
```

执行结果

```
字符串
10
10.5
```