型別本來有：簡單型別和複雜型別，引入泛型後把複雜型別分的更細了.

概述

泛型是Java SE 1.5的新特性，泛型的本質是引數化型別，也就是說所操作的資料型別被指定為一個引數。這種引數型別可以用在類、介面和方法的建立中，分別稱為泛型類、泛型介面、泛型方法。Java語言引入泛型的好處是安全簡單。

在Java SE 1.5之前，沒有泛型的情況的下，通過對型別Object的引用來實現引數的“任意化”，“任意化”帶來的缺點是要做顯式的強制型別轉換，而這種轉換是要求開發者對實際引數型別可以預知的情況下進行的。對於強制型別轉換錯誤的情況，編譯器可能不提示錯誤，在執行的時候才出現異常，這是一個安全隱患。

泛型的好處是在編譯的時候檢查型別安全，並且所有的強制轉換都是自動和隱式的，以提高程式碼的重用率。

泛型的規則限制

泛型的型別引數只能是類型別（包括自定義類），不能是簡單型別。

同一種泛型可以對應多個版本（因為引數型別是不確定的），不同版本的泛型類例項是不相容的。

泛型的型別引數可以有多個。

泛型的引數型別可以使用extends語句，例如。習慣上稱為“有界型別”。

泛型的引數型別還可以是萬用字元型別。例如Class<?> classType = Class.forName(“java.lang.String”);

1、具體例子

下面給出兩個簡單的例子，實現同樣的功能，一個使用了泛型，一個沒有使用泛型。

例子一：使用了泛型

```
public class Gen<T> {
private T t;
public Gen(T t){
this.t = t;
}
public T getT() {
return t;
}
public void setT(T t) {
this.t = t;
}
public void showType(){
```

```
System.out.println("T的實際型別是："   t.getClass().getName());
}
public static void main(String[] args) {
Gen<Integer> gen = new Gen<Integer>(1);
gen.showType();
int i = gen.getT();
System.out.println(" value = "   i);
System.out.println(" ====================== ");
//定義泛型類Gen的一個String的版本
Gen<String>strObj = new Gen<String>("Hello Gen!");
strObj.showType();
String s = strObj.getT();
System.out.println(" value = "   s);
}
}
```

例子二：沒有使用泛型

```
public class Gen2 {
// 定義一個通用型別成員
private Object obj;
public Gen2(Object obj) {
this.obj = obj;
}
public Object getObj() {
return obj;
}
public void setObj(Object obj) {
this.obj = obj;
}
public void showType() {
System.out.println("T的實際型別是: "   obj.getClass().getName());
}
public static void main(String[] args) {
// 定義類Gen2的一個Integer版本
Gen2 intObj = new Gen2(2);
intObj.showType();
int i = (Integer) intObj.getObj();
System.out.println(" value = "   i);
System.out.println(" ====================== ");
// 定義類Gen2的一個String版本
Gen2 strOb = new Gen2("Hello Gen!");
strOb.showType();
```

```
String s = (String) strOb.getObj();
System.out.println(" value= "   s);
}
}
```

2、深入泛型

在Java 5之前，為了讓類有通用性，往往將引數型別、返回型別設定為Object型別，當獲取這些返回型別來使用時候，必須將其“強制”轉換為原有的型別或者介面，然後才可以呼叫物件上的方法。

泛型和使用“Object泛型”方式實現結果的完全一樣，但是簡單多了，因為不需要強制型別轉換。

泛型類語法：

使用來宣告一個型別持有者名稱，然後就可以把T當作一個型別代表來宣告成員、引數和返回值型別。當然T僅僅是個名字，這個名字可以自行定義。

class GenericsFoo 宣告瞭一個泛型類，這個T沒有任何限制，實際上相當於Object型別，實際上相當於 class GenericsFoo。

與Object泛型類相比，使用泛型所定義的類在宣告和構造例項的時候，可以使用“<實際型別>”來一併指定泛型型別持有者的真實型別。例如：

```
GenericsFoo<Double> douFoo=new GenericsFoo<Double>(new Double("33
```

當然，也可以在構造物件的時候不使用尖括號指定泛型型別的真實型別，但是你在使用該物件的時候，就需要強制轉換了。比如：

```
GenericsFoo douFoo=new GenericsFoo(new Double("33"));
```

實際上，當構造物件時不指定型別資訊的時候，預設會使用Object型別，這也是要強制轉換的原因。

3、高階應用

限制泛型

在上面的例子中，由於沒有限制class GenericsFoo型別持有者T的範圍，實際上這裡的限定型別相當於Object，這和“Object泛型”實質是一樣的。限制比如我們要限制T為集合介面型別。只需要這麼做：
class GenericsFoo，這樣類中的泛型T只能是Collection介面的實現類，傳入非Collection介面編譯會出錯。

多介面限制

雖然Java泛型簡單的用 extends 統一的表示了原有的 extends 和 implements 的概念，但仍要遵循應用的體系，Java 只能繼承一個類，但可以實現多個介面，所以你的某個型別需要用 extends 限定，且有多種型別的時候，只能存在一個是類，並且類寫在第一位，介面列在後面，也就是：

（泛型方法的型別限定）

```
<T extends SomeClass & interface1 & interface2 & interface3>
```

（泛型類中型別引數的限制）

```
public class Demo<T extends Comparable & Serializable> {
// T型別就可以用Comparable宣告的方法和Seriablizable所擁有的特性了
}
```

萬用字元泛型

為了解決型別被限制死了不能動態根據例項來確定的缺點，引入了“萬用字元泛型”，針對上面的例子，使用通配泛型格式為<? extends Collection>，“？”代表未知型別，這個型別是實現Collection介面。

注意：

如果只指定了<?>，而沒有extends，則預設是允許Object及其下的任何Java類了。也就是任意類。

萬用字元泛型不單可以向下限制，如<? extends Collection>，還可以向上限制，如<? super Double>，表示型別只能接受Double及其上層父類型別，如Number、Object型別的例項。

泛型類定義可以有多個泛型引數，中間用逗號隔開，還可以定義泛型介面，泛型方法。這些都與泛型類中泛型的使用規則類似。

4、泛型方法

是否擁有泛型方法，與其所在的類是否泛型沒有關係。要定義泛型方法，只需將泛型引數列表置於返回值前。如:

```
public class GenericMethod {
public <T> void print(T x) {
System.out.println(x.getClass().getName());
}
public static void main(String[] args) {
GenericMethod method = new GenericMethod();
method.print(" ");
method.print(10);
method.print('a');
method.print(method);
}
}
```

需要注意的是，一個static方法，無法訪問泛型類的型別引數，所以，若要static方法需要使用泛型能力，必須使其成為泛型方法。

泛型的好處如：

開始版本

```
public void write(Integer i, Integer[] ia);
public void write(Double d, Double[] da);
```

泛型版本

```
public <T> void write(T t, T[] ta);
```

簡便了程式碼

定義泛型

定義在類後面

緊跟類名後面

```
public class TestClassDefine<T, S extends T>{......}
```

定義泛型 T, S, 且S 繼承 T

定義在方法裝飾符後面

緊跟修飾符後面（public）

```
public <T, S extends T> T testGenericMethodDefine(T t, S s){.....
```

定義泛型 T, S, 且S 繼承 T

例項化泛型

例項化定義在類上的泛型

第一宣告類變數或者例項化時。例如

```
List<String> list;
list = new ArrayList<String>;
```

第二繼承類或者實現介面時。例如

```
public class MyList<E> extends ArrayList<E> implements List<E> {.
```

例項化定義方法上的泛型

當呼叫範型方法時，編譯器自動對型別引數(泛型)進行賦值，當不能成功賦值時報編譯錯誤。

萬用字元(?)

上面有泛型的定義和賦值；當在賦值的時候，上面一節說賦值的都是為具體型別，當賦值的型別不確定的時候，我們用萬用字元(?)代替了：

```
List<?> unknownList;
List<? extends Number> unknownNumberList;
List<? super Integer> unknownBaseLineIntgerList;
```