类的创建有很多方式，并不仅仅局限于普通的创建，内部类顾名思义，就是创建在内部的类，那具体是什么的内部呢？
*注意：有点绕

- *成员内部类（非**static**）*：

我们可以直接在类的内部定义成员内部类（既然是成员，所以是非**static**的）
如：

```
public class Test {
public class Inner { //内部类也是类，所以说里面也可以有成员变量、方法等，甚至还可以继续套娃一个成员内部类
public void test(){
System.out.println("我是成员内部类！");
}
}
}
```

**注意：
成员内部类和成员方法，成员变量一样（非**static**），是对象所有的，如果我们要使用成员内部类，那么就需要：

```
public static void main(String[] args) {
Test test = new Test(); //我们首先需要创建外部类对象
Test.Inner inner = test.new Inner(); //成员内部类的类型名称就是 外部类对象.内部类名称
}————————注意这里的写法，是 外部类+内部类+ 内部类对象名 + = + 外部类对象+ . +new +内部类
//通过外部类的对象，来创建内部类的对象
```

创建完内部类的对象后就可以使用内部类里面的方法了（要注意不要是**private**的，支持访问权限控制）

比如：

```
public class Test {
class Inner{
void test(){
System.out.println("我是内部类方法");
}
}
}
```

```
public static void main(String[] args) {
Test test = new Test(); //我们首先需要创建外部类对象
Test.Inner inner = test.new Inner(); //成员内部类的类型名称就是 外部类对象.内部类名称
inner.test(); //这里调用了内部类里面的方法
}
```

结果为：我是内部类方法

**由外部类创建的所有对象，都有内部类，而且每个对象的内部类都不同，就是说，每个类可以创建一个对象，而每个对象种都有一个单独的类定义（内部类），可以通过这个成员内部类创建出更多的对象

---

*我们说了方法权限不能改为**private**，那内部类的权限改为**private**，方法还是**public**会怎么样呢？

```
public class Test {
private class Inner{
public void test(){
System.out.println("我是内部类方法");
}
}
}

public static void main(String[] args) {
Test test = new Test();
Test.Inner inner = test.new Inner();//报错，找不到了Inner,想要用，只能改为public
inner.test();
}
```

这里我们需要特别注意一下，在成员内部类中，是可以访问到外层的变量的：

```
public class Test {
private final String name;
public Test(String name){
this.name = name;
}
public class Inner {
public void test(){
System.out.println("我是成员内部类："+name);
//成员内部类可以访问到外部的成员变量
//因为成员内部类本身就是某个对象所有的，每个对象都有这样的一个类定义，这里的name是其所依附对象的
}
}
}

public class Main {
public static void main(String[] args) {
Test test=new Test("小明");
Test.Inner inner =test.new Inner();
inner.test();
}
}
```

结果为：我是成员内部类：小明 //所以说，内部类是可以访问到外部类的

**一句话概括：内部类可以访问到外部类的东西，但是外部类不能访问到成员内部类的东西（注意用词，是成员内部类，如果是后面的静态内部类 ，那么可以访问到静态内部类里的静态属性） （主要看作用域）

为什么外部类，不能访问成员内部类的静态属性呢？因为在java8里面，成员内部类里的不允许出现静态属性的，所以说，连成员内部类里面连静态属性都没有，怎么访问成员内部类的静态属性呢

当我们在创建几个内部类对象时：

```
Test test1=new Test("小明");//外部类小明
Test.Inner inner1 =test1.new Inner();//依赋于外部类小明来创建的内部类对象
inner1.test();
```

```
    Test.Inner inner3 =test1.new Inner();//依赋于外部类小明来创建的内部类对象
    inner3.test();


    Test test2=new Test("小红");
    Test.Inner inner2  =test2.new Inner();//依赋于外部类小红来创建的内部类对象
    inner2.test();
```

结果为：
我是内部类方法小明
我是内部类方法小明
我是内部类方法小红

可以看出，是依附于那个外部类对象的，就是哪个对象的

当内部类与外部类里有同名变量时，我们要怎样去明确使用哪一个呢？

```
public class Test {
private final String name;
public Test(String name){
this.name = name;
}
public class Inner {
String name;
public void test(String name){
System.out.println("方法参数的name = "+name);
//依然是就近原则，最近的是参数，那 就是参数了
System.out.println("成员内部类的name = "+this.name);
//在内部类中使用this关键字， 只能表示内部类对象
System.out.println("成员内部类的name = "+Test.this.name);
//如果需要指定为外部的对象，那么需要在前面添加外部类型名称
}
}
}
```

在main中写入：

```
Test t=new Test("小明");
Test.Inner inner=t.new Inner();
inner.name="小刚";
inner.test("小红");
```

结果为：

```
方法参数的name = 小红
成员内部类的name = 小刚
成员内部类的name = 小明
```

包括对方法的调用和super关键字的使用，也是一样的：

```
public class Inner {
String name;
public void test(String name){
this.toString(); //内部类自己的toString方法
super.toString(); //内部类父类的toString方法
Test.this.toString(); //外部类的toSrting方法
Test.super.toString(); //外部类父类的toString方法
}
}
```

**所以说成员内部类其实在某些情况下使用起来比较麻烦，对于这种成员内部类，我们一般只会在类的内部自己使用

_____

- 静态内部类：有static

前面我们介绍了成员内部类，它就像成员变量和成员方法一样，是属于对象的，同样的，静态内部类就像静态方法和静态变量一样，是属于类的，我们可以直接创建使用。

与静态成员变量和方法一样。它是属于类(属于外部类)的，也就是说它可以由外部类直接访问，而不用创建外部类的对象，再使用内部类

如：

```
public static class Inner{//Inner为静态内部类
}
```

你可以直接创建对象了，而不需要像成员内部类一样，先创建外部类对象，在通过外部类对象创建内部类对象

```
Test.Inner inner=new Test.Inner();//不需要依附外部类对象，直接创建内部类对象
```

静态内部类由于是静态的，所以相对外部来说，整个内部类中都处于静态上下文（注意只是相当于外部来说）是无法访问到外部类的非静态内容的

**外部类只能访问静态内部类的静态属性（注意用词，是静态内部类，不是成员内部类），其他的成员属性不行

*以上一句话概括：静态类内部不能访问外部类的非静态属性，只能访问静态属性（就是之前说的静态内容只能访问静态内容）
只不过受影响的只是外部内容的使用，内部倒是不受影响，还是跟普通的类一样：
内部（就是它自己本身）可以随便用自己的东西

如果内部类里也有static方法(而且外部类也可以直接访问静态属性)

```
public static class Inner{
public static void test1(){}
}
```

你也可以直接通过类来访问
如：Test.Inner.test1();

静态内部类加载也是一样的：
.class文件是给jvm去执行的，而每一个.class文件就是类，
java中使用一个类之前，jvm不会在一开始就去加载它，而是在用到的时候就会去加载类：

1.访问静态变量，或者为静态变量赋值,调用静态方法
2.new 创建类的对象（隐式加载）
3.子类初始化
4.其他

来各例子：

```
public class Test {

static{
System.out.println("我是外部类初始化");
}

public static class Inner{

    static{
        System.out.println("静态内部类初始化");
    }

    public static void test(){
        System.out.println("我是静态内部类方法");
    }

}
}
```

当我们在main中只调用test()方法：

```
Test.Inner.test();
```

结果会是什么?
结果为:
静态内部类初始化
我是静态内部类方法

//可以看到并没有初始化外部类 只有静态内部类初始化了
为什么会这样呢?

当我们打开反编译文件，发现jvm弄了两个class文件
一个是Test.class
另一个是Test$Inner.class
所以说两个类（外部类和静态内部类）是分别编译的

因为上面我们只调用了静态内部类的方法，所以只有静态内部类加载了，而外部类不会加载，只有当我们使用外部类里面的内容时，才会加载外部类

- *局部内部类（定义在方法里面的类，它的作用域只能在方法的花括号里）****

局部内部类，就像局部变量一样，可以在方法里定义
定义在方法中的类
如

```
public void test(){
class Inner{//Inner的作用域仅限在test()方法里，所以不能加权限修饰符，static
}

}
```

这样Inner的作用域只能在这个test方法中 不能加权限修饰符（因为方法中的东西只能在方法中有用，不能加权限修饰符，也不能用static）

这种局部内部类的形式（仅限于在方法里的局部内部类），使用频率很低，基本上不会用到，所以了解就行
