static class 静态类（Java）

> 一般情况下是不可以用**static**修饰类的。如果一定要用static修饰类的话，通常**static**修饰的是匿名内部类。
>
> 在一个类中创建另外一个类，叫做成员内部类。这个成员内部类可以静态的（利用**static**关键字修饰），也可以是非静态的。由于静态的内部类在定义、使用的时候会有种种的限制。所以在实际工作中用到的并不多。
>
> 在开发过程中，内部类中使用的最多的还是非静态地成员内部类。不过在特定的情况下，静态内部类也能够发挥其独特的作用。

一、静态内部类的使用目的。

在定义内部类的时候，可以在其前面加上一个权限修饰符static。此时这个内部类就变为了静态内部类。不过由于种种的原因，如使用上的限制等等因素(具体的使用限制，笔者在下面的内容中会详细阐述)，在实际工作中用的并不是很多。但是并不是说其没有价值。在某些特殊的情况下，少了这个静态内部类还真是不行。如在进行代码程序测试的时候，如果在每一个Java源文件中都设置一个主方法(主方法是某个应用程序的入口，必须具有)，那么会出现很多额外的代码。而且**最主要的时这段主程序的代码对于Java文件来说，只是一个形式，其本身并不需要这种主方法。但是少了这个主方法又是万万不行的。在这种情况下，就可以将主方法写入到静态内部类中，从而不用为每个Java源文件都设置一个类似的主方法。**这对于代码测试是非常有用的。在一些中大型的应用程序开发中，则是一个常用的技术手段。为此，这个静态内部类虽然不怎么常用，但是程序开发人员还必须要掌握它。也许在某个关键的时刻，其还可以发挥巨大的作用也说不定。

```java
package com.fei.stackandqueue;

/**
 * @author Author  知秋
 * @email fei6751803@163.com
 * @time Created by Auser on 2017/8/24 17:58.
 */
public class MainInStaticClass {

    static class Main {
        static String aaa = "我是内部来的";

        static void main() {
//将主方法写到静态内部类中，从而不必为每个源文件都这种一个类似的主方法
            new MainInStaticClass().print();
        }
    }
    /**
     * 运行结果:
     * main in static inner class
     * 我是内部来的
     */
    public static void main(String[] args) {
        new MainInStaticClass().print();
        System.out.println(Main.aaa);
    }

    public void print() {
        System.out.println("main in static inner class");
    }
}


class TestMain {

    /**
     * 运行结果:
     * main in static inner class
     * 我是内部来的
     */
    public static void main(String[] args) {
// TODO Auto-generated method stub
// new MainInStaticClass().print();
        MainInStaticClass.Main.main();
        new MainInStaticClass.Main();
        System.out.println(MainInStaticClass.Main.aaa);
    }
}
```

二、静态内部类的使用限制。

将某个内部类定义为静态类，跟将其他类定义为静态类的方法基本相同，引用规则也基本一致。不过其细节方面仍然有很大的不同。具体来说，主要有如下几个地方要引起各位程序开发人员的注意。　　一是静态成员(包括静态变量与静态成员)的定义。一般情况下，**如果一个内部类不是被定义成静态内部类，那么在定义成员变量或者成员方法的时候，是不能够被定义成静态成员变量与静态成员方法的(静态成员针对于类来讲的，非静态类无法通过外部类直接拿到，如对于一个静态内部类里的一个静态字符串变量字段`ccc`而言，想从外面直接拿到，可以通过Aaa.Bbb.ccc)**。也就是说，在非静态内部类中不可以声明静态成员。如现在在一个student类中定义了一个内部类age，如果没有将这个类利用static关键字修饰，即没有定义为静态类，那么在这个内部类中如果要利用static关键字来修饰某个成员方法或者成员变量是不允许的。在编译的时候就通不过。故程序开发人员需要注意，**只有将某个内部类修饰为静态类，然后才能够在这个类中定义静态的成员变量与成员方法。这是静态内部类都有的一个特性**。也正是因为这个原因，有时候少了这个静态的内部类，很多工作就无法完成。或者说要绕一个大圈才能够实现某个用户的需求。这也是静态的内部类之所以要存在的一个重要原因。　　二是在成员的引用上，有比较大的限制。**一般的非静态内部类，可以随意的访问外部类中的成员变量与成员方法。即使这些成员方法被修饰为private(私有的成员变量或者方法)，其非静态内部类都可以随意的访问。则是非静态内部类的特权。因为在其他类中是无法访问被定义为私有的成员变量或则方法。**但是如果一个内部类被定义为静态的，那么在银用外部类的成员方法或则成员变量的时候，就会有诸多的限制。如**不能够从静态内部类的对象中访问外部类的非静态成员(包括成员变量与成员方法)**。这是什么意思呢?如果在外部类中定义了两个变量，一个是非静态的变量，一个是静态的变量。那么在静态内部类中，无论在成员方法内部还是在其他地方，都只能够引用外部类中的静态的变量，而不能够访问非静态的变量。**在静态内部类中，可以定义静态的方法(也只有在静态的内部类中可以定义静态的方法)，在静态方法中引用外部类的成员。**但是无论在内部类的什么地方引用，有一个共同点，**即都只能够引用外部类中的静态成员方法或者成员变量。**对于那些非静态的成员变量与成员方法，在静态内部类中是无法访问的**(因为那些非静态成员想要访问需要依赖于外部类的实例，而假如你通过类字节码来直接new这个静态内部类的时候，就会出现问题，所以必须不能访问这些非静态成员变量)**。这就是静态内部类的最大使用限制。在普通的非静态内部类中是没有这个限制的。也正是这个原因，决定了静态内部类只应用在一些特定的场合。其应用范围远远没有像非静态的内部类那样广泛。　　三是**在创建静态内部类时不需要将静态内部类的实例绑定在外部类的实例上。**通常情况下，**在一个类中创建成员内部类的时候，有一个强制性的规定，即内部类的实例一定要绑定在外部类的实例中。**也就是说，在创建内部类之前要先在外部类中要利用new关键字来创建这个内部类的对象。如此的话如果从外部类中初始化一个内部类对象，那么内部类对象就会绑定在外部类对象上。也就是说，**普通非静态内部类的对象是依附在外部类对象之中的。**但是，如果成员开发人员创建的时静态内部类，那么这就又另当别论了。通常情况下，程序员在定义静态内部类的时候，是不需要定义绑定在外部类的实例上的。也就是说，要在一个外部类中定义一个静态的内部类，不需要利用关键字new来创建内部类的实例。即**在创建静态类内部对象时，不需要其外部类的对象。**

```
new MainInStaticClass.Main();
```

具体为什么会这样，一般程序开发人员不需要了解这么深入，只需要记住有这个规则即可。在定义静态内部类的时候，千万不要犯画蛇添足的错误。　　从以上的分析中可以看出，静态内部类与非静态的内部类还是有很大的不同的。一般程序开发人员可以这么理解，**非静态的内部类对象隐式地在外部类中保存了一个引用，指向创建它的外部类对象。**不管这么理解，程序开发人员都需要牢记静态内部类与非静态内部类的差异。如是否可以创建静态的成员方法与成员变量(静态内部类可以创建静态的成员而非静态的内部类不可以)、对于访问外部类的成员的限制(静态内部类只可以访问外部类中的静态成员变量与成员方法而非静态的内部类即可以访问静态的也可以访问非静态的外部类成员方法与成员变量)。这两个差异是静态内部类与非静态外部类最大的差异，也是静态内部类之所以存在的原因。了解了这个差异之后，程序开发人员还需要知道，在什么情况下该使用静态内部类。如在程序测试的时候，为了避免在各个Java源文件中书写主方法的代码，可以将主方法写入到静态内部类中，以减少代码的书写量，让代码更加的简洁。

总之，静态内部类在Java语言中是一个很特殊的类，跟普通的静态类以及非静态的内部类都有很大的差异。作为程序开发人员，必须要知道他们之间的差异，并在实际工作中在合适的地方采用合适的类。不过总的来说，静态内部类的使用频率并不是很高。但是在有一些场合，如果没有这个内部静态类的话，可能会起到事倍功半的反面效果