初识引用类型

本节目标

> 1. 能区分引用和对象
> 2. 能理解引用和对象的关系
> 3. 能理解通过引用操作对象的"共享"特性

1. 初步认识引用（reference）和对象（object）

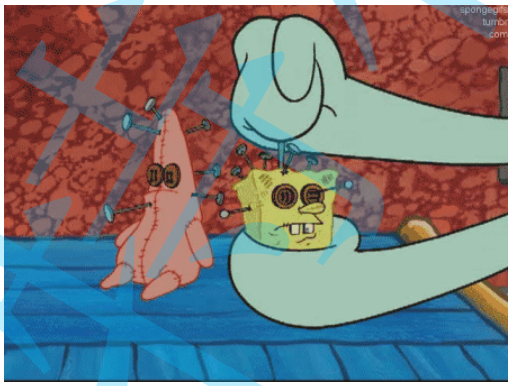
到目前为止，我们已经接触了两类对象，分别是 `String 对象` 和 `数组对象`.

```
String s = "Hello World";    // 这里的 s 的类型是 String 类型的引用；该引用指向了一个 String 类型的对象。
```

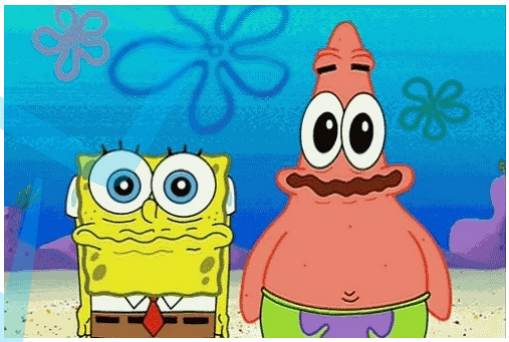
```
int[] a = { 1, 2, 3, 4, 5 };    // 这里的 a 的类型是 int[] 类型的引用；该引用指向了一个元素类型是 int 的数组类型对象。
```

2. 理解引用和对象之间的关系

引用：

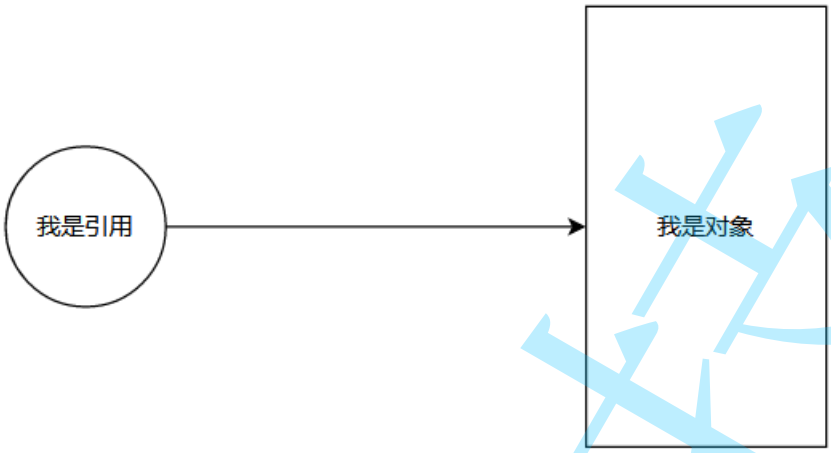


对象：



通过引用可以对对象施加影响。我们通常把这个关系称为，该引用指向该对象。

3. 课件中绘制引用和对象。



4. 关于引用指向对象的一些规则

1. 只有引用指向对象；没有对象指向引用，也没有引用指向引用，更没有对象指向对象。
2. 对象可以被多个引用指向。
3. 操作引用，其实操作的是引用指向的对象。

5. null 的理解

```
int[] a = null;      // a 不指向任何的对象
```

通常把这种情况称为：a 等于空，或者 a 等于 null。表达的意思是 a 这个引用**不指向**任何对象。

6. 引用的赋值操作符理解

引用的赋值运算，是让被赋值的引用指向该引用当前指向的对象。

```
int[] a = { 1, 2, 3, 4, 5 };
int[] b = null;

b = a;  // 让 b 指向 a 目前指向的对象
```

```
int[] a = null;      //  a 不指向任何的对象
int[] b = { 1, 2, 3, 4, 5 };
b = a;               // 让 b 指向 a 目前指向的对象
                     // 由于 a 目前不指向任何的对象。所以 b 也不再指向任何的对象
```

7. 引用的比较操作符理解

引用的相等与不等比较，是比较两个引用是否指向同一个对象。

```
int[] a = { 1, 2, 3, 4, 5 };
int[] b = { 1, 2, 3, 4, 5 };
```

```
a == b;      // 由于 a 和 b 指向不同的对象，所以结果是 false
a != b;      // 由于 a 和 b 指向不同的对象，所以结果是 true

b = a;       // 让 b 指向 a 目前指向的对象
a == b;      // 由于 a 和 b 指向相同的对象，所以结果是 true
a != b;      // 由于 a 和 b 指向相同的对象，所以结果是 false

a = null;
a == b;      // 由于 a 不指向对象，b 指向对象，所以结果是 false
a != b;      // 由于 a 不指向对象，b 指向对象，所以结果是 true

b = null;
a == b;      // 由于 a 和 b 都不指向对象，所以结果是 true
a != b;      // 由于 a 和 b 都不指向对象，所以结果是 false
```

8. 引用的 . 操作符理解

引用的 . 操作符，可以当成"的"理解，实际就是通过引用访问对象。

```
int[] a = { 1, 2, 3, 4, 5 };

System.out.println(a.length);    // 获取 a 指向的数组对象的长度
```

9. 数组类型引用的 [] 操作符理解

只有数组引用才支持 [] 操作符!

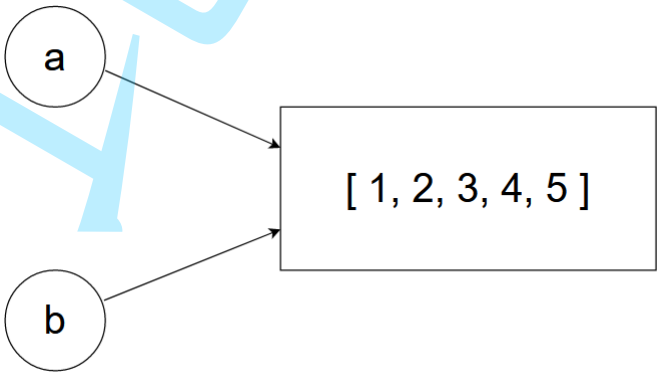
```
int[] a = { 1, 2, 3, 4, 5 };

System.out.println(a[0]);        // 获取 a 指向的数组对象的第一个元素
```

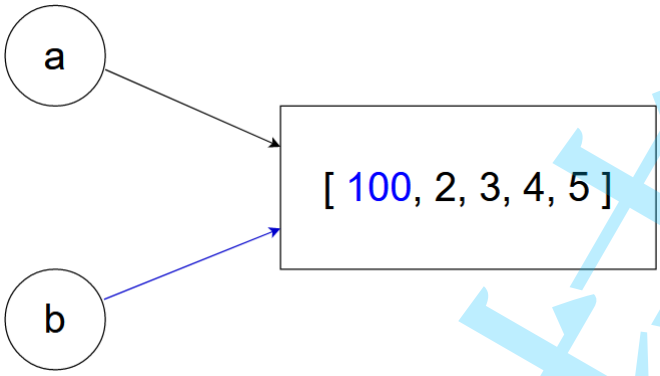
10. 通过引用操作对象表现出的"共享"特性

10.1 通过赋值表现

```
int[] a = { 1, 2, 3, 4, 5 };
System.out.println(Arrays.toString(a)); // [1, 2, 3, 4, 5]
int[] b = a;
System.out.println(Arrays.toString(b)); // [1, 2, 3, 4, 5]
```



```
b[0] = 100;
System.out.println(Arrays.toString(b)); // [100, 2, 3, 4, 5]
System.out.println(Arrays.toString(a)); // [100, 2, 3, 4, 5];
```

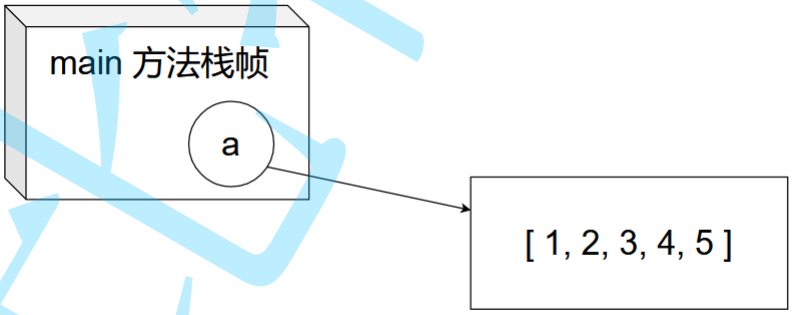


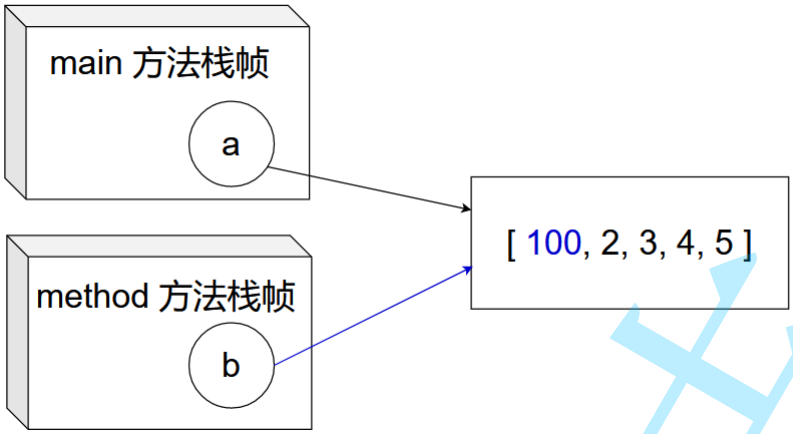
由于通过引用修改的对象中的数据，而 a 和 b 指向同一个对象。所以，通过 b 修改了数组对象中的数据之后，通过 a 可以看到修改后的变化。

10. 2 通过方法传参表现

```
public static void method(int[] b) {
    System.out.println(Arrays.toString(b)); // [1, 2, 3, 4, 5]
    b[0] = 100;
    System.out.println(Arrays.toString(b)); // [100, 2, 3, 4, 5]
}

public static void main(String[] args) {
    int[] a = { 1, 2, 3, 4, 5 };
    System.out.println(Arrays.toString(a)); // [1, 2, 3, 4, 5]
    method(a);
    System.out.println(Arrays.toString(a)); // [100, 2, 3, 4, 5]
}
```





通过方法传递参数，其实也是让 b 指向 a 目前指向的对象，然后做修改。所以同样表现出“共享”特性

11. `NullPointerException` 异常

常被翻译为**空指针异常**，但大家要注意，这里的指针和我们 C 语言解答学过的指针毫无关系。

已知，在对一个引用做 `.` 或者 `[]` 操作的时候，就是需要通过引用操作引用指向的对象。

如果有 一个 引用 == null，则表示该引用没有指向任何对象。

则此时在对该引用使用 `.` 或者 `[]` 操作符，就会抛出 `NullPointerException` 了。

接下来，我们通过代码，看 java 出现 `NullPointerException` 的常见形式。

通过引用.某某某，进行解引用操作

```
int[] arr = null;

System.out.println(arr.length);

// 运行结果，会出现运行时异常
Exception in thread "main" java.lang.NullPointerException
```

这里就是要对 arr 这个引用做解引用动作时，由于 arr 是空，所以出现了异常。

通过引用[下标]，进行解引用操作

```
int[] arr = null;
System.out.println(arr[0]);

// 执行结果
Exception in thread "main" java.lang.NullPointerException
```

这里就是要对 arr 这个引用做解引用动作时，由于 arr 是空，所以出现了异常。

总结

1. 引用是一种数据类型，用来指向对象。
2. 对引用进行的大部分操作实际上都是操作的该引用指向的对象。
3. 当多个引用指向同一个对象时，通过哪个引用修改了对象，其他引用都可以看到变化
4. 当一个引用不指向对象时，要求访问其指向的对象，就会遇到 `NullPointerException`

最后，关于引用和对象的理解是整个 java 学习过程中的重点和难点，我们会在以后的课程中多次反复地从不同的角度带着大家理解这两个概念，切勿盲人摸象。