这个结构体指明了消息应该被传递给特定超类的定义，虽然如此，receiver接收者仍然是self本身。

因为当我们想通过[super class]获取超类时，super只是编译器的标识符，只会将指向self的id指针和class的SEL传递给objc_msgSendSuper函数，因为只有在NSObject类中找到class方法，然后class方法调用object_getClass(),接着调用objc_msgSend(objc_super->receiver,@selector (class))，传入的第一个参数是指向self的id指针，与调用[self class]相同，那么得到的还是self的类型。

2、动态方法解析

通常而言，对象在接收到消息后，会通过IMP去找到并执行响应方法，而动态方法解析则是会在消息启动转发机制之前执行。具体表现在当对象接收到未知的消息时，先会调用所属的类方法或者实例方法。在这个方法中我们可以为未知的消息添加处理方法，当然了，我们所添加的处理方法肯定是我们已经实现的，至于如何添加，我们会在后边讲解。

```
+ (BOOL)resolveClassMethod:(SEL)sel OBJC_AVAILABLE(10.5, 2.0, 9.0, 1.0);
+ (BOOL)resolveInstanceMethod:(SEL)sel OBJC_AVAILABLE(10.5, 2.0, 9.0, 1.0);
```

```
+ (BOOL)resolveInstanceMethod:(SEL)sel
{
    NSString *method1 = NSStringFromSelector(sel);
    if ([method1 isEqualToString:@"getC"]) {
        class_addMethod([self class], @selector(printText),(IMP)textMethod , "v@:");
    }
    return YES;
}
```

如果有的方法在解析这一步不想进行处理，系统会自动将方法选择器传送到消息转发机制，此时需要上述方法返回为NO