07/20/2017 ARC, selector

1. ARC
2. @Selector:
   send a message to an object that does not implement the method.

   - sending an unrecognized message produces a runtime error, causing an application to crash. But before the crash happens, iOS's runtime system gives each object a second chance to handle a message.
   - `SEL a=@selector(methodName)/ NSSelectorFromString(@"methodNameaaa")`
     `[self performSelector:aSelector];`
   - it will crash no matter "`SEL a=@selector(methodName)"(compile time)/"NSSelectorFromString(@"methodNameaaa")" (run time)`
   - use @Selector https://stackoverflow.com/questions/3482344/what-actually-is-a-selector
     - `@selector()` is a compiler directive to turn whatever's inside the parenthesis into a `SEL`. A `SEL` is a type to indicate a method name, but *not* the method implementation. (For that you'd need a different type, probably an `IMP` or a `Method`) Under-the-hood, a `SEL` is implemented as a `char*`, although relying on that behavior is not a good idea. If you want to inspect what `SEL` you have, the best way to do it is to turn it into an `NSString*` like this:

       ```
       NSLog(@"the current method is: %@",
       NSStringFromSelector(_cmd));
       ```

     - (Assuming you know that `_cmd` is one of the hidden parameters of every method call, and is the `SEL` that corresponds to the current method)

     - The Objective-C Programming Language Guide has much more information on the subject.
   - objective C using string to call a method
     - **You can't use `performSelector` for a method with 3 (or more) arguments.**
     - But for your information, here's how to use it:

```
SEL m1;
SEL m2;
SEL m3;

m1 = NSSelectorFromString( @"someMethodWithoutArg" );
m2 = NSSelectorFromString( @"someMethodWithAnArg:" );
m1 = NSSelectorFromString(
@"someMethodWithAnArg:andAnotherOne:" );

[ someObject performSelector: m1 ];
[ someObject performSelector: m2 withObject: anArg ];
[ someObject performSelector: m2 withObject: anArg
withObject: anOtherArg ];
```

- For methods with more than 2 arguments, **you will have to use the [NSInvocation](#) class**.
- Take a look at the documentation to learn how to use it.

- Basically:

- NSInvocation * invocation = [NSInvocation new];
  [invocation setSelector: NSStringFromSelector(
  @"methodWithArg1:arg2:arg3:")];

  // Argument 1 is at index 2, as there is self and _cmd before
  [invocation setArgument: &arg1 atIndex: 2];
  [invocation setArgument: &arg2 atIndex: 3];
  [invocation setArgument: &arg3 atIndex: 4];

  [invocation invokeWithTarget: targetObject];

  // If you need to get the return value
  [invocation getReturnValue: &someVar];

  [invocation release];
- `Solution 2:`
- You can directly use `objc_msgsend` :
- NSString *methodName = [plistA objectForKey:@"method"];
  objc_msgSend(self, methodName, c, b, a);

◦ `Message Forwarding`
- If you send a message to an object that does not handle that message, before announcing an error the runtime sends the object a `forwardInvocation:` message with an `NSInvocation` object as its sole argument—the `NSInvocation` object encapsulates the original message and the arguments that were passed with it.
- There are several ways to avoid crash:
- 1. simply passes the message on to an instance of the other class. (cumbersome)

```
- (id)negotiate

{

    if ( [someOtherObject respondsTo:@selector(negotiate)] )

        return [someOtherObject negotiate];

    return self;

}
```

- 2. forward a message and inheritance.
  Class subA: A { … }
  Class A: NSObject {
  -(void)negotiate() { … }
  }
  if ([subA respondsToSelector:@selector(negotiate)]) // return NO, so we must re-implement the respondsToSelector: and isKindOfClass: methods to include your forwarding algorithm:

```
- (BOOL)respondsToSelector:(SEL)aSelector

{

    if ( [super respondsToSelector:aSelector] )

        return YES;

    else {

        /* Here, test whether the aSelector message can     *

         * be forwarded to another object and whether that  *

         * object can respond to it. Return YES if it can.  */

    }

    return NO;

}
```

- 3. forward a message to its surrogate.
  Using `forwardInvocation:` and `methodSignatureForSelector:`
  Note: in this class, must do not have method `methodNameB`, otherwise does not call to methodSignatureForSelector.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    _someOtherObject = [baseObject new];
//    SEL a = @selector(methodName);
    SEL aSelector = NSSelectorFromString(@"methodNameB");
    [self performSelector:aSelector];
}
```

```
- (void)forwardInvocation:(NSInvocation *)anInvocation
{
    if ([_someOtherObject respondsToSelector:
         [anInvocation selector]])
        [anInvocation invokeWithTarget:_someOtherObject];
    else
        [super forwardInvocation:anInvocation];
}
SEL// must write following method
- (NSMethodSignature*)methodSignatureForSelector:(SEL)selector
{
    NSMethodSignature* signature = [super
methodSignatureForSelector:selector];
    if (!signature) {
        signature = [_someOtherObject
methodSignatureForSelector:selector];
    }
    return signature;
}
```

- 4. if Protocol also used:
  In addition to `respondsToSelector:` and `isKindOfClass:`, the `instancesRespondToSelector:` method should also mirror the forwarding algorithm. If protocols are used, the `conformsToProtocol:` method should likewise be added to the list.
- 5. Super-easy Forwarding (iOS 4.0 and later)

```
- (id)forwardingTargetForSelector:(SEL)sel {
   if ([_someOtherObject respondsToSelector: sel]) return
_someOtherObject;
   return nil;
}
```

- Related Article:
  - objc does not provide true multiple-inheritance, but we can let object to respond to another class's messages

```
[_dataSource respondsToSelector:()] need make sure _dataSource <NSObject>.
Otherwise,

@protocol WDPWorkerProfileSummaryViewDataSource <NSObject>
- (WDPWorkerGridButtonTypes)gridButtonTypesForSummaryView:
(WDPWorkerProfileSummaryView *)summaryView;

@property (nonatomic, weak) id<WDPWorkerProfileSummaryViewDataSource>
dataSource;
```

Objective-C: dynamic type

The Handbook of Software for Engineers and Scientists

Learning IPad Programming: A Hands-on Guide to Building IPad Apps with IOS 5