Custom Formatters

自定义格式化程序

When formatting data into a user-readable format we tend to use quick one-off solutions. This is a shame because Foundation comes with NSFormatter, which is perfectly suited for this task and can be easily reused throughout your code base. Heck, if you’re on a Mac, AppKit classes have built-in support for NSFormatter, making your life a lot easier.

当把一些数据格式化成我们易懂的格式时，我们希望能有一种简单而快速的方案。Foundation框架中的NSFormatter能很好的胜任这个工作。如果你用的Mac，AppKit类内建了对NSFormatter的支持，

Built-in Formatters

Foundation comes with the abstract NSFormatter class and two concrete subclasses: NSNumberFormatter and NSDateFormatter. We’re going to skip these and jump right into the deep end, implementing our own subclass.

If you need a more subtle introduction, I recommend reading this [NSHipster post](http://nshipster.com/nsformatter/).

Foundation框架中的NSFormatter是一个抽象类，它有两个子类NSNumberFormatter和NSDateFormatter。这里我们将略过上述内容，实现我们自己的子类。如果你还想了解更多，请移步这里。

Introduction

NSFormatter by itself doesn’t do anything, except throw errors. I have yet to find a programmer who wants this, but if such a thing tickles your fancy, go for it!

NSFormatter除了抛出错误，其它什么也不干。不知道有没有程序员想要这样。

Because we don’t like errors, we’ll implement an NSFormatter subclass that can transform instances of UIColor to a human-readable name. For example, the following code will return the string “Blue”:

大家都不喜欢错误，所以我们实现一个子类，它能够将UIColor的实例变得更可读。例如下面的代码会返回一个“blue”字符串：

KPAColorFormatter \*colorFormatter = [[KPAColorFormatter alloc] init]; [colorFormatter stringForObjectValue:[UIColor blueColor]] // Blue

Two methods are required when implementing a NSFormatter subclass:stringForObjectValue: and getObjectValue:forString:errorDescription:. We’ll start of with the first because that’s the one you’ll use most often. The second is, as far as I know, most often used in OS X and actually not very useful. More on that later.

子类化NSFormatter需要实现stringForObjectValue：和getObjectValue:forString:errorDescription：两个方法。第一个方法最常用，第二个方法更多是在OSX上使用。

The Initializer

Hold your horses, as first we need to do some setup. There is no pre-defined mapping from colors to their names, so we need to define this. For the sake of simplicity, this will be our initializer:

首先，我们需要做一些设置，先预定义颜色对象和颜色的对应关系。下面是一个简单的实现：

- (id)init; { return [self initWithColors:@{ [UIColor redColor]: @"Red", [UIColor blueColor]: @"Blue", [UIColor greenColor]: @"Green" }]; }

Our ‘known’ colors are a dictionary keyed by a UIColor with the English name as values. I’ll leave the implementation of the initWithColors: method to your imagination. Or, if you’re that person who looks at the answers on the last page of the puzzle book, go ahead and take a look at the [Github repo](https://github.com/klaaspieter/KPAColorFormatter).

Formatting Object Values

The first thing we need to do in stringForObjectValue: is verify that the value is of the expected class. We can only format UIColors so this is the start of our method:

我们的方法只能格式化UIColor对象，所以做的第一件事就是判断传入的参数。

- (NSString \*)stringForObjectValue:(id)value; { if (![value isKindOfClass:[UIColor class]]) { return nil; } // To be continued... }

After we’ve verified that the value is what we expect it to be, we can do the real magic. Recall that our formatter has a dictionary of color names keyed by their color. To make it work, all we need to do is look up the name using the color value as key:

在判断参数合法之后，我们的真正逻辑就要开始了。在我们的类中，有一个包含颜色名字的字典，我们只需要用颜色的名字作为Key来查找。

- (NSString \*)stringForObjectValue:(id)value; { // Previously on KPAColorFormatter return [self.colors objectForKey:value]; }

This is the simplest implementation possible. A more advanced (and useful) formatter would also be able to look up color names that don’t exist in our dictionary by finding the closest known color. I’ll leave that as an exercise to the reader. Or, if you don’t work out much, take a look at the [Github repo](https://github.com/klaaspieter/KPAColorFormatter).

上面的代码是一个简单的实现。一个更成熟的格式化器是能够找到在我们字典中不存在的颜色，返回一个相近的颜色。

Reverse Formatting

Any formatter should also support reverse formatting from a string back to an instance of the class. This is done usinggetObjectValue:forString:errorDescription:. The reason for this is that on OS X, formatters are often used in combination with NSCells.

任何的格式化器都应该支持任何对象的反转格式化 。通过getObjectValue:forString:errorDescription:方法可以实现。在OSX上，在使用NSCell时会经常用到这个方法。

NSCell’s have a objectValue property. By default, NSCell will use the objectValue’s description, but it can optionally use a formatter. In the case of NSTextFieldCell, a user can also enter a string value and you, as the programmer, now can expectobjectValue to be an instance of UIColor that represents that string value. In our case, the user could enter “Blue” and we should return, by reference, a [UIColor blueColor] instance.

NSCell有一个objectValue的属性。默认情况下，NSCell用objectValue的描述。在遇到NSTextFieldCell时，用户输入一个字符串，作为程序员的我们可能期望objectValue能变成一个UIColor的实例对象。例如，用户输入“Blue”，我们应该返回一个[UIColor blueColor] 的实例变量。

There are two parts to implementing reverse formatting: the part where the formatter can successfully transform the string value into an instance of UIColor, and one where it cannot. Let’s start with the happy path:

实现反转格式化分为两部分：一部分是从一个字符串可以格式化成UIColor实例对象，代码如下：

- (BOOL)getObjectValue:(out \_\_autoreleasing id \*)obj forString:(NSString \*)string errorDescription:(out NSString \*\_\_autoreleasing \*)error; { \_\_block UIColor \*matchingColor = nil; [self.colors enumerateKeysAndObjectsUsingBlock:^(UIColor \*color, NSString \*name, BOOL \*stop) { if([name isEqualToString:string]) { matchingColor = color; \*stop = YES; } }]; if (matchingColor) { \*obj = matchingColor; return YES; } // Snip

There is some optimization that can be done here, but let’s not do that prematurely. This enumerates through every object in our colors dictionary and when a name is found it will return the color instance associated with it by reference. We also returnYES to notify the caller that we were able to turn the string back into an object.

上面的代码还可以优化，但现在还不是时候。我们遍历包含颜色的字典，当找到一个我们需要的颜色的名字时，会返回一个颜色对象的引用，同时也会返回一个YES，告知调用者我们已经成功把字符串转换成了颜色对象。

Now the error path:

if (matchingColor) { // snap } else if (error) { \*error = [NSString stringWithFormat:@"No known color for name: %@", string]; } return NO;

If we can’t find a matching color we check if the caller is interested in errors, and if so, return it by reference. The check for error here is important. If you don’t do this you *will* crash. We also return NO to notify the caller that conversion was not successful.

如果没有找匹配的颜色，我们会检测调用者是否需要错误，如果需要，就返回。错误检测很重要，如果你不做检测，程序很可能崩溃。我们也会返回NO告诉调用者，转换失败。

Localization

That’s it! We have a fully functional NSFormatter subclass, for English speakers, living in the United States.

我们已经有一个NSFormatter 子类，可以帮助生活在美国讲英文的人。

That’s about [319 million people](http://www.wolframalpha.com/input/?i=population+of+the+united+states) compared to [7.13 billion](http://www.wolframalpha.com/input/?i=population+of+the+world) in the entire world. In other words [96 percent](http://www.wolframalpha.com/input/?i=ratio+of+population+of+the+United+States+vs+population+of+the+world) of your potential users aren’t impressed, yet. Of course I hear you say: most of those don’t own iPhones or Macs and it’s really a much smaller number. Where’s the fun in that? Party pooper!

相比全世界71.3亿人口，那才3.19亿人口。换句话说，你还有大约96%的潜在用户。当然了，你可能认为他们中的大多数都没有iPhone或者Mac.

If you take a look at NSNumberFormatter and NSDateFormatter you’ll see that both have a locale property taking an instance of NSLocale. Let’s extend our formatter to support this and return a translated name based on the locale.

NSNumberFormatter 和NSDateFormatter 都有一个locale的属性，它是一个NSLocale类的实例对象。接下来让我们扩展我们的格式化器，让它可以根据locale返回对应翻译的名字。

Translating

The first thing we need to do is translate the color strings. Messing with genstring, and \*.lprojs is outside of the scope of this article. I mean, you have to get back to work at some point right? If not, [there](https://developer.apple.com/internationalization/) [are](http://nshipster.com/nslocalizedstring/) [a](https://developer.apple.com/library/ios/documentation/MacOSX/Conceptual/BPInternational/BPInternational.html#//apple_ref/doc/uid/10000171i) [bunch](http://www.tethras.com/apple) [of](http://www.ibabbleon.com/iphone_app_localization.html) [articles](http://www.getlocalization.com/library/get-localization-mac/) [about](http://www.daveoncode.com/2010/05/15/iphone-applications-localization/) [it](http://useyourloaf.com/blog/2010/12/15/localize-iphone-application-name.html). Read them all? Awesome! No more work, time to go home.

首先，我们需要做的是翻译颜色字符串。如何产生\*.lprojs 和genstring超出了本文的范围。可以移步到这里阅读相关文章。

Localized Formatting

Back to implementing localization. After we have access to the translated strings, we need to update stringForObjectValue: so it knows where to get the translation. Those who have worked with NSLocalizedString before probably already went ahead and replaced every string with NSLocalizedString. Wrong!

接下来是本地化功能的实现。在获取翻译字符串后，我们需要更新stringForObjectValue来实现翻译。

We are dealing with a dynamic locale here and NSLocalizedString will only find the translation for the current default language. Now in 99 percent of the cases, this is probably what you want, hence the default behavior, but we need to look up the language dynamically using the locale set on our formatter.

The new implementation of stringForObjectValue:

下面是stringForObjectValue的最新实现

- (NSString \*)stringForObjectValue:(id)value; { // Previously on... don't you hate these? I just watched that 20 seconds ago! NSString \*languageCode = [self.locale objectForKey:NSLocaleLanguageCode]; NSURL \*bundleURL = [[NSBundle bundleForClass:self.class] URLForResource:languageCode withExtension:@"lproj"]; NSBundle \*languageBundle = [NSBundle bundleWithURL:bundleURL]; return [languageBundle localizedStringForKey:name value:name table:nil]; }

This leaves room for some refactoring, but bear with me. It’s easier to read if all the code is in the same place.

上面的代码还可以进一步优化。所有代码在一起会更容易读

We first find the language code for the locale, after which we look up the NSBundlefor that language. We then ask the bundle to give us the translation of the English name. If it can’t be found, the name: argument (our English name) will be returned as a fallback value. In case you’re interested, this is exactly what NSLocalizedStringdoes except that we’re dynamically looking up the bundle.

我们通过NSBundle找到对应的语言后，转换为locale.然后我们会让bundle把英文名字进行翻译。如果在locale中没找到，则会返回英文的名字。

Localized Reverse Formatting

That leaves us with reverse formatting from a translated color name back to color instances. Honestly, I don’t think it’s worth it. Our current implementation works perfect for probably 99 percent of the cases. In the other one percent where you are using this formatter on a Mac, on an NSCell, and you’re allowing your users to enter color names which you’re going to try and parse, you’ll need something a lot more complicated than a simple NSFormatter subclass. Also, you probably shouldn’t be letting your users enter colors using text. [NSColorPanel](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/ApplicationKit/Classes/NSColorPanel_Class/) is a much better solution.

同样，我们也可以反转格式化，将一个被翻译的颜色名字变化成颜色对象。当然我认为这是不值得的。我们现在的版本适用于99%的情况。另外1%的情况是在Mac上，当在NSCell类上使用该格式化器，用户正输入一个你试图解析的颜色名字时，你可能需要一个比NSFormatter 子类更复杂的对象来处理。或许你不应该允许用户用文本输入颜色。[NSColorPanel](https://developer.apple.com/library/mac/documentation/Cocoa/Reference/ApplicationKit/Classes/NSColorPanel_Class/)是一个更好的方案。

Attributed Strings

Now that our formatter does what we want it to do, let’s implement something completely useless. You know, because we can. cxzq

Formatters also have support for attributed strings. I find that whether or not you want to use that support really depends on your specific application and its user interface. Therefore, you will probably want to make that part of your formatter configurable.

For ours, I want to set the text color to the color that we’re formatting. This is what that looks like:

- (NSAttributedString \*)attributedStringForObjectValue:(id)value withDefaultAttributes:(NSDictionary \*)defaultAttributes; { NSString \*string = [self stringForObjectValue:value]; if (!string) { return nil; } NSMutableDictionary \*attributes = [NSMutableDictionary dictionaryWithDictionary:defaultAttributes]; attributes[NSForegroundColorAttributeName] = value; return [[NSAttributedString alloc] initWithString:string attributes:attributes]; }

First we format the string like normal, after which we check that formatting was successful. Then we merge the default attributes with our foreground color attribute. Finally, we return the attributed string. Simple, right?

Convenience

Because initializing built-in formatters [is slow](https://twitter.com/ID_AA_Carmack/status/28939697453), it has become common practice to also expose a convenience class method on your formatter. The formatter should use same defaults and the current locale. This is the implementation for our formatter:

因为初始化内建的格式化器是很慢的，

+ (NSString \*)localizedStringFromColor:(UIColor \*)color; { static dispatch\_once\_t onceToken; dispatch\_once(&onceToken, ^{ KPAColorFormatterReusableInstance = [[KPAColorFormatter alloc] init]; }); return [KPAColorFormatterReusableInstance stringForObjectValue:color]; }

Unless your formatter is doing really crazy formatting, like NSNumberFormatter andNSDateFormatter, you probably don’t need this for performance reasons. But it’s still good to do because it makes using your formatter easier.

除非你的格式化器像NSNumberFormatter 和NSDateFormatter那样变态，否则你是不需要考虑性能问题的。

Wrapping Up

Our color formatter can now format a UIColor instance into a human-readable name and the other way around. There is a lot more that NSFormatter can do that we haven’t covered yet. Especially on the Mac where, because of its integration withNSCell, you can use more advanced stuff, like specifying and validating a string representation while the user is editing.

现在我们的颜色转换器能够将颜色转换成人们易懂的名字。NSFormatter 还可以干很多事。在Mac上，因为集成了NSCell, 有更多高级功能。例如当用户编辑时，验证字符串。

There is also more we can do to make our formatter more customizable. For example, my implementation will attempt to look up the closest color name if there is no direct match. Sometimes you may not want this so our formatter should have a Boolean property to control this behavior. Similarly, our attributed string formatting might not be what you want either and should also be more customizable.

我们的格式化器可以做更多的自定义.例如，没有查找到一个你需要的颜色名字时，我们会返回给你最相关的颜色名字。我们可以暴露一个布尔值来开关该功能。或许我们的格式化器可能不是你想要的，你也可以自己定义一个。

That said, we ended up with a pretty solid formatter. All the code (with an OS X example) is available on [Github](https://github.com/klaaspieter/KPAColorFormatter). My implementation is also available on [CocoaPods](http://cocoapods.org/). If you desperately need this functionality in your app, add pod "KPAColorFormatter"to your Podfile and have fun experimenting!

所有的代码在都放在了Github上。我的实现也放在了CocoaPods上。如果你的App需要此功能，可以将KPAColorFormatter放到你的Podfile文件中。