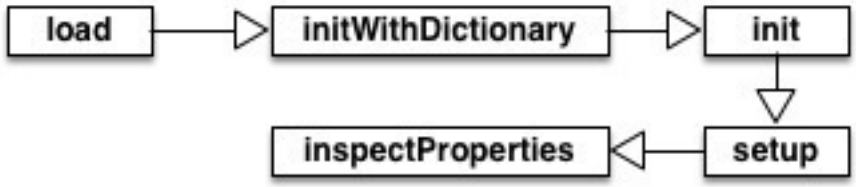
JSONModel

主要是基于Objective-C Runtime的反射机制

JSONModel中的实现

打断点记录了下JSONModel这个类中的方法调用顺序如下：



对象属性的获取则主要在最后一个inspectProperties方法
以下是inspectProperties方法的代码片段：

```
-(void)__setup__
{
    //if first instance of this model, generate the property list
    if (!objc_getAssociatedObject(self.class,
&kClassPropertiesKey)) {
        [self __inspectProperties];
    }

    //if there's a custom key mapper, store it in the associated
object
    id mapper = [[self class] keyMapper];
    if ( mapper && !objc_getAssociatedObject(self.class,
&kMapperObjectKey) ) {
        objc_setAssociatedObject(
                                 self.class,
                                 &kMapperObjectKey,
                                 mapper,
                                 OBJC_ASSOCIATION_RETAIN // This
is atomic
                                 );
    }
}

//inspects the class, get's a list of the class properties
```

```
-(void)__inspectProperties
{
    //JMLog(@"Inspect class: %@", [self class]);

    NSMutableDictionary* propertyIndex = [NSMutableDictionary
dictionary];

    //temp variables for the loops
    Class class = [self class];
    NSScanner* scanner = nil;
    NSString* propertyType = nil;

    // inspect inherited properties up to the JSONModel class
    while (class != [JSONModel class]) {
        //JMLog(@"inspecting: %@", NSStringFromClass(class));

        unsigned int propertyCount;
        objc_property_t *properties =
class_copyPropertyList(class, &propertyCount);

        //loop over the class properties
        for (unsigned int i = 0; i < propertyCount; i++) {

            JSONModelClassProperty* p = [[JSONModelClassProperty
alloc] init];

            //get property name
            objc_property_t property = properties[i];
            const char *propertyName =
property_getName(property);
            p.name = @(propertyName);

            //JMLog(@"property: %@", p.name);

            //get property attributes
            const char *attrs = property_getAttributes(property);
            NSString* propertyAttributes = @(attrs);
            NSArray* attributeItems = [propertyAttributes
componentsSeparatedByString:@","];

            //ignore read-only properties
            if ([attributeItems containsObject:@"R"]) {
                continue; //to next property
            }
```

```
            //check for 64b BOOLs
            if ([propertyAttributes hasPrefix:@"Tc,"]) {
                //mask BOOLs as structs so they can have custom
convertors
                p.structName = @"BOOL";
            }

            scanner = [NSScanner scannerWithString:
propertyAttributes];

            //JMLog(@"attr: %@", [NSString
stringWithCString:attrs encoding:NSUTF8StringEncoding]);
            [scanner scanUpToString:@"T" intoString: nil];
            [scanner scanString:@"T" intoString:nil];

            //check if the property is an instance of a class
            if ([scanner scanString:@"@\"" intoString:
&propertyType]) {

                [scanner
scanUpToCharactersFromSet:[NSCharacterSet
characterSetWithCharactersInString:@"\"<"]
                                        intoString:
&propertyType];

                //JMLog(@"type: %@", propertyClassName);
                p.type = NSClassFromString(propertyType);
                p.isMutable = ([propertyType rangeOfString:
@"Mutable"].location != NSNotFound);
                p.isStandardJSONType = [allowedJSONTypes
containsObject:p.type];

                //read through the property protocols
                while ([scanner scanString:@"<" intoString:NULL])
{

                    NSString* protocolName = nil;

                    [scanner scanUpToString:@">" intoString:
&protocolName];

                    if ([protocolName isEqualToString:
@"Optional"]) {
                        p.isOptional = YES;
                    } else if([protocolName isEqualToString:
```

```
@"Index"]) {
                        p.isIndex = YES;
                        objc_setAssociatedObject(
                                                 self.class,
                                                 &kIndexPropertyN
ameKey,
                                                 p.name,
                                                 OBJC_ASSOCIATION
_RETAIN // This is atomic
                                                 );
                    } else if([protocolName isEqualToString:
@"ConvertOnDemand"]) {
                        p.convertsOnDemand = YES;
                    } else if([protocolName isEqualToString:
@"Ignore"]) {
                        p = nil;
                    } else {
                        p.protocol = protocolName;
                    }

                    [scanner scanString:@">" intoString:NULL];
                }

            }
            //check if the property is a structure
            else if ([scanner scanString:@"{" intoString:
&propertyType]) {
                [scanner scanCharactersFromSet:[NSCharacterSet
alphanumericCharacterSet]
                                    intoString:&propertyType];

                p.isStandardJSONType = NO;
                p.structName = propertyType;

            }
            //the property must be a primitive
            else {

                //the property contains a primitive data type
                [scanner
scanUpToCharactersFromSet:[NSCharacterSet
characterSetWithCharactersInString:@","]
                                        intoString:
&propertyType];
```

```
                //get the full name of the primitive type
                propertyType = valueTransformer.primitivesNames[propertyType];

                if (![allowedPrimitiveTypes containsObject:propertyType]) {

                    //type not allowed - programmer mistaked -> exception
                    @throw [NSException exceptionWithName:@"JSONModelProperty type not allowed"
                                                   reason:[NSString stringWithFormat:@"Property type of %@.%@ is not supported by JSONModel.", self.class, p.name]
                                                 userInfo:nil];
                }

            }

            NSString *nsPropertyName = @(propertyName);
            if([[self class] propertyIsOptional:nsPropertyName]){
                p.isOptional = YES;
            }

            if([[self class] propertyIsIgnored:nsPropertyName]){
                p = nil;
            }

            //few cases where JSONModel will ignore properties automatically
            if ([propertyType isEqualToString:@"Block"]) {
                p = nil;
            }

            //add the property object to the temp index
            if (p && ![propertyIndex objectForKey:p.name]) {
                [propertyIndex setValue:p forKey:p.name];
            }
        }

        free(properties);

        //ascend to the super of the class
        //(will do that until it reaches the root class - JSONModel)
```

```
        class = [class superclass];
    }

    //finally store the property index in the static property
index
    objc_setAssociatedObject(
                             self.class,
                             &kClassPropertiesKey,
                             [propertyIndex copy],
                             OBJC_ASSOCIATION_RETAIN // This is
atomic
                             );
}
```

在这边可以看到基本步骤如下

1. 通过调用自身的class方法获取当前类的元数据信息
2. 通过runtime的 class_copyPropertyList 方法取得当前类的属性列表，以指针数组的形式返回
3. 遍历指针数组，通过property_getName获取属性名，property_getAttributes获取属性类型
4. 使用NSScanner来扫描属性类型字符串，将类似如下的形式"T@"NSNumber",&,N,V_id"，处理成NSNumber，逐个属性循环处理
5. 将所有处理好的数据放入propertyIndex这个字典中
6. 通过objc_setAssociatedObject将这些数据关联到kClassPropertiesKey

使用时在properties方法中这样取出属性数据：

```
//returns a list of the model's properties
-(NSArray*)__properties__
{
    //fetch the associated object
    NSDictionary* classProperties =
objc_getAssociatedObject(self.class, &kClassPropertiesKey);
    if (classProperties) return [classProperties allValues];

    //if here, the class needs to inspect itself
    [self __setup__];

    //return the property list
    classProperties = objc_getAssociatedObject(self.class,
&kClassPropertiesKey);
    return [classProperties allValues];
}
```