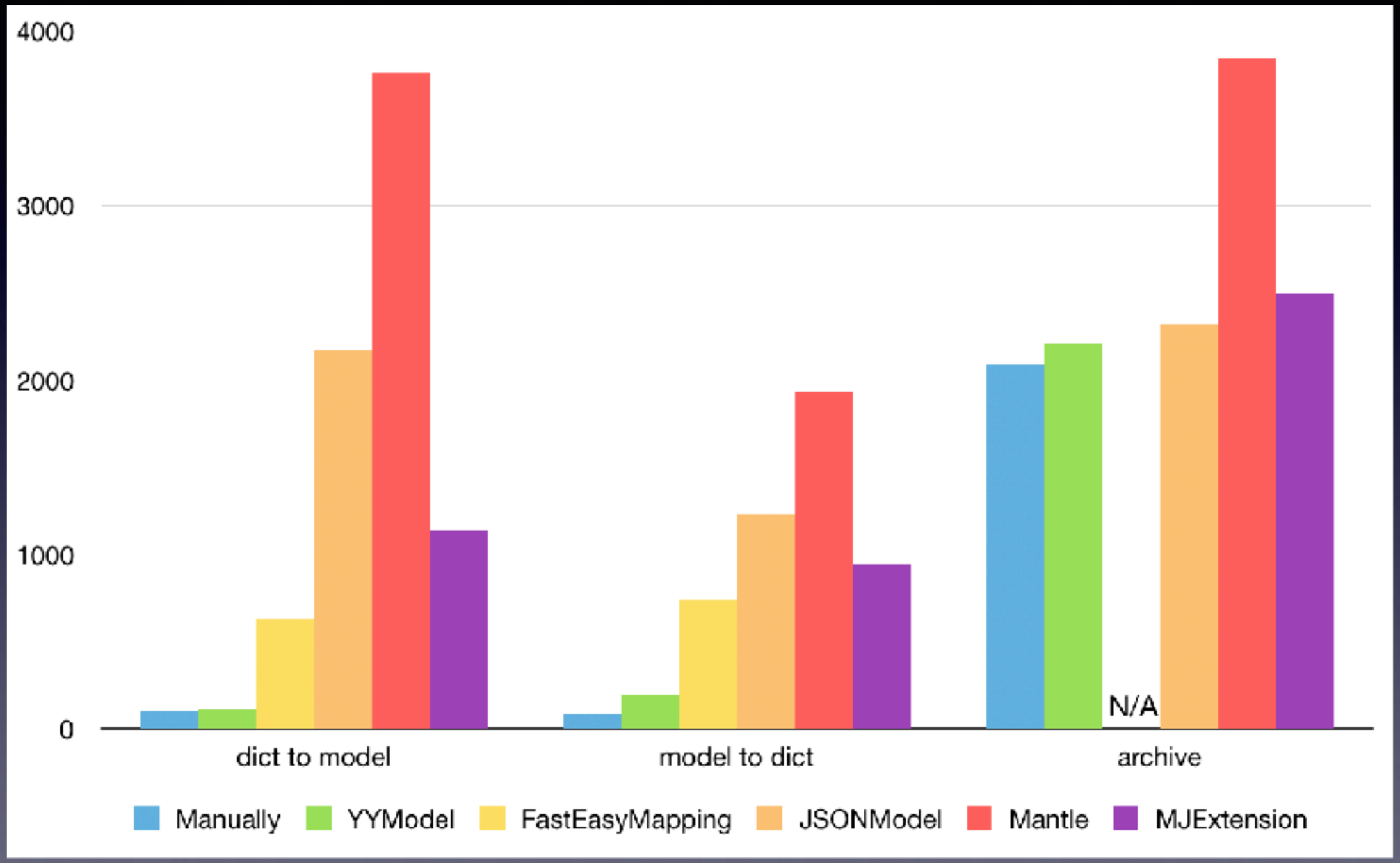
# YYModel分享

nate

处理 GithubUser 数据 10000 次耗时统计 (iPhone 6):



# 特性

- **高性能**: 模型转换性能接近手写解析代码。
- **自动类型转换**: 对象类型可以自动转换，详情见下方表格。
- **类型安全**: 转换过程中，所有的数据类型都会被检测一遍，以保证类型安全，避免崩溃问题。
- **无侵入性**: 模型无需继承自其他基类。
- **轻量**: 该框架只有 5 个文件 (包括.h文件)。
- **文档和单元测试**: 文档覆盖率100%, 代码覆盖率99.6%。

```
// JSON:
{
    "uid":123456,
    "name":"Harry",
    "created":"1965-07-31T00:00:00+0000"
}

// Model:
@interface User : NSObject
@property UInt64 uid;
@property NSString *name;
@property NSDate *created;
@end
@implementation User
@end


// 将 JSON (NSData,NSString,NSDictionary) 转换为 Model:
User *user = [User yy_modelWithJSON:json];

// 将 Model 转换为 JSON 对象:
NSDictionary *json = [user yy_modelToJSONObject];
```

# 描述一个类

```
//定义一个变量

/**
 Instance variable information.
 */
@interface YYClassIvarInfo : NSObject
@property (nonatomic, assign, readonly) Ivar ivar;                  ///< ivar opaque struct
@property (nonatomic, strong, readonly) NSString *name;             ///< Ivar's name
@property (nonatomic, assign, readonly) ptrdiff_t offset;           ///< Ivar's offset
@property (nonatomic, strong, readonly) NSString *typeEncoding; ///< Ivar's type encoding
@property (nonatomic, assign, readonly) YYEncodingType type;     ///< Ivar's type
```

```
//定义一个方法

/**
 Method information.
 */
@interface YYClassMethodInfo : NSObject
@property (nonatomic, assign, readonly) Method method;                  ///< method opaque struct
@property (nonatomic, strong, readonly) NSString *name;                 ///< method name
@property (nonatomic, assign, readonly) SEL sel;                        ///< method's selector
@property (nonatomic, assign, readonly) IMP imp;                        ///< method's implementation
@property (nonatomic, strong, readonly) NSString *typeEncoding;         ///< method's parameter and return types
@property (nonatomic, strong, readonly) NSString *returnTypeEncoding;   ///< return value's type
@property (nullable, nonatomic, strong, readonly) NSArray<NSString *> *argumentTypeEncodings; ///< array of arguments'
    type
```

```objc
//定义一个属性

/**
 Property information.
 */
@interface YYClassPropertyInfo : NSObject
@property (nonatomic, assign, readonly) objc_property_t property; ///< property's opaque struct
@property (nonatomic, strong, readonly) NSString *name;           ///< property's name
@property (nonatomic, assign, readonly) YYEncodingType type;      ///< property's type
@property (nonatomic, strong, readonly) NSString *typeEncoding;   ///< property's encoding value
@property (nonatomic, strong, readonly) NSString *ivarName;       ///< property's ivar name
@property (nullable, nonatomic, assign, readonly) Class cls;      ///< may be nil
@property (nullable, nonatomic, strong, readonly) NSArray<NSString *> *protocols; ///< may nil
@property (nonatomic, assign, readonly) SEL getter;               ///< getter (nonnull)
@property (nonatomic, assign, readonly) SEL setter;               ///< setter (nonnull)

/**
```

```objc
//定义一个类

/**
 Class information for a class.
 */
@interface YYClassInfo : NSObject
@property (nonatomic, assign, readonly) Class cls; ///< class object
@property (nullable, nonatomic, assign, readonly) Class superCls; ///< super class object
//元类
@property (nullable, nonatomic, assign, readonly) Class metaCls;  ///< class's meta class object
@property (nonatomic, readonly) BOOL isMeta; ///< whether this class is meta class
@property (nonatomic, strong, readonly) NSString *name; ///< class name
@property (nullable, nonatomic, strong, readonly) YYClassInfo *superClassInfo; ///< super class's class info
@property (nullable, nonatomic, strong, readonly) NSDictionary<NSString *, YYClassIvarInfo *> *ivarInfos; ///< ivars
@property (nullable, nonatomic, strong, readonly) NSDictionary<NSString *, YYClassMethodInfo *> *methodInfos; ///< methods
@property (nullable, nonatomic, strong, readonly) NSDictionary<NSString *, YYClassPropertyInfo *> *propertyInfos; ///<
    properties
```

```objc
//+ (NSDictionary *)modelCustomPropertyMapper {
//    return @{ @"name" : @"n",
//              @"count" : @"ext.c",
//              @"desc1" : @"ext.d", // mapped to same key path
//              @"desc2" : @"ext.d", // mapped to same key path
//              @"desc3" : @"ext.d.e",
//              @"desc4" : @".ext",
//              @"modelID" : @[@"ID", @"Id", @"id", @"ext.id"]};
//}

//一个property 的描述
/// A property info in object model.
@interface _YYModelPropertyMeta : NSObject {
    @package
    NSString *_name;             ///< property's name
    YYEncodingType _type;        ///< property's type
    YYEncodingNSType _nsType;    ///< property's Foundation type
    BOOL _isCNumber;             ///< is c number type
    Class _cls;                  ///< property's class, or nil
    Class _genericCls;           ///< container's generic class, or nil if threr's no generic class
    SEL _getter;                 ///< getter, or nil if the instances cannot respond
    SEL _setter;                 ///< setter, or nil if the instances cannot respond
    BOOL _isKVCCompatible;       ///< YES if it can access with key-value coding
    BOOL _isStructAvailableForKeyedArchiver; ///< YES if the struct can encoded with keyed archiver/unarchiver
    BOOL _hasCustomClassFromDictionary; ///< class/generic class implements +modelCustomClassForDictionary:

    /*
     property->key:       _mappedToKey:key     _mappedToKeyPath:nil            _mappedToKeyArray:nil
     property->keyPath:   _mappedToKey:keyPath _mappedToKeyPath:keyPath(array) _mappedToKeyArray:nil
     property->keys:      _mappedToKey:keys[0] _mappedToKeyPath:nil/keyPath    _mappedToKeyArray:keys(array)
     */
    NSString *_mappedToKey;      ///< the key mapped to
    NSArray *_mappedToKeyPath;   ///< the key path mapped to (nil if the name is not key path)
    NSArray *_mappedToKeyArray;  ///< the key(NSString) or keyPath(NSArray) array (nil if not mapped to multiple keys)
    YYClassPropertyInfo *_info;  ///< property's info
    _YYModelPropertyMeta *_next; ///< next meta if there are multiple properties mapped to the same key.
}
@end
```

```objc
//_mapper是包括了所有class和superclass的property的key value缓存,
//其中的key是所有已经映射过key name和不需要映射的property name组成。
//_allPropertyMetas是所有class和superclass的property解析_YYModelPropertyMeta列表。
//_keyPathPropertyMetas是表示映射如果是一个路径比如@"count" : @"ext.c"下的_YYModelPropertyMeta列表。
//_multiKeysPropertyMetas是表示映射如果是一个NSArray
//比如@"modelID" : @[@"ID", @"Id", @"id", @"ext.id"]下的_YYModelPropertyMeta列表。
//一个类的描述

/// A class info in object model.
@interface _YYModelMeta : NSObject {
    @package
    YYClassInfo *_classInfo;
    /// Key:mapped key and key path, Value:_YYModelPropertyMeta.
    NSDictionary *_mapper;
    /// Array<_YYModelPropertyMeta>, all property meta of this model.
    NSArray *_allPropertyMetas;
    /// Array<_YYModelPropertyMeta>, property meta which is mapped to a key path.
    NSArray *_keyPathPropertyMetas;
    /// Array<_YYModelPropertyMeta>, property meta which is mapped to multi keys.
    NSArray *_multiKeysPropertyMetas;
    /// The number of mapped key (and key path), same to _mapper.count.
    NSUInteger _keyMappedCount;
    /// Model class type.
    YYEncodingNSType _nsType;

    BOOL _hasCustomWillTransformFromDictionary;
    BOOL _hasCustomTransformFromDictionary;
    BOOL _hasCustomTransformToDictionary;
    BOOL _hasCustomClassFromDictionary;
}
@end
```

```objc
+ (instancetype)yy_modelWithDictionary:(NSDictionary *)dictionary {
    if (!dictionary || dictionary == (id)kCFNull) return nil;
    if (![dictionary isKindOfClass:[NSDictionary class]]) return nil;

    Class cls = [self class];
    _YYModelMeta *modelMeta = [_YYModelMeta metaWithClass:cls];
    if (modelMeta->_hasCustomClassFromDictionary) {
//          查看是否需要特殊成不同的类
        cls = [cls modelCustomClassForDictionary:dictionary] ?: cls;
    }

    NSObject *one = [cls new];
    if ([one yy_modelSetWithDictionary:dictionary]) return one;
    return nil;
}
```

```objc
if (modelMeta->_keyMappedCount >= CFDictionaryGetCount((CFDictionaryRef)dic)) {

    对key value设置
    CFDictionaryApplyFunction((CFDictionaryRef)dic, ModelSetWithDictionaryFunction, &context);

    @{@"name":@"user.name"}
    if (modelMeta->_keyPathPropertyMetas) {
        CFArrayApplyFunction((CFArrayRef)modelMeta->_keyPathPropertyMetas,
                             CFRangeMake(0, CFArrayGetCount((CFArrayRef)modelMeta->_keyPathPropertyMetas)),
                             ModelSetWithPropertyMetaArrayFunction,
                             &context);
    }
    //对应多
    //        @{
    //          @"name" : @"name",
    //          @"fullName" : @"name",
    //          @"username" : @"name"
    //          }

    if (modelMeta->_multiKeysPropertyMetas) {
        CFArrayApplyFunction((CFArrayRef)modelMeta->_multiKeysPropertyMetas,
                             CFRangeMake(0, CFArrayGetCount((CFArrayRef)modelMeta->_multiKeysPropertyMetas)),
                             ModelSetWithPropertyMetaArrayFunction,
                             &context);
    }
} else {

    CFArrayApplyFunction((CFArrayRef)modelMeta->_allPropertyMetas,
                         CFRangeMake(0, modelMeta->_keyMappedCount),
                         ModelSetWithPropertyMetaArrayFunction,
                         &context);
}
```

```objc
/**
 Apply function for dictionary, to set the key-value pair to model.

 @param _key     should not be nil, NSString.
 @param _value   should not be nil.
 @param _context _context.modelMeta and _context.model should not be nil.
 */
static void ModelSetWithDictionaryFunction(const void *_key, const void *_value, void *_context) {
    ModelSetContext *context = _context;
    __unsafe_unretained _YYModelMeta *meta = (__bridge _YYModelMeta *)(context->modelMeta);
    __unsafe_unretained _YYModelPropertyMeta *propertyMeta = [meta->_mapper objectForKey:(__bridge id)(_key)];
    __unsafe_unretained id model = (__bridge id)(context->model);
    while (propertyMeta) {
        if (propertyMeta->_setter) {
            ModelSetValueForProperty(model, (__bridge __unsafe_unretained id)_value, propertyMeta);
        }
        propertyMeta = propertyMeta->_next;
    };
}
```

```objc
/**
 Apply function for model property meta, to set dictionary to model.

 @param _propertyMeta should not be nil, _YYModelPropertyMeta.
 @param _context      _context.model and _context.dictionary should not be nil.
 */
static void ModelSetWithPropertyMetaArrayFunction(const void *_propertyMeta, void *_context) {
    ModelSetContext *context = _context;
    __unsafe_unretained NSDictionary *dictionary = (__bridge NSDictionary *)(context->dictionary);
    __unsafe_unretained _YYModelPropertyMeta *propertyMeta = (__bridge _YYModelPropertyMeta *)(_propertyMeta);
    if (!propertyMeta->_setter) return;
    id value = nil;

    if (propertyMeta->_mappedToKeyArray) {
        value = YYValueForMultiKeys(dictionary, propertyMeta->_mappedToKeyArray);
    } else if (propertyMeta->_mappedToKeyPath) {
        value = YYValueForKeyPath(dictionary, propertyMeta->_mappedToKeyPath);
    } else {
        value = [dictionary objectForKey:propertyMeta->_mappedToKey];
    }

    if (value) {
        __unsafe_unretained id model = (__bridge id)(context->model);
        ModelSetValueForProperty(model, value, propertyMeta);
    }
}
```

# 代码段

```objc
typedef NS_OPTIONS(NSUInteger, YYEncodingType) {
    YYEncodingTypeMask       = 0xFF, ///< mask of type value 11111111
    YYEncodingTypeUnknown    = 0, ///< unknown
    YYEncodingTypeVoid       = 1, ///< void
    YYEncodingTypeBool       = 2, ///< bool
    YYEncodingTypeInt8       = 3, ///< char / BOOL
    YYEncodingTypeUInt8      = 4, ///< unsigned char
    YYEncodingTypeInt16      = 5, ///< short
    YYEncodingTypeUInt16     = 6, ///< unsigned short
    YYEncodingTypeInt32      = 7, ///< int
    YYEncodingTypeUInt32     = 8, ///< unsigned int
    YYEncodingTypeInt64      = 9, ///< long long
    YYEncodingTypeUInt64     = 10, ///< unsigned long long
    YYEncodingTypeFloat      = 11, ///< float
    YYEncodingTypeDouble     = 12, ///< double
    YYEncodingTypeLongDouble = 13, ///< long double
    YYEncodingTypeObject     = 14, ///< id
    YYEncodingTypeClass      = 15, ///< Class
    YYEncodingTypeSEL        = 16, ///< SEL
    YYEncodingTypeBlock      = 17, ///< block
    YYEncodingTypePointer    = 18, ///< void*
    YYEncodingTypeStruct     = 19, ///< struct
    YYEncodingTypeUnion      = 20, ///< union
    YYEncodingTypeCString    = 21, ///< char*
    YYEncodingTypeCArray     = 22, ///< char[10] (for example)

    YYEncodingTypeQualifierMask   = 0xFF00,   ///< mask of qualifier 1111111100000000
    YYEncodingTypeQualifierConst  = 1 << 8,  ///< const
    YYEncodingTypeQualifierIn     = 1 << 9,  ///< in
    YYEncodingTypeQualifierInout  = 1 << 10, ///< inout
    YYEncodingTypeQualifierOut    = 1 << 11, ///< out
    YYEncodingTypeQualifierBycopy = 1 << 12, ///< bycopy
    YYEncodingTypeQualifierByref  = 1 << 13, ///< byref
    YYEncodingTypeQualifierOneway = 1 << 14, ///< oneway

    YYEncodingTypePropertyMask         = 0xFF0000, ///< mask of property 111111110000000000000000
    YYEncodingTypePropertyReadonly     = 1 << 16, ///< readonly
    YYEncodingTypePropertyCopy         = 1 << 17, ///< copy
    YYEncodingTypePropertyRetain       = 1 << 18, ///< retain
    YYEncodingTypePropertyNonatomic    = 1 << 19, ///< nonatomic
    YYEncodingTypePropertyWeak         = 1 << 20, ///< weak
    YYEncodingTypePropertyCustomGetter = 1 << 21, ///< getter=
    YYEncodingTypePropertyCustomSetter = 1 << 22, ///< setter=
    YYEncodingTypePropertyDynamic      = 1 << 23, ///< @dynamic
};
```

# setter 组装

```
    _type = type;
    if (_name.length) {
        if (!_getter) {
            //组装get方法
            _getter = NSSelectorFromString(_name);
        }
        if (!_setter) {
//              组装set方法
            _setter = NSSelectorFromString([NSString stringWithFormat:@"set%@%@:", [_name substringToIndex:1].
                uppercaseString, [_name substringFromIndex:1]]);
        }
    }
```

# 基本信息获取

```
//    获取类的方法列表，生成_methodInfos
    Method *methods = class_copyMethodList(cls, &methodCount);
    if (methods) {
        NSMutableDictionary *methodInfos = [NSMutableDictionary new];
        _methodInfos = methodInfos;
        for (unsigned int i = 0; i < methodCount; i++) {
            YYClassMethodInfo *info = [[YYClassMethodInfo alloc] initWithMethod:methods[i]];
            if (info.name) methodInfos[info.name] = info;
        }
        free(methods);
    }
    unsigned int propertyCount = 0;
//    获取property列表，生成_propertyInfos
    objc_property_t *properties = class_copyPropertyList(cls, &propertyCount);
    if (properties) {
        NSMutableDictionary *propertyInfos = [NSMutableDictionary new];
        _propertyInfos = propertyInfos;
        for (unsigned int i = 0; i < propertyCount; i++) {
            YYClassPropertyInfo *info = [[YYClassPropertyInfo alloc] initWithProperty:properties[i]];
            if (info.name) propertyInfos[info.name] = info;
        }
        free(properties);
    }

    unsigned int ivarCount = 0;
//    成员变量
    Ivar *ivars = class_copyIvarList(cls, &ivarCount);
    if (ivars) {
        NSMutableDictionary *ivarInfos = [NSMutableDictionary new];
        _ivarInfos = ivarInfos;
        for (unsigned int i = 0; i < ivarCount; i++) {
            YYClassIvarInfo *info = [[YYClassIvarInfo alloc] initWithIvar:ivars[i]];
            if (info.name) ivarInfos[info.name] = info;
        }
        free(ivars);
    }
```

# 线程安全&缓存

```objc
+ (instancetype)classInfoWithClass:(Class)cls {
    if (!cls) return nil;

    static CFMutableDictionaryRef classCache;
    static CFMutableDictionaryRef metaCache;
    static dispatch_once_t onceToken;
    static dispatch_semaphore_t lock;
    dispatch_once(&onceToken, ^{
//        类的缓存信息，初始化一次
        classCache = CFDictionaryCreateMutable(CFAllocatorGetDefault(), 0, &kCFTypeDictionaryKeyCallBacks, &
            kCFTypeDictionaryValueCallBacks);
        metaCache = CFDictionaryCreateMutable(CFAllocatorGetDefault(), 0, &kCFTypeDictionaryKeyCallBacks, &
            kCFTypeDictionaryValueCallBacks);
        lock = dispatch_semaphore_create(1);
    });

//    信号量控制始终只有一个线程在操作类信息，lock -1 = 0，如果再有线程过来就需要等待
    dispatch_semaphore_wait(lock, DISPATCH_TIME_FOREVER);
//    获取类缓存
    YYClassInfo *info = CFDictionaryGetValue(class_isMetaClass(cls) ? metaCache : classCache, (__bridge const void *)(cls)
        );
    if (info && info->_needUpdate) {
        [info _update];
    }
//    lock+1=1，上面就可执行
    dispatch_semaphore_signal(lock);
    if (!info) {
        info = [[YYClassInfo alloc] initWithClass:cls];
        if (info) {
            dispatch_semaphore_wait(lock, DISPATCH_TIME_FOREVER);
//            设置缓存
            CFDictionarySetValue(info.isMeta ? metaCache : classCache, (__bridge const void *)(cls), (__bridge const void
                *)(info));
            dispatch_semaphore_signal(lock);
        }
    }
    return info;
}
```

# 时间转换

```
    NSDateFormatter *formatter2 = [NSDateFormatter new];
    formatter2.locale = [[NSLocale alloc] initWithLocaleIdentifier:@"en_US_POSIX"];
    formatter2.dateFormat = @"yyyy-MM-dd'T'HH:mm:ss.SSSZ";

    blocks[20] = ^(NSString *string) { return [formatter dateFromString:string]; };
    blocks[24] = ^(NSString *string) { return [formatter dateFromString:string]?: [formatter2
        string]; };
    blocks[25] = ^(NSString *string) { return [formatter dateFromString:string]; };
    blocks[28] = ^(NSString *string) { return [formatter2 dateFromString:string]; };
    blocks[29] = ^(NSString *string) { return [formatter2 dateFromString:string]; };
}

{
    /*
     Fri Sep 04 00:12:21 +0800 2015 // Weibo, Twitter
     Fri Sep 04 00:12:21.000 +0800 2015
     */
    NSDateFormatter *formatter = [NSDateFormatter new];
    formatter.locale = [[NSLocale alloc] initWithLocaleIdentifier:@"en_US_POSIX"];
    formatter.dateFormat = @"EEE MMM dd HH:mm:ss Z yyyy";

    NSDateFormatter *formatter2 = [NSDateFormatter new];
    formatter2.locale = [[NSLocale alloc] initWithLocaleIdentifier:@"en_US_POSIX"];
    formatter2.dateFormat = @"EEE MMM dd HH:mm:ss.SSS Z yyyy";

    blocks[30] = ^(NSString *string) { return [formatter dateFromString:string]; };
    blocks[34] = ^(NSString *string) { return [formatter2 dateFromString:string]; };
}
});
if (!string) return nil;
if (string.length > kParserNum) return nil;

//根据字符串的长度，进行转换成nsdate，如@"yyyy-MM-dd" length = 10
YYNSDateParseBlock parser = blocks[string.length];
if (!parser) return nil;
return parser(string);
#undef kParserNum
```

# 类型的适配

```objc
//类型转换
+ (NSDictionary *)_yy_dictionaryWithJSON:(id)json {
    if (!json || json == (id)kCFNull) return nil;
    NSDictionary *dic = nil;
    NSData *jsonData = nil;
    if ([json isKindOfClass:[NSDictionary class]]) {
        dic = json;
    } else if ([json isKindOfClass:[NSString class]]) {
        jsonData = [(NSString *)json dataUsingEncoding : NSUTF8StringEncoding];
    } else if ([json isKindOfClass:[NSData class]]) {
        jsonData = json;
    }
    if (jsonData) {
        dic = [NSJSONSerialization JSONObjectWithData:jsonData options:kNilOptions error:NULL];
        if (![dic isKindOfClass:[NSDictionary class]]) dic = nil;
    }
    return dic;
}
```

# 数组字典的遍历

```
CFDictionaryApplyFunction((CFDictionaryRef)dic, ModelSetWithDictionaryFunction, &context);

  @{@"name":@"user.name"}
if (modelMeta->_keyPathPropertyMetas) {
    CFArrayApplyFunction((CFArrayRef)modelMeta->_keyPathPropertyMetas,
                         CFRangeMake(0, CFArrayGetCount((CFArrayRef)modelMeta->_keyPathPropertyMetas)),
                         ModelSetWithPropertyMetaArrayFunction,
                         &context);
}
```

```
/*!
    @function CFDictionaryApplyFunction
    Calls a function once for each value in the dictionary.
    @param theDict The dictionary to be queried. If this parameter is
        not a valid CFDictionary, the behavior is undefined.
    @param applier The callback function to call once for each value in
        the dictionary. If this parameter is not a
        pointer to a function of the correct prototype, the behavior
        is undefined. If there are keys or values which the
        applier function does not expect or cannot properly apply
        to, the behavior is undefined.
    @param context A pointer-sized user-defined value, which is passed
        as the third parameter to the applier function, but is
        otherwise unused by this function. If the context is not
        what is expected by the applier function, the behavior is
        undefined.
*/
CF_EXPORT
void CFDictionaryApplyFunction(CFDictionaryRef theDict, CFDictionaryApplierFunction CF_NOESCAPE applier, void *context);
```

```
/*!
    @function CFArrayApplyFunction
    Calls a function once for each value in the array.
    @param theArray The array to be operated upon. If this parameter is not
        a valid CFArray, the behavior is undefined.
    @param range The range of values within the array to which to apply
        the function. If the range location or end point (defined by
        the location plus length minus 1) is outside the index
        space of the array (0 to N-1 inclusive, where N is the count
        of the array), the behavior is undefined. If the range
        length is negative, the behavior is undefined. The range may
        be empty (length 0).
    @param applier The callback function to call once for each value in
        the given range in the array. If this parameter is not a
        pointer to a function of the correct prototype, the behavior
        is undefined. If there are values in the range which the
        applier function does not expect or cannot properly apply
        to, the behavior is undefined.
    @param context A pointer-sized user-defined value, which is passed
        as the second parameter to the applier function, but is
        otherwise unused by this function. If the context is not
        what is expected by the applier function, the behavior is
        undefined.
*/
CF_EXPORT
void CFArrayApplyFunction(CFArrayRef theArray, CFRange range, CFArrayApplierFunction CF_NOESCAPE applier, void *con
```

1. 缓存

Model JSON 转换过程中需要很多类的元数据，如果数据足够小，则全部缓存到内存中。

2. 查表

当遇到多项选择的条件时，要尽量使用查表法实现，比如 switch/case，C Array，如果查表条件是对象，则可以用 NSDictionary 来实现。

3. 避免 KVC

Key-Value Coding 使用起来非常方便，但性能上要差于直接调用 Getter/Setter，所以如果能避免 KVC 而用 Getter/Setter 代替，性能会有较大提升。

4. 避免 Getter/Setter 调用

如果能直接访问 ivar，则尽量使用 ivar 而不要使用 Getter/Setter 这样也能节省一部分开销。

5. 避免多余的内存管理方法

在 ARC 条件下，默认声明的对象是 __strong 类型的，赋值时有可能会产生 retain/release 调用，如果一个变量在其生命周期内不会被释放，则使用 __unsafe_unretained 会节省很大的开销。

访问具有 __weak 属性的变量时，实际上会调用 objc_loadWeak() 和 objc_storeWeak() 来完成，这也会带来很大的开销，所以要避免使用 __weak 属性。

创建和使用对象时，要尽量避免对象进入 autoreleasepool，以避免额外的资源开销。

6. 遍历容器类时，选择更高效的方法

相对于 Foundation 的方法来说，CoreFoundation 的方法有更高的性能，用 CFArrayApplyFunction() 和 CFDictionaryApplyFunction() 方法来遍历容器类能带来不少性能提升，但代码写起来会非常麻烦。

7. 尽量用纯 C 函数、内联函数

使用纯 C 函数可以避免 ObjC 的消息发送带来的开销。如果 C 函数比较小，使用 inline 可以避免一部分压栈弹栈等函数调用的开销。

8. 减少遍历的循环次数

在 JSON 和 Model 转换前，Model 的属性个数和 JSON 的属性个数都是已知的，这时选择数量较少的那一方进行遍历，会节省很多时间。

# 谢谢