

Json视图工具

[demo地址](#)

简介

描述



`BaseJsonViewController` 是一个用 OC 编写的提供了 搜索、插入、编辑、查看路径、复制 json/value 等功能的 Json 可视化编辑工具。

由于网络数据请求下来后，APP 端对 json 原数据的展示并不明朗。修改网络数据只能通过 Charles 等抓包工具实现，受到的限制太多，所以诞生了在 APP 端直接对 json 进行查看、修改的 Json 视图工具: `BaseJsonViewController`。

后续会对 `BaseJsonViewController` 进行持续的更新优化，欢迎使用。

主要功能

json结构展示：

1. 一键压缩/展开：点击  all 展开全部，点击  ... 压缩全部（需要注意的是，如果进行了压缩，处在插入状态的cell，将被删除）。
2. 添加了层级的背景色、缩进等。默认最大展示6个层级，如果超过6个层级则跳转到新的页面，进行展示。
3. 对类型的区分：分为 Dictionary Array String Number。
4. 支持展开与收起功能，如果有子节点，则单击可以展开\收起。
5. value的展示：一行cell 的 value默认最多展示两行。如果超过两行则压缩，并在底部展示。

搜索功能：

点击放大镜可以进入搜索页面

在源码中的位置：`BaseJsonViewController` -> `BaseJsonViewMainView` -> `BaseJsonHeaderView` -> `BaseJsonViewSearchView`



1. **搜索关键词**: 输入关键词, 并且会自动进行搜索。
2. **精准搜索**: 如果选中精准搜索, 搜索策略将从 `containsString` 变成 `isEqualToString`。

注意: 不管是否为精准搜索, 都区分大小写。

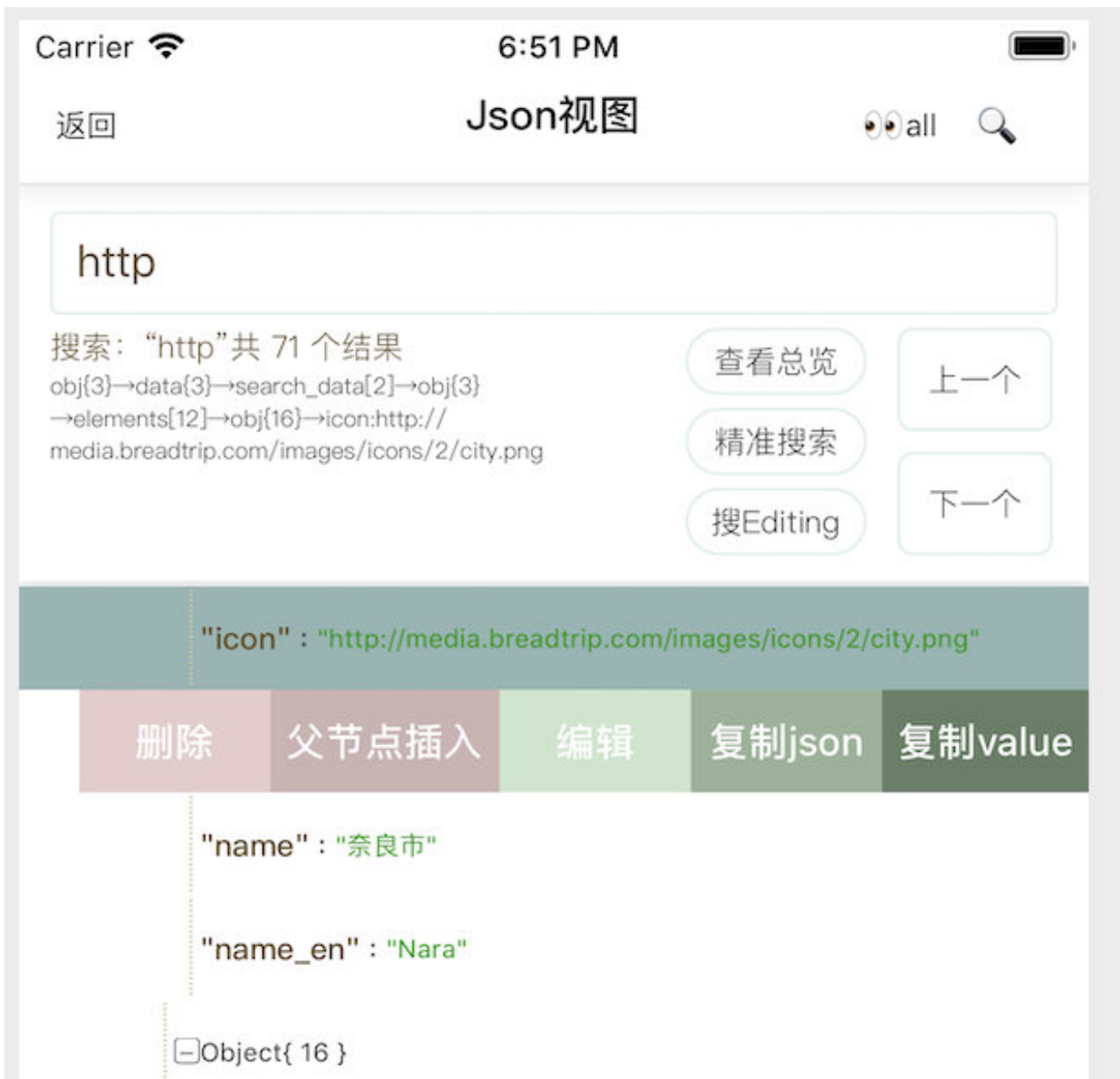
3. **搜索Editing**: 如果选中【搜Editing】按钮, 则会搜索整个 `json` 中处在 `Editing` 状态并符合关键词搜索的数据。

注意: 如果有处在 `插入状态` 的数据, 这时候会自动被删除。

4. **上一个\下一个**: 当搜索完成后, 点击【上一个】、【下一个】自动跳转到相应的行。
5. **查看总览**: 跳转到一个搜索结果总览控制器, 显示了搜索结果的路径及 ``value`
6. **展示路径/搜索数量**: 当没有搜索条件(即: 没有 `搜索词`、且 `搜Editing` 处于非选中状态)时, 显示的是本控制器节点的路径。否则显示的是搜索结果数量。
7. **展示路径/报错信息**: 具有滚动、放大功能, 最大放大倍数为1倍
 1. 当搜索条件报错时, 展示的是红色的报错信息。
 2. 当有搜索内容时, 展示的是当前选中的搜索结果的节点路径,

删除功能

侧滑cell, 出现删除功能(注意, 因为侧滑功能比较多, 所以在 `iphone5` 上面会导致删除功能被遮挡)



复制功能

复制功能分为两种：

1. 如果侧滑 cell 对应的节点为 `Array` 或 `Dictionary` 则会只能复制 `json`
2. 如果侧滑 cell 对应的节点为 `String` 或 `Number` 则可以复制 `json` 与 `value`

编辑功能

侧滑 cell，并点击 编辑 按钮开启编辑功能(下面把被编辑的节点称为 `Model`，把 `Model` 的父节点称为 `SuperModel`)。

1. `SuperModel` 类型对 `Model` 的 `key` 的影响：
 1. `SuperModel` 点为 `Array` 类型： `Model` 的 `key` 必须为空。
 2. `SuperModel` 为 `Dictionary` 类型： `Model` 的 `key` 必须有值。
2. 点击取消按钮：取消所有修改。
3. 点击完成按钮：

1. 选中 `Number` 按钮, 转成 `Numbser` 类型, 输入的值必须为数字, 否则会报错, 并在报错位置进行显示。
2. 选中 `String` 按钮, 转成 `String` 类型, 会有个默认值, 默认值为 `""`。
3. 选中 `json` 按钮:
 1. 如果 `Model` 为 `Array` 类型, 则会把 `jons` 解析出来作为 `Model` 的子节点数据
 2. 如果 `Model` 为 `Dictionary` 类型, 则会直接解析 `Json`, 如果 `json` 内包含一个对象则该对象作 `Model` 的数据, 把对象的 `key` 作为 `Model` 的 `key`。
4. 选中 `Dictionary` 按钮:
 1. 如果 `Model` 为 `Dictionary` 类型, 则不会产生任何效果, 否则 `Model` 清空子节点数据, 并把 `Model` 转成 `Dictionary` 类型。
5. 选中 `Array` 按钮:

如果 `Model` 为 `Array` 类型, 则不会产生任何效果, 否则 `Model` 清空子节点数据, 并把 `Model` 转成 `Array` 类型。

插入功能

侧滑 `cell`, 并点击 `插入` 按钮开启编辑功能

注意: 如果在 `插入` 的 `节点` 没有点击完成的情况下, 对 `节点` 的 `父节点` 执行 `收起` 操作, 会自动删除刚刚插入的 `节点`

注意: 如果 `插入节点` 的 `父节点` 为 `Dictionary` 类型, `插入的节点` 在 `父节点` 中的 `顺序` 不固定

把被编辑的节点称为 `Model`。

把 `Model` 的父节点称为 `SuperModel`。

把 `Model` 插入的子节点称为 `SubModel`。

把 `SuperModel` 插入的子节点称为 `SuperSubModel`。

1. 如果 `Model` 为 `Dictionary` 则可以【插入子节点】或【插入父节点】。
 1. 【插入子节点】:
 1. 如果 `Model` 为 `关闭` 状态, 则自动展开 `Model`, 并在 `Model` 字节点的第一行插入一个新的节点 `SubModel`, 这时候, `SubModel` 处于被 `编辑` 状态。
 2. **注意:** 此时插入的 `SubModel` 在父节点 `Model` `无序`
 2. 【插入父节点】: 在 `Model` 的后面插入为 `SuperSubModel` 插入 `SuperSubModel`
2. 如果 `Model` 为 `Array` 类型, 则可以【插入子节点】或【插入父节点】。
 1. 【插入子节点】:
 1. 如果 `Model` 为 `展开` 状态, 则自动压缩 `Model`, 并在 `Model` 字节点的第一行插入一个新的节点 `SubModel`, 这时候, `SubModel` 处于被 `编辑` 状态。

2. 注意：此时插入的 `SubModel` 在父节点 `Model` 有序。
2. 【插入父节点】：在 `Model` 的后面插入为 `SuperSubModel` 插入 `SuperSubModel`
3. 如果 `Model` 为 `String` 或 `Number` 类型，则可以【插入父节点】。在 `Model` 的后面插入为 `SuperSubModel` 插入 `SuperSubModel`

实现思路

1. 对 `json` 的解析
 1. 为了避免造成不必要的开销，对 `json` 解析的时机做了调整：
 1. 当节点 `A` 被打开时候，才会解析 `A` 的子节点数据。
 2. 在解析节点 `A` 数据时，优先获取缓存的 `A` 子节点数据。
 3. 在对 `A` 进行 编辑 或 插入 时，对 `A` 的的子节点数据进行更新。
2. 对视图的展示
 1. 对与无限层级缩放的视图来说，我们有必要把数据展平。
 2. 数据中创建一个用于标记层级的变量。来做一个无限缩放层级的假象。

实现细节

对于节点Model的定义

`Model` 就代表了一个节点，所以 `Model` 的结构至关重要。

主要的属性：

1. `level`：所处层级,在进行初始化时，根据父节点的 `level` 进行赋值。

```
@property (nonatomic,assign) NSInteger level;
```

2. `count`：子节点的个数

```
@property (nonatomic,assign) NSInteger count;
```

3. `isOpen`是否为打开状态

```
@property (nonatomic,assign) BOOL isOpen;
```

4. `originData`：所有子节点的原始数据(可能为`nil`、`Array`、`Dictionary`、`Number`、`String`)

```
@property (nonatomic,strong) id originData;
```

5. key: 如果originData为字典, 则key就是originData的key, 否则为nil

```
@property (nonatomic,strong) NSString *key;
```

5. data: originData 转化成的数据(可能为: nil、NSString、NSArray、BaseJsonViewStepModel)

```
@property (nonatomic,strong) id data;
```

6. originData: 父节点(在父节点创建子节点时, 进行的赋值)

```
@property (nonatomic,weak) BaseJsonViewStepModel *superPoint;
```

7. type: 当前节点的类型

```
typedef enum : NSUInteger {  
    BaseJsonViewStepModelType_Dictionary,  
    BaseJsonViewStepModelType_Array,  
    BaseJsonViewStepModelType_Number,  
    BaseJsonViewStepModelType_String,  
} BaseJsonViewStepModelType;  
  
@property (nonatomic,assign) BaseJsonViewStepModelType type;
```

8. 所处的状态

```
typedef enum : NSUInteger {  
    BaseJsonViewStepCellStatus_Normal,  
    BaseJsonViewStepCellStatus_EditingSelf,  
    BaseJsonViewStepCellStatus_InsertItem,  
} BaseJsonViewStepCellStatus;  
  
@property (nonatomic,assign) BaseJsonViewStepCellStatus status;
```

对model的创建

1. + (BaseJsonViewStepModel *) createStepModelWithOriginData: (id) data andKey: (NSString *)key

```

/**
 创建 一个model

@param data 原始的子节点数据
@param key 创建出的model对应的key
@return model
*/
+ (BaseJsonViewStepModel *) createStepModelWithOriginData: (id) data andKey:
(NSString *)key{
    BaseJsonViewStepModel *model = [BaseJsonViewStepModel new];
    model.originData = data;
    model.key = key;
    return model;
}

```

2. + (BaseJsonViewStepModel (^)(id)) createWithID

类方法，返回一个block，block 传入的是id类型的数据。数据可以是

1. BaseJsonViewStepModel: 直接返回这个data。不再创建
2. NSString: 先转成字典，然后创建model

使用方法 BaseJsonViewStepModel.createWithId(data);

```

+ (BaseJsonViewStepModel (^)(id)) createWithID {
    return ^(id data) {
        BaseJsonViewStepModel *model;
        if ([data isKindOfClass:BaseJsonViewStepModel.class]) {
            model = data;
        }
        if ([data isKindOfClass:NSString.class]) {
            NSDictionary *dic =
BaseJsonViewManager.convertToDicWithJson(data);
            if (dic) {
                model = BaseJsonViewManager.convertToStepModelWithDic(dic);
            }
        }
        if (!model) {
            model = [BaseJsonViewStepModel createStepModelWithOriginData:data
andKey:@""];
        }
        return model;
    };
}

```

搜索功能

搜索功能将会搜索出 所有的符合条件的model，并返回一个数组

`isSearching` 的筛选策略

1. `isSearching`: 如果为true。
 1. 如果 `key` 为 `nil`，则搜索全部处在编辑状态的model。
 2. 如果 `key` 有值
 1. 如果 `isAccurateSearch` 为true: 搜索所有 `key` 或 `value` `isEqualToString` `key` 的正在编辑状态的 `model`
 2. 如果 `isAccurateSearch` 为false: 搜索所有 `key` 或 `value` `containsString` `key` 的正在编辑状态的 `model`

```
/**
 搜索

  @param key 搜索 关键字
  @param isAccurateSearch 是否为精准搜索（如果选中精准搜索，搜索策略将从
  `containsString` 变成 `isEqualToString`。不管是否为精准搜索，都区分大小写）
  @param isSearching 是否搜索正在编辑状态的model
  @return 搜索结果
 */
- (NSMutableArray <BaseJsonViewStepModel *>*) searchWithKey:(NSString *)key
andIsAccurateSearch: (BOOL) isAccurateSearch andIsSearching:(BOOL)
isSearching {
    SBaseJsonViewStepSearchModelConfig config;
    config.isSearching = isSearching;
    config.isAccurateSearch = isAccurateSearch;
    config.key = key;
    config.model = self;
    return BaseJsonViewStepSearchModel.getResultwithSearchConfig(config);
}
```

删除功能

从父节点移除本节点

这个功能主要是找到originData中相同的元素，进行删除。

```
- (void) removeFromSuper {

    if ([self.superPoint.originData isKindOfClass:NSArray.class]) {
        NSArray *array = self.superPoint.originData;
        NSMutableArray *arrayM = [[NSMutableArray alloc] initWithArray:array];
```



```

        [arrayM removeObject:self.originData];
        self.superPoint.originData = arrayM;
    }
    if ([self.superPoint.originData isKindOfClass:NSDictionary.class]) {
        NSDictionary *dic = self.superPoint.originData;
        NSMutableDictionary *dicM = [[NSMutableDictionary
alloc] initWithDictionary:dic];
        NSString *key = self.key;
        if (key.length > 0) {
            dicM[self.key] = nil;
        }
        self.superPoint.originData = dicM;
    }

    [self.superPoint reloadDataWithOriginDataProperty];
}

```

插入节点

根据原始数据插入节点，并返回 `BaseJsonViewStepErrorModel`。

`BaseJsonViewStepErrorModel` 记录了插入时的错误信息

```

/**
 插入一个节点

  @param key 节点的key
  @param originData 节点的原始子节点y数据
  @param index 插入的位置
  @return 插入报错的model
 */
- (BaseJsonViewStepErrorModel *) insertWithKey: (NSString *)key
    andOriginData: (id) originData
    andIndex:(NSInteger) index;

```

根据model插入节点，并返回 `BaseJsonViewStepErrorModel`。

`BaseJsonViewStepErrorModel` 记录了插入时的错误信息

```
/**
 插入一个Model

  @param model 准备插入的 节点 model
  @param index 插入的位置
  @return 错误信息
 */
- (BaseJsonViewStepErrorModel *) insertwithModel: (BaseJsonViewStepModel *)
model
                                andIndex:(NSInteger) index;
```

最后

彩蛋：点击title会复制当前Controller显示的json数据

工具刚刚成型，很多需要修改的地方，希望大家勇于提bug 谢谢~

[demo地址](#)