https://developer.apple.com/documentation/coreml/core\_ml\_api

Overview

In most cases, you interact only with your model's dynamically generated interface, which is created by Xcode automatically when you add a model to your Xcode project. You can use Core ML APIs directly in cases where you need to support custom workflows or advanced use cases. As an example, if you need to make predictions while asynchronously collecting input data into a custom structure, you can use that structure to provide input features to your model by adopting the **MLFeatureProvider** protocol.

通常，人们只对学习模型的动态接口感兴趣，这些动态接口可以在将一个模型添加到Xcode工程的时候由Xcode自动生成。

使用Core ML APIs可以支持自定义的工作流，或者其他增强场景。比如，想要对一组特定结构的异步输入数据做预测，只要让模型符合MLFeatureProvider协议，然后将这个特定结构作为这个模型的输入即可。

Machine Learning Model

**MLModel**

https://developer.apple.com/documentation/coreml/mlmodel

An encapsulation of all the details of your machine learning model.

机器学习模型详细信息的封装是MLModel

Overview

In many cases, you can use Core ML without accessing the MLModel class directly. Instead, the automatically generated model class (named after your .mlmodel file) provides a programmer friendly interface that you should use. If you do need an MLModel instance, use the model property on the automatically generated model class.

However, if you create your own MLFeatureProvider, you need to use the MLModel prediction(from:) or prediction(from:options:) directly.

通常情况下，人们可以直接使用Core ML，而不用直接使用MLModel类。然而，自动生成的模型类提供了易编程的接口。

然而，如果想要创建自己的MLFeatureProvider，就需要直接使用prediction(from:)或者prediction(from:options:)

1、模型的构造器（Initializer）

Creating a Model，模型创建

不使用Xcode自动生成接口时，使用如下声明创建一个MLModel

**convenience init(contentsOf url: URL) throws//convenience是便利构造器，throws表示这个方法可能会抛出一个异常，调用时就需要使用try？来捕获可能抛出的异常。**

URL:A value that identifies the location of a resource, such as an item on a remote server or the path to a local file.

URL是资源存放文职的标识。例如一个本地文件的路径就是一个URL。

init是构造器的关键字，参数contentsOf是外部参数名称，url是内部参数名称，：URL表示参数的类型

【备注】可以定义便利构造器来调用同一个类中的指定构造器，并为其参数提供默认值。你也可以定义便利构造器来创建一个特殊用途或特定输入的实例，便利构造器定义如下

convenience init(parameters) {

statements

}

Parameters

contentsOf（外部参数名称）：The path to your .mlmodelc file, the compiled version of your .mlmodel file.

参数是.mlmodelc文件的路径，这些文件是由 .mlmodel文件编译后生成的译文。

2、模型的编译方法（**类型属性方法Type Method**，此方法只能类本身调用，而不能有类的实例调用）

compileModel(at:)，编译模型

Compiles a model on the device to update the model in your app.

在设备上编译一个模型，用于更新APP中的模型。使用如下定义：

**class func compileModel(at modelURL: URL) throws -> URL//class表示类的类型属性**

编译模型解释：声明函数compileModel，外部参数名称是at，内部参数名称是modelURL，参数类型是URL(**即是下载好的.mlmodel文件的路径**），函数可以投入一个错误，函数返回值是一个URL（**即是编译好的.mlmodelc文件的路径**）。

讨论：源文件.mlmodel文件必须在设备上，将使用compileModel编译好的.mlmodelc文件传递给init，就能初始化一个模型实例。

举例：

使用如下代码即将.mlmodel文件编译成.mlmodelc文件，并完成对应模型model的实例化。

**let compiledUrl = try MLModel.compileModel(at: modelUrl)**//声明常量compiledUrl ，取值是调用类MLMODEL的方法，将.mlmodel文件编译成.mlmodelc文件的路径，使用try是因为MLMODEL类的方法compileModel可能抛出错误。

**let model = try MLModel(contentsOf: compiledUrl)**//声明变量model，取值是用刚刚编译生成的.mlmodelc实例化出来的MLMODEL类。使用try是因为MLMODEL类的构造器可能抛出错误。

【备注】类型属性是作为类型定义的一部分，类似于实例的属性，类型属性的访问也是通过点运算符(.)来进行。但是，**类型属性是通过类型本身来获取和设置**，而不是通过实例

struct StudMarks {

static let markCount = 97

static var totalCount = 0

var InternalMarks: Int = 0 {

didSet {

if InternalMarks > **StudMarks.markCount** {

InternalMarks = **StudMarks.markCount**

}

if InternalMarks > **StudMarks.totalCount** {

**StudMarks.totalCount** = InternalMarks

}

}

}

}

3、模型的预测方法（**实例方法Instance Method，由类的实例调用**）

（1）根据给定的输入预测输出，定义如下

**func prediction(from input: MLFeatureProvider) throws -> MLFeatureProvider//MLFeatureProvider是一个协议或者类型**

入参input是这个模型做预测需要的所有数据。返回值是模型的预测结果。

（2）带条件的预测，定义如下

**func prediction(from input: MLFeatureProvider, options: MLPredictionOptions) throws -> MLFeatureProvider**

option是这个模型做预测需要设定的所有条件或者选项。

**【备注】1 ：**MLPredictionOptions是一个类，只有一个成员usesCPUOnly，是一个BOOL型数据，用于说明计算是否只有CPU完成。

usesCPUOnly（实例属性Instance Property），一个BOOL型变量，表明预测结果是否只有CPU计算得出。声明如下：

class MLPredictionOptions {

**var usesCPUOnly: Bool { get set }//类型声明后加上{ set get }来表示属性是可读可写的**

}

**【备注】2** ：MLFeatureProvider是一个协议（Protocol），是一种接口协议，代表了模型一组特性值的集合。（An interface that represents a collection of feature values for a model.）

If you have a more complex data source, consider adding this protocol to your data source. The interface is straightforward to implement, being mainly an accessor for feature values. By implementing the protocol, you allow the MLModel to query your data sources without constructing a separate input instance. If your data is collected asynchronously, implement this protocol on your data structure. Or, if using the autogenerated interface leads to copying excessive amounts of data, use this protocol to integrate your data directly with the MLModel.

**MLMODEL可以直接访问符合MLFeatureProvider协议的数据，而不需要为此构造一个独立的输入接口。**

protocol MLFeatureProvider｛

var featureNames: **Set**<String> { get }//具有可读属性的变量成员featureNames，类型是set，get表示变量成员featureNames值具有可读属性

func featureValue(for featureName: String) -> MLFeatureValue?//协议的方法featureValue，入参是String类型的特性名成，返回值是可选MLFeatureValue型数据，表示如果特性名称没有值，就返回nil。

｝

**【备注】**3 set是一个数据类型，代表一组无重复元素的无序集合，例如：

let ingredients: **Set** = ["cocoa beans", "sugar", "cocoa butter", "salt"]

【备注】4 MLFeatureValue是类，是代表特性类型或者值的不变的实例，它的成员非常广泛。

class MLFeatureValue｛

//枚举

enum MLFeatureType{

case invalid = 0

case int64 = 1

case double = 2

case string = 3

case image = 4

case multiArray = 5

case dictionary = 6

}

/\*实例属性 begin\*/Instance Property

var type: MLFeatureType { get }

var isUndefined: Bool { get }

var int64Value: Int64 { get }

var doubleValue: Double { get }

var stringValue: String { get }

var dictionaryValue: [AnyHashable : NSNumber] { get }

var imageBufferValue: CVPixelBuffer? { get }

var multiArrayValue: MLMultiArray? { get }

/\*实例属性end\*/

/\*构造器begin\*/

convenience init(int64 value: Int64)

convenience init(double value: Double)

convenience init(string value: String)

convenience init(dictionary value: [AnyHashable : NSNumber]) throws

convenience init(pixelBuffer value: CVPixelBuffer)

convenience init(multiArray value: MLMultiArray)

convenience init(undefined type: MLFeatureType)

/\*构造器end\*/

//实例方法

func isEqual(to value: MLFeatureValue) -> Bool

｝

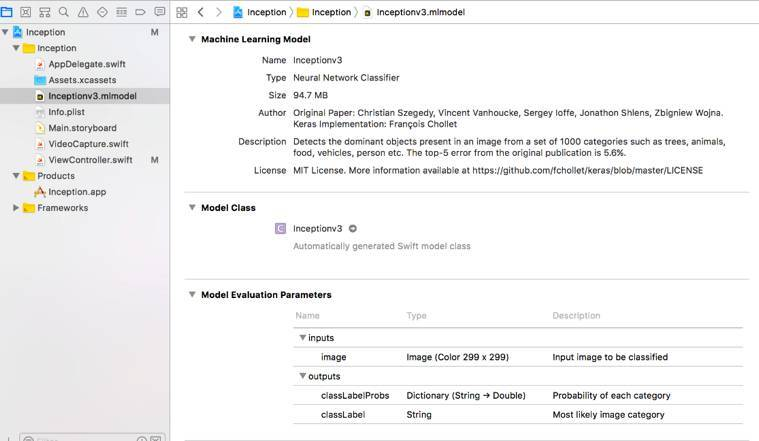
4、模型的描述（实例属性Instance Property）

这些数据是可以通过Xcode显示出来的模型的描述信息，声明如下：

**var modelDescription: MLModelDescription { get }//声明变量modelDescription，类型是MLModelDescription，具有只读属性，存储模型的描述信息**

【备注】5 MLModelDescription是一个类，是模型的描述信息，主要是模型的输入和输出的格式，还有预测需要的选项的信息。更多信息，参见https://developer.apple.com/documentation/coreml/mlmodeldescription

MLModelDescription的实例数据在Xcode上显示后，如下图，包括了模型的输入和输出，还有描述等信息。

****

至此可以推断出MLMODL的定义：

class MLMODEL ｛

var modelDescription: MLModelDescription { get }//声明变量modelDescription，类型是MLModelDescription，具有只读属性，是模型的描述信息

class func compileModel(at modelURL: URL) throws -> URL｛//类型属性方法，用于从.mlmodel文件编译得到.mlmodelc文件

statements

｝

convenience init(parameters) {//便利构造函数

statements

}

func prediction(from input: MLFeatureProvider) throws -> MLFeatureProvider//预测方法

func prediction(from input: MLFeatureProvider, options: MLPredictionOptions) throws -> MLFeatureProvider//预测方法

}