**CFET动态加载说明文档**

## 需求来源

之前CFET程序中Thing的功能和数量是固定的。开发起来没什么问题，但是部署使用的时候并不是很灵活，每个用户维护各自独立的程序，如果需要修改Thing就非常麻烦。

比如说，在采集程序中，一张采集卡是一个Thing。但使用时经常会增删采集卡，这就需要修改代码重新编译，无法真正交给管理和使用者。

动态加载实现后，无需在CFETApp代码中添加如MyHub.TryAddThing(…)之类的代码，而是直接通过建立一些文件结构来**在运行阶段**动态添加Thing。

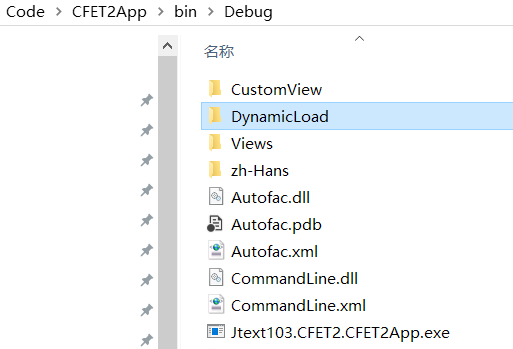
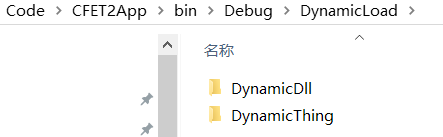
## 依赖关系：

动态加载的实现在CFETApp项目中，与Core和其它项目没有依赖关系

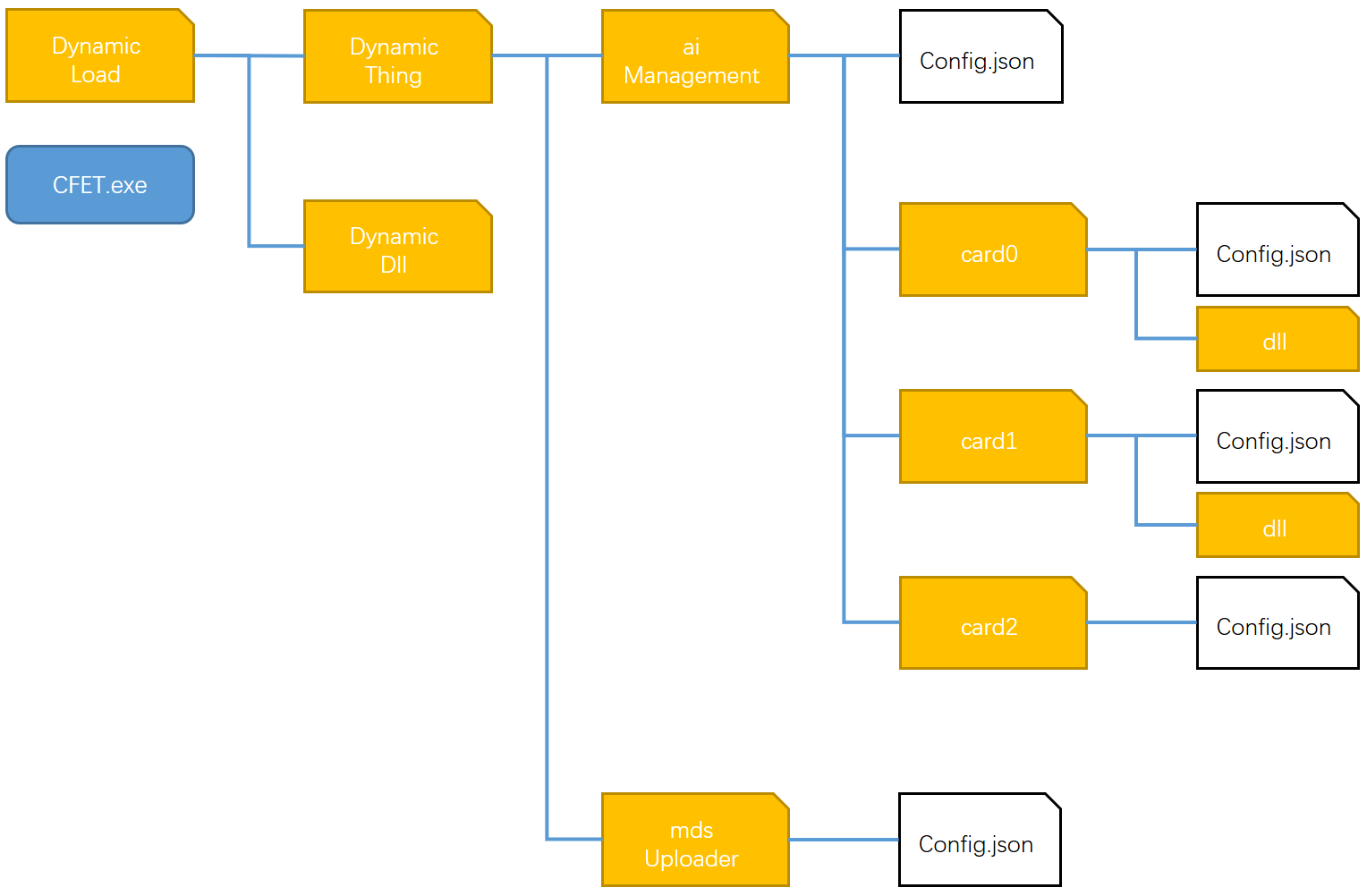
底层实现依赖Autofac Nuget包，V4.9.4，有官方说明文档

## 动态加载目录结构

动态加载需要在执行目录下建立名称固定为DynamicLoad的文件夹，并在其下建立DynamicThing和DynamicDll两个文件夹：

在CFET中所有动态加载的Thing都放在DynamicThing目录下。每个Thing需要建立一个文件夹，文件夹的名字代表Thing在CFET中的名字，文件夹在DynamicThing下的路径代表在CFET中的挂载路径。在Thing文件夹下必须有Config.json文件，来表示这个文件夹是为一个Thing，否则此文件夹不被作为Thing挂载。比如下图：

在这张图中，挂载的所有Thing和在CFET下的完整路径为：

/aiManagement

/mdsUploader

/aiManagement/card0

/aiManagement/card1

/aiManagement/card2

注意，在Thing的文件夹下可以选择性地建立dll文件夹来存放属于这个Thing的dll文件。程序会从DynamicDll和每个Thing下面的dll文件夹下动态加载所有dll。

## 具体实现

利用Autofac，可以在程序启动时将所有Thing的dll信息，包括Thing本身和Thing依赖的，加载到一个容器中。然后利用这个容器，和Config.json中保存的Thing的类型和参数信息，生成这个Thing的实例，并添加到CFET的Hub中去。

在Config.json中存放3个字段：

Thing在命名空间下的完整名字：

string Type

Thing的类型所在的dll的名字（仅文件名）：

string DllName

在TryAddThing时传入的初始化参数：

object InitObj

## 动态Thing的设计注意事项

动态加载的Thing在设计上和普通Thing并没有多大区别，但是有几个地方需要注意：

**1.Thing的所有依赖必须放在构造函数中：**

下面的Thing的设计是不允许的：

var niNonSync = new AIThing();

niNonSync.basicAI = new NIAI();

niNonSync.DataFileFactory = new HDF5DataFileFactory();

MyHub.TryAddThing(…);

需要采用如下形式：

var niNonSync = new AIThing(**new NIAI(), new HDF5DataFileFactory()**);

MyHub.TryAddThing(…);

注意，依赖项的依赖项也必须放在构造函数中。

**2.Thing的InitObj需要特别关注**

由于InitObj存在Config.json中，是一个JSON字符串，所有动态加载的设计是利用Newtonsoft.Json将其反序列化成一个Object后作为参数传入TryAddThing方法。由于反序列化后的类型可能比较特别，并不能保证可以转换成原始的数据类型，所以请慎重处理。这里建议在Thing中将这个InitObje参数重新先序列化成JSON，然后再反序列化，保证类型可以接受。

**3.依赖项目的特殊处理**

使用动态加载之后，Thing的依赖项的版本类型必须完全匹配。比如ThingA依赖RelyA V1.0，RelyA依赖RelyB V1.0。在非动态加载时，RelyB的dll版本为1.0或1.1等以上时可以顺利运行，但使用动态加载时RelyB的版本必须为1.0。

**4.Thing依赖配置文件并将配置文件路径作为参数**

动态加载时，可能理所当然会把Thing的配置文件放在Thing的动态加载文件夹下。注意，动态加载并不处理这种情况！动态路径需要Thing自行处理！