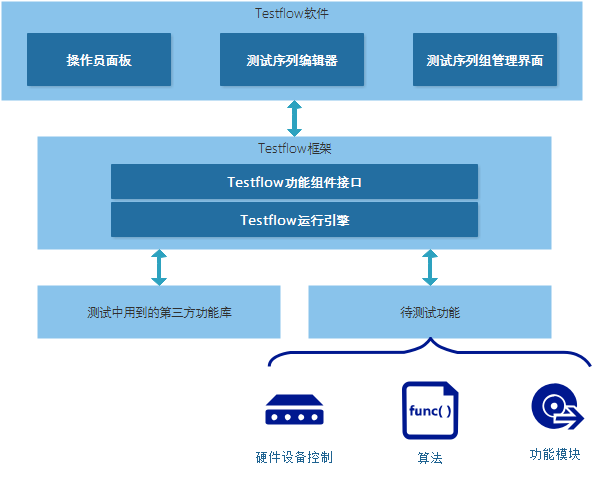
# Testflow的说明

Testflow是一个基于.NET平台的自动化测试框架，着眼于测试测量行业的自动化测试的设计与运行管理，可以实现多个测试用例从创建到结束的全套自定义配置、自动化流程控制和结果生成。该框架应实现的流程主要包括：用例设计、用例调度/运行/调试、报表生成、错误定位等。

## Testflow平台的设计规格

Testflow平台提供对硬件设备、算法、功能模块的功能性和性能的测试平台。该平台提供基于测试序列组的管理方式，并提供对应的测试序列编辑器，集成通用的测试执行引擎、操作面板、报表生成等功能。整个平台的架构如下所示。



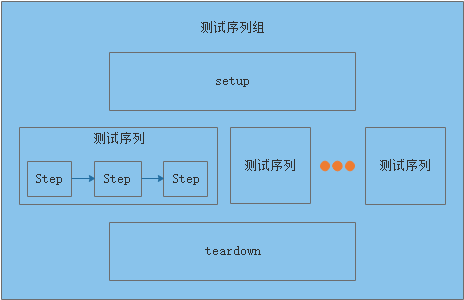
Testflow平台主要由两部分构成：Testflow框架和Testflow软件。

Testflow框架实现自动化测试的功能部分，主要包括Testflow功能组件接口(提供测试的管理模块、框架使用的辅助功能组件和对外调用接口，例如日志、报告生成等)、运行引擎(提供运行测试和维护状态的功能)。

Testflow软件实现对Testflow框架和测试的可视化管理。提供操作员面板、测试序列编辑器、测试序列管理等功能。

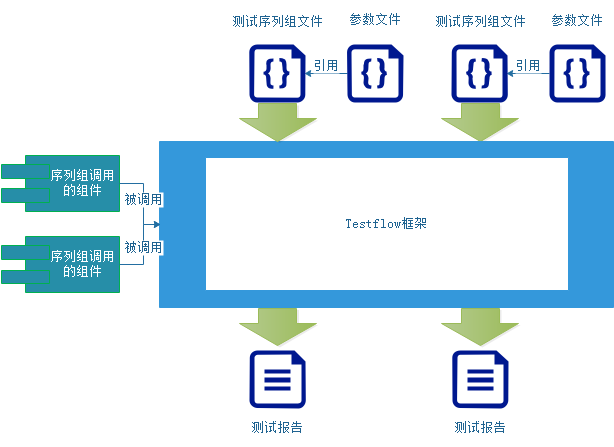
### Testflow的测试管理

TestFlow提供基于“测试序列组->测试序列->Step”的测试用例分层管理策略，这种管理策略的结构和说明如下所示：



* 测试序列组：测试序列组是多个相关联的测试序列组成的集合，完成某个对象的一套完整测试。测试序列组是Testflow框架能够调度运行的最小单元，不同的测试序列组在运行时通过AppDomain和线程互相隔离。一个测试序列组中包含了setup、多个互相关联的测试序列、teardown。
  + setup：用于初始化多个测试序列共用的资源和变量，功能依赖内部的多个step实现。用例组被执行时，setup始终是第一个被执行的过程，且在测试用例组的一个执行周期内只会被执行一次。如果setup模块执行失败，则所有用例组内的测试序列都不会被执行，并且会被标记为失败。该模块不支持Retry操作。
  + 待执行的测试序列：，一个或多个测试序列，这些序列在测试的功能上相互关联，共享相同的待测试资源。测试序列的功能由内部的多个step实现。
  + teardown：用于释放多个测试序列共用的资源和变量，功能依赖内部的多个step实现。用例组执行时，teardown始终是最后一个执行的过程，且在测试用例组的一个执行周期内只会被执行一次。该模块执行失败不会影响各个测试序列的测试结果。该模块不支持Retry操作。
* 测试序列：单个测试项流程，完成一个独立的测试用例的功能。一个测试序列由多个有序的Step串行执行完成。一个完整的测试序列应该包括：执行前置条件的校验、中间执行的功能、后置的结果检查。
* Step：完成单一功能，是Testflow框架中用户能够增删和配置的最小单元。step的功能由开发组件中的某个功能或者内置关键字实现，一个step可以提供的功能包括：
  + 功能库组件：实现单一功能和操作，例如某个算法，操作板卡执行数据采集等。
  + 用户交互组件：接收用户输入或者输出的组件。
  + 流程控制：简单的循环流程。
  + 一个完整测试流程：执行一个完整的测试流程。
  + 数据校验：执行某项数据的校验，用以判断测试是否通过。

在Testflow中，每个测试序列组会被保存为一个测试序列组文件(xml格式)，该测试序列组中所有的数据配置也会被保存为一个独立的参数文件(xml格式)。这些文件可以在Testflow提供的软件中被二次编辑。通过Testflow框架，这些文件可以被加载、运行、生成测试报告。测试序列组被调用的方式如下图所示：



### Testflow框架的设计规格

TestFlow框架设计规格的主要描述如下：

* 支持测试序列设计：可以通过Testflow软件工具编辑测试序列并序列化到文件，这些文件可以通过Testflow软件工具进行二次编辑。
* 提供测试序列组功能，可以将多个测试序列添加到测试序列组中。一个测试序列组是能够被运行引擎调用的最小单元。
* 可以同时运行一个或多个测试序列组，测试序列组之间可以互相独立的运行。一个测试序列组内部的多个用例可以串行的执行。在运行时需要保证足够的健壮性，不同的测试序列组在运行时层需要进行一定程度的隔离。
* 支持使用Testflow软件工具对测试序列以step粒度进行调试，支持基于Visual studio的代码级别的调试。
* 提供报表生成功能，报表内容包括：运行结果统计、关键信息打印等，其中该功能提供用户自定义接口，用户可以定制报表的生成方式与类型。同时报表的结果输出可以重定向，在报表的文件保存格式上支持常用的几种，例如txt和html(前期可以只实现txt方式)。
* 提供常用的外围开发组件，包括：常用硬件的配置与控制、用户数据交互(例如数据输入和信息显示)、常用算法、文件读写、其他方式的IO数据流读写等。这些组件在可视化设计的用例中也同时可用。同时框架应提供开发外围开发组件的接口。
* 提供日志功能，各个功能模块都需要嵌入日志功能以便错误定位。该日志功能接口同时需要提供给用例。
* 提供常用的错误码和错误信息映射。并将这些添加这种映射的接口提供给用户，被使用在外围开发组件中。
* Testflow可载入任意基于.NET开发的dll库作为开发组件，其中的类方法和静态方法都可以作为Testflow调用的step。Testflow提供额外的Attribute用以标记指定方法的可见性和运行时特征(该Attribute不会影响库的独立运行)。

### Testflow软件的设计规格

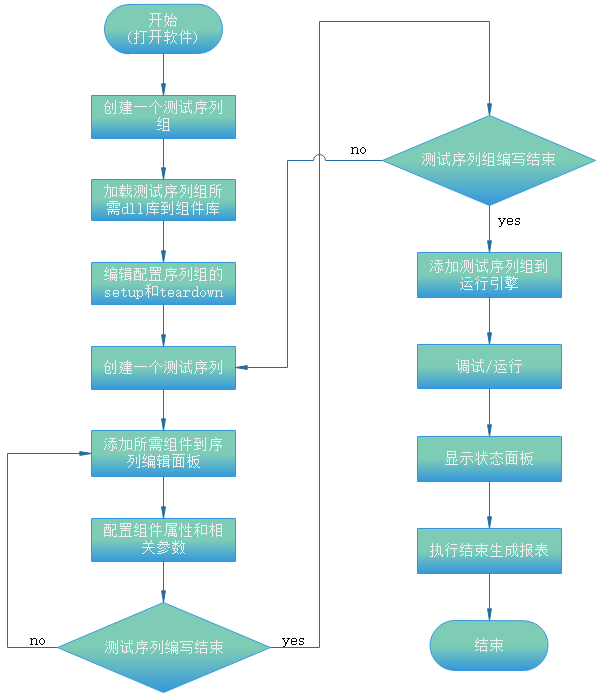
Testflow软件是基于Testflow框架的参考设计，提供基于Testflow框架从测试管理、设计用例、运行/调试、结果生成的可视化管理功能。软件一共有三种视图，分别提供测试序列编辑、调试、运行的可视化管理功能：

* 编辑视图：用于测试序列组的管理和编辑。
* 调试视图：用于测试序列组的调试，可以实时查看各个序列组当前执行的Step，并且支持在step的断点上查看当前的所有变量和参数的值
* 操作员面板(OI)：提供测试序列组运行控制、状态监控和报表查看的功能。

该软件中所有需要实现的窗体模块包括：测试序列组管理模块、测试序列组的编辑窗体模块、step级调试窗体模块、测试序列组运行窗体模块、运行状态显示窗体模块、结果显示模块。

* 测试序列组管理模块：可视化管理测试序列组的窗体模块，需实现的窗体功能包括：新建测试序列组、从文件打开测试序列组、保存测试序列组到文件、管理已执行过的测试序列组、管理当前所有打开的测试序列组。该模块在三种视图中都可见。
* 测试序列组的编辑窗体模块：编辑单个测试序列组的窗体，需实现的窗体功能包括：显示所有可用组件库、加载组件库到可用组件列表、显示所有组件库的所有功能方法、构造测试序列组全局参数、当前测试序列组显示、添加/删除测试序列组中某个step、配置某个step的相关参数。该视图在编辑视图中可见。
* 测试序列组运行窗体模块：和测试序列组运行相关的窗体，需实现的窗体功能包括：选择待运行的测试序列组、序列组的运行/调试/停止窗体功能。该视图在调试视图和操作员面板可见。
* step级调试窗体模块：和step级调试相关的窗体组件，包括：测试序列组当前执行step显示、在某个step上添加/删除断点、显示当前step各个参数的值。该视图在调试视图可见。
* 运行状态显示窗体模块：显示当前各个测试序列组运行状态的窗体，需实现的窗体功能包括：当前执行的状态和进度、资源的使用信息、运行中的控制台信息等。该视图在调试模块和操作员面板可见。
* 报表显示模块：显示各个测试序列组的运行结果，包括：成功失败的统计信息、资源使用的统计信息、失败和成功的说明信息。该视图在调试视图和操作员面板可见。

基于Testflow框架，在Testflow软件中创建编辑用例并运行的流程如下图所示：



关于Testflow软件的详尽设计参考文档“Testflow软件设计说明”。

## Testflow的部署、运行和调试的方式

Testflow的核心功能会以一组动态链接库的方式提供，同时还会提供一个用以编辑测试序列、执行、单个步骤调试的软件工具。提供OI的使用示例。

测试序列的运行有两种场景：在Testflow软件工具中的运行、不依赖Testflow软件工具的独立可执行文件的运行。后者可以通过Testflow软件工具生成。

Testflow的软件工具提供以step为最小粒度的调试功能，该调试功能可以提供各个变量的实时检查功能。为了实现代码级的调试，需要基于Visual studio的运行时调试功能，详细描述参见1.5节。

## 环境和语言支持

目前TestFlow应支持的环境信息如下，对其他环境的支持将后续开发：

|  |  |
| --- | --- |
| 操作系统 | Windows 7及以上版本(x86和x64) |
| 硬件配置 |  |
| 代码调试IDE | Visual Studio 2015及以上版本 |
| .NET版本 | .NET Framework 4.5及以上版本 |

TestFlow支持国际化功能，目前需至少提供英文和中文两种语言。需要国际化的信息至少应包括：提示信息、框架运行信息、错误信息、框架自带的报表功能、所有的内置日志信息。同时国际化接口应提供给用例。

## 关于Testflow的源码调试

Testflow提供测试序列中被调用组件库的源码调试功能，该功能需使用Visual studio的调试功能实现。Testflow提供专门用于调试的可执行文件，使用该可执行文件作为启动文件即可在Visual studio中调试有源码工程的组件。

使用方法说明：拷贝Testflow提供的调试可执行文件档到待调试工程的Debug目录下；使用Visual studio打开源码工程；在工程属性中配置启动文件为该可执行档；运行调试；在弹窗中选择待执行的测试序列组；在待调试位置增加断点即可对源码进行调试。

# Testflow框架的功能设计

Testflow框架主要用以完成自动化测试测量的序列组管理、测试序列组设计、运行/调试与结果管理，同时，Testflow框架还会提供测试开发组件的使用说明。该章节将分别说明这几个功能模块的设计预期和目标。

## 序列组管理的功能设计

序列管理模块主要完成测试序列组的创建、删除。从功能上

## 测试序列组设计的功能设计

一个序列中包含几个部分：对象的初始化和创建

## 运行/调试的功能设计

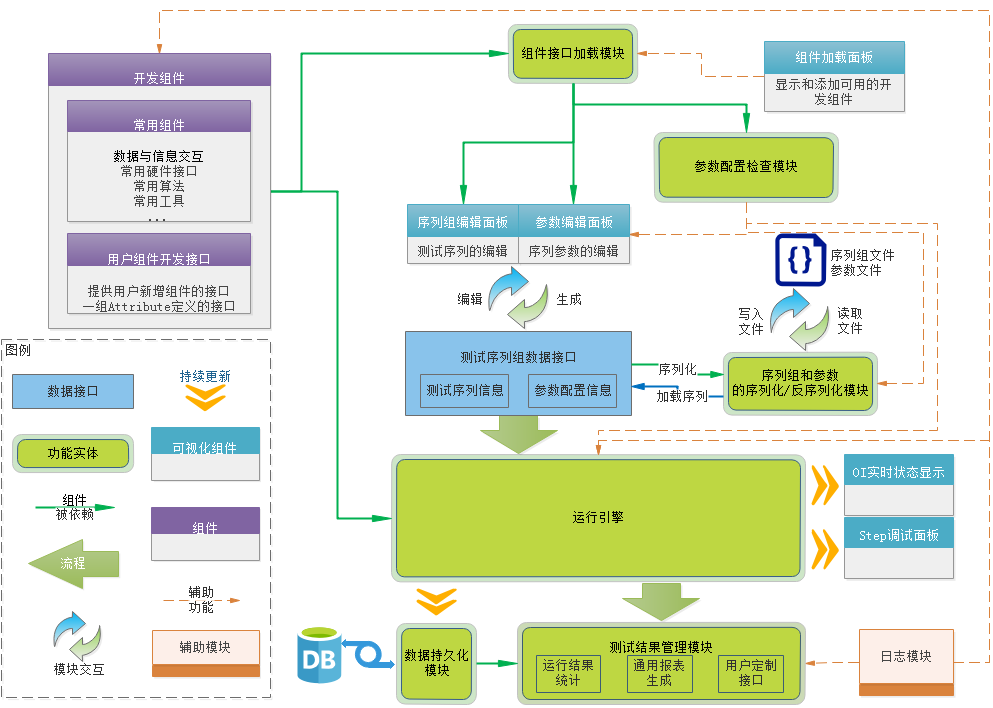
## 结果管理的功能设计

## 测试开发组件的说明

# Testflow的架构设计

## 逻辑架构设计

基于Testflow的设计规格和用例管理策略，Testflow框架基本的逻辑架构设计如下图所示：

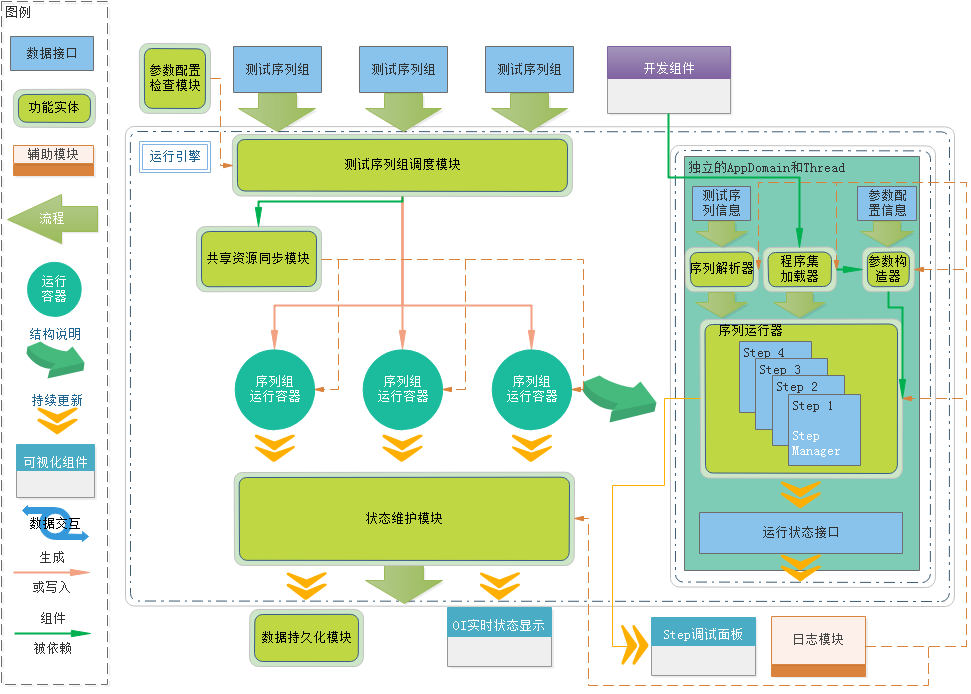


图中涉及到的功能模块如下说明：

* 开发组件：包含在dll中的用例开发组件，单个功能和相关参数配置构成测试序列的一个step。主要包括的功能类型有：
  + 常用硬件的接口：各种DAQ/DSA/Digitizer等的抽象接口以便于在GUI测试序列设计层使用，同时也能最大化保证测试用例的灵活性。
  + 其他常用组件：包括各种算法、文件读写、流读写。

同时该模块提供Testflow中会用到的各种Attribute标记(例如功能是否可见)，方便客户开发自己的组件库。

* 组件接口加载模块：用户通过组件加载面板选择待使用的库或者读取测试序列组文件后，加载对应库的可用功能接口，生成接口描述类。该类会被用于序列组编辑面板、参数编辑面板和参数配置检查模块。
* 序列组和参数的序列化/反序列化模块：该模块完成测试序列和参数配置的序列化和反序列化。上下行工作包括：
  + 读取测试序列文件和参数配置文件，生成测试序列组数据接口，可供序列组编辑器编辑或者运行引擎调用。
  + 将已编辑完成的测试序列组数据接口序列化，保存到序列组文件和参数文件中。
* 参数配置检查模块：检查用户从界面输入或者从文件读取的参数是否符合参数类型的模块，返回检查结构接口。该接口会被用于参数编辑面板、序列化反序列化模块和运行引擎。
* 运行引擎：独立运行用例组各个测试序列组的模块，维护各个测试序列组的运行状态，具备足够的健壮性。运行引擎的逻辑架构和各个子模块的说明如下所示：



* 测试序列组调度模块：加载对应测试序列组文件或者数据结构，执行参数检查，生成序列组的运行容器，并对这些序列组所在运行容器进行调度。
* 序列组运行容器：独立运行一个测试序列组的容器，基于AppDomain和线程进行隔离，可以避免对状态维护功能的干扰。该模块利用状态维护模块提供的一个持续更新接口，对状态维护模块中的运行状态数据进行实时更新。该容器还包含几个子模块和功能，如下所述：
  + 序列解析器：将测试序列组的结构解析为具体的流程，构造每个步骤的StepManager。
  + 程序集加载器：将测试序列组会用到的所有组件库加载到AppDomain中。
  + 参数构造器：根据参数结构中的数据，在运行时构造测试序列中用到的各种参数。
  + 序列运行器：加载序列解析器生成的StepManager列表，依次运行每个Step。在每个Step执行前调用参数构造器生成当前Step的参数，配置到StepManager中，然后执行Step。
  + 状态维护功能：实时将当前AppDomain的执行情况更新到状态维护模块中。
* 共享资源同步模块：该模块实现单个资源被多个测试序列组共用场景下的互斥访问功能。为了保证该模块的正常工作，该资源在组件库中定义时应该定义同步策略。
* 状态维护模块：实时获取各个序列运行容器执行状态，将这些状态数据写入持久化模块，同时提供当前的实时执行状态接口。同时构造状态统计集合结构，在测试序列组运行完成后，从数据持久化模块获取当前功能的所有状态信息，生成状态统计结果结构，保存到状态统计集合结构中。将集合数据结构传递到报表生成模块。
* 数据持久化模块：持久化各个测试用例执行过程中的状态信息，通过数据库实现持久化功能，前期主要基于SqlLite，后续可以增加MySql和SqlServer的支持。
* 报表生成模块：包含用例运行的统计信息、关键信息输出等。报表初期可以只支持文本方式，后续可以考虑加入结构化的数据管理和数据库。
* 日志模块：记录框架运行过程中的重要信息和用户需要打印的关键信息，用于故障定位和用例运行情况回溯。

## 数据架构设计

TestFlow中的数据主要包括：测试序列组内共享的数据、运行中的状态数据和最终的统计数据。

* 测试序列组内的共享数据：在运行时，这些共享数据存在于独立的AppDomain内，所以运行时只需要包装到参数类中，无需额外处理。在Testflow工具中进行调试时提供跨AppDomain的MarshalByRef的访问机制保证数据在调试窗口的可见性。
* 运行中的状态数据：跨AppDomain的状态信息通过数据持久化模块写入DB中，这些数据会被状态维护模块持续获取并持续更新更新面板上的状态信息。
* 最终的统计数据由状态维护模块通过数据库中已有数据统计并封装到状态和结果统计数据结构中，发布到报表模块。同时这些数据也会被更新到数据库中。

(详尽的数据接口列表在开发设计时补全)

## 开发架构设计

Testflow中所有的功能都基于模块化设计，模块之间通过定义的功能接口或者数据接口进行耦合。所有外部面板或者应用都需要借用由Framework提供的各个接口操作所有模块。同时Testflow会提供一组用户组件中会用到的接口，方便用户实现对组件的管理和同步控制等操作。

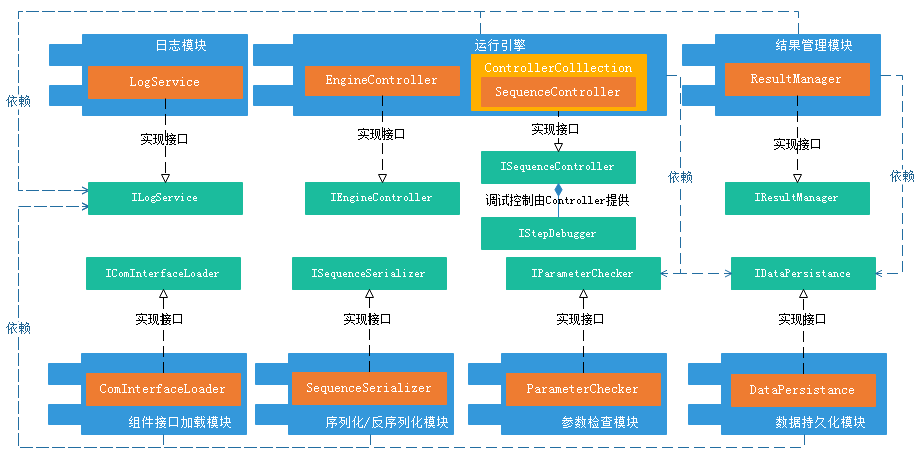
Testflow框架包括的主要模块控制类及功能描述如下所示：

|  |  |
| --- | --- |
| 名称 | 说明 |
| SequenceSerializer | 序列组文件的序列化和反序列控制 |
| ComInterfaceLoader | 组件接口加载模块控制 |
| ParametersChecker | 参数检查模块控制 |
| ConfigurationManager | 框架配置管理模块控制 |
| EngineController | 测试运行引擎控制，完成所有测试序列组的运行、状态维护。 |
| ResultsManager | 测试管理模块控制，生成相关测试的报表和统计等信息。 |
| LogService | 提供日志服务控制 |

框架中用到的功能接口主要包括：

|  |  |
| --- | --- |
| 名称 | 说明 |
| IComInterfaceLoader | 组件接口加载器的操作接口 |
| ISequenceSerializer | 测试序列组数据的序列化与反序列化接口 |
| IParameterChecker | 参数检查控制接口 |
| IStepDebugger | Step级调试的功能接口 |
| IDataPersistance | 数据持久化接口，操作数据库保存状态数据 |
| IEngineController | 运行引擎的控制接口 |
| ISequenceController | 单个测试序列组运行时的控制接口 |
| ILogService | 日志模块的读写接口 |
| IConfigurationManager | 框架配置管理接口 |
| IResultManager | 结果管理接口 |

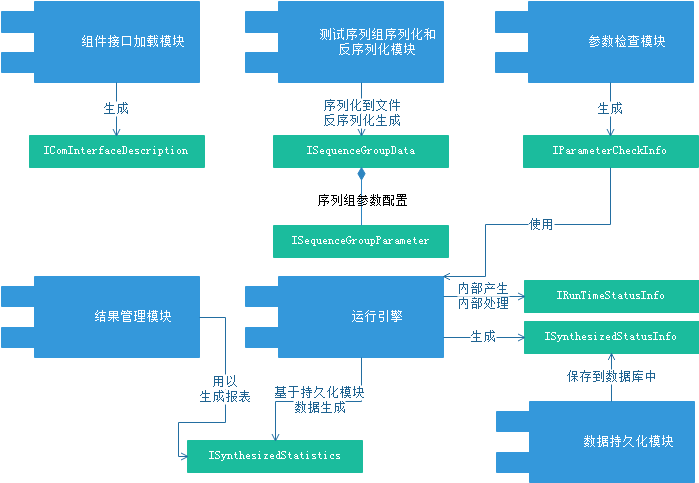
功能接口模块控制类和框架模块之间的关系如下图所示：



在框架中使用和对外开放的数据接口主要包括：

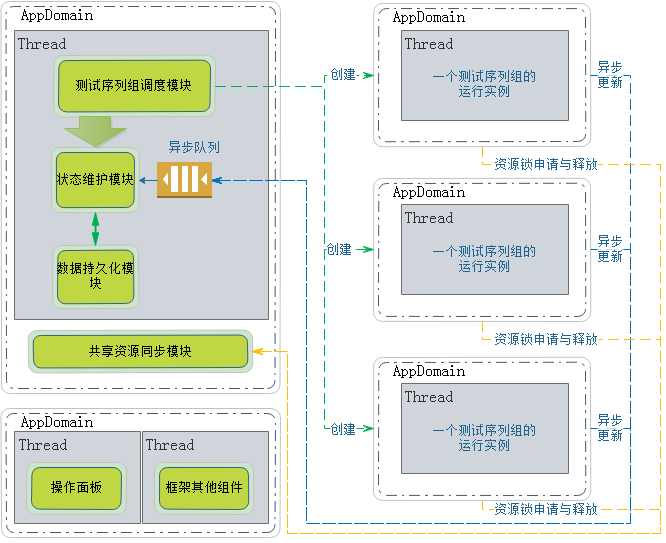
|  |  |
| --- | --- |
| 名称 | 说明 |
| IComInterfaceDescription | 组件接口描述类，保存某个组件接口的描述信息。该类由ComInterfaceLoader生成。 |
| ISequenceGroupData | 保存测试序列组信息的数据结构 |
| ISequenceCollection | ISequeceData的列表容器 |
| ISequenceData | 保存单个测试序列信息的数据结构 |
| ISequenceStepData | 保存测试序列中单个步骤的数据结构 |
| ISequenceGroupParameter | 保存测试序列组数据配置信息的数据结构 |
| IParameterCheckInfo | 保存参数检查结果信息的数据结构 |
| IRunTimeStatusInfo | 保存一个测试序列组运行时单个监视点的即时状态信息，运行引擎内部使用。 |
| ISynthesizedStatusInfo | 保存一个测试序列组运行时单个监视点的综合状态信息，提供给外部模块使用。 |
| ISynthesizedStatistics | 保存一个测试序列组的综合状态统计信息，该信息通过整合各个监视点的数据获得。 |
| IConfiguration | 框架配置数据接口 |

各数据接口和模块之间的关联关系如下图所示：



## 运行架构设计

Testflow的运行引擎在执行不同的测试序列组时提供基于AppDomain的隔离策略，在保证轻量级数据通信的情况下为不同的测试序列组提供最大程度的独立性。运行引擎和相关模块的运行架构如下图所示：



# Testflow的接口设计

Testflow的框架中包括多个数据接口和功能接口，所有的操作都通过这些接口进行控制。这些接口的名称和说明见底3.3节。该章分别就数据接口和功能接口进行详尽的定义和说明

## Testflow的数据接口定义

Testflow中的数据接口主要包括以下类型：序列组数据接口、序列设计相关数据接口、状态数据接口。

### 序列组数据接口定义

序列组数据接口包括序列组流程数据接口和参数配置接口。序列组流程数据接口ISequenceGroupData的定义如下：

|  |  |  |
| --- | --- | --- |
| 实现的接口 | | ICloneable, ISerializable |
| 属性 | 类型 | 说明 |
| Name | String | 序列组的名称 |
| Description | String | 序列组的描述信息 |
| Version | String | 序列组格式的版本号 |
| Hash | String | 序列组的哈希值，用来唯一确定一个序列组 |
| MD5 | String | 序列组当前MD5，每次更新流程都会被修改，和Hash一起决定序列组文件和参数文件是否匹配。 |
| CreationTime | DateTime | 创建日期 |
| ModifiedTime | DateTime | 修改日期 |
| Parameters | ISequenceGroupParameter | 序列组当前的所有参数配置，如果未配置则为null |
| SetUp | ISequenceData | 序列组的setup序列 |
| Sequences | ISequenceCollection | 当前测试序列组中的所有测试序列的集合 |
| TearDown | ISequenceData | 序列组的TearDown序列 |
| Modified | Bool | 序列组是否被更新的标识位 |
| 方法签名 | | 说明 |
| void Initialize() | | 初始化一个序列组文件 |
| ISequenceGroupData Clone() | | 测试序列组的深拷贝 |
| void Clear() | | 清空所有序列信息 |
| void RefreshSignature() | | 在序列组文件被修改结束后强制更新所有签名 |

序列集合接口ISequenceCollection的定义如下：

|  |  |  |
| --- | --- | --- |
| 实现的接口 | | ICloneable, ISerializable, IList<ISequenceData> |
| 属性 | 类型 | 说明 |
| Name | String | 序列组的名称 |
| Hash | String | 序列组的哈希值，用来唯一确定一个序列组 |
| MD5 | String | 序列组当前MD5，每次更新流程都会被修改，和Hash一起决定序列组文件和参数文件是否匹配。 |
| CreationTime | DateTime | 创建日期 |
| ModifiedTime | DateTime | 修改日期 |
| Parameters | ISequenceGroupParameter | 序列组当前的所有参数配置，如果未配置则为null |
| SetUp | ISequenceData | 序列组的setup序列 |
| Sequences | ISequenceCollection | 当前测试序列组中的所有测试序列的集合 |
| TearDown | ISequenceData | 序列组的TearDown序列 |
| Modified | Bool | 序列集合是否被更新的标识位 |
| 方法签名 | | 说明 |
| void Initialize() | | 初始化一个序列集合 |
| ISequenceCollection Clone() | | 测试序列组的深拷贝 |
| void Clear() | | 清空所有序列集合数据 |
| void RefreshSignature() | | 在序列组文件被修改结束后强制更新所有签名 |

### 序列设计相关数据接口定义

### 状态数据接口定义

## Testflow的功能接口定义

# Testflow的模块设计

序列组的xml格式设计

数据库设计

(开发时完善)