TCM技术应用文档

版本：2013/11/6（重制 3 修订 30）

此规范用于说明TCM标准化开发，是TCM SDK的重要组成部分。

目录

1. 起始

TCM 设计

TCM 是一种高性能、低资源占用的组件化软件开发技术，

支持跨语言开发，能够跨平台运行，支持组件操作注入，支持任务式调用。

字面上，TCM 指 Task（任务）+Component（组件）+Model（模型）。

其中表达了这样的涵义：

功能可被归类在组件中，组件调用可被抽象为任务，组件可以参与复杂建模。

在软件项目的整个开发周期中应用 TCM 技术，

能够显著提高开发效率，有效降低架构耦合程度和项目复杂度。

以下简述 TCM 技术的关键机制，具体请参考其他有关 TCM 特性的文档。

驱动机制 与 跨语言和跨平台特性

开发人员可以根据自己的需要，选择任何支持的语言开发 TCM 组件，

任何 TCM 组件可以用于任何一种支持的执行环境。

开发人员也可以按照 TCM 的相关规范，开发用于所需特定语言和平台的驱动。

插件机制 与 组件操作注入特性

“组件操作注入”指开发人员在组件的功能的执行过程中将某些对象与组件的执行过程关联。

TCM 遵循 AOP 思想，提供了特性插件机制。

开发人员可以根据自己的需要，在运行期间为执行过程注入特性对象。

特性对象包括日志、计时器等。

宿主机制 与 任务式组件调用特性

开发人员可以通过任务式调用隔离应用与组件。

在保证效率和功能不损失的前提下，进一步提高应用的稳定性，

并可以适用于更多样复杂的集成场景。

以下“组件”皆代表“基于 TCM 的组件”。

Vapula 框架

当前 SDK 是 TCM 技术思想的可能实现方案之一，该实现的代号是 Vapula。

其中包含这些内容：

1. 用于所支持的开发语言的编译期依赖和运行时
2. 相关开发文档及示例
3. 辅助开发与测试的工具

扩展包提供这些内容：

1. 服务端软件，用于提供 Web 可用的 TCM 服务。
2. 建模与再发布软件，用于提供基于图的快速应用开发。

另有一些基于 TCM 开发的业务组件包，

具体请参考这些组件包各自的说明。

注意，业务包中许多组件不是 Vapula 的组成部分。

授权

Vapula 框架是开源的 TCM 实现，遵守 Apache 2 许可证进行分发。

你可以在以下位置获得源代码：https://github.com/sartrey-studio/vapula

具体来说，Vapula 框架的授权

允许你：

1. 自由的下载并使用部分或完整的 Vapula 框架，允许用于私人、公司内部或商业目的
2. 将 Vapula 放入你自己创建的安装包或分发中

禁止你：

1. 在没有合适的权力声明的情况下重新分发 Vapula 的任意部分
2. 以任何方式（声明或暗示 Sartrey Studio 已经为你的分发背书）使用 Sartrey Studio 拥有的任何商标
3. 以任何方式（声明或暗示你创建了此软件）使用任何 Sartrey Studio 拥有的商标

需要你：

1. 在你的包含了 Vapula 的分发中包含一份许可证文件
2. 对你所包含的 Vapula 进行准确的声明，其权利归属于 Sartrey Studio

不需要你：

1. 在你的分发中包含 Vapula 源码或你对其进行的任何修改
2. 向 Vapula 项目提交你对 Sartrey Studio 的修改

2. 组件

定义

在 TCM 技术范畴内，组件特指基于 TCM 的软件模块。

物理形式

组件的物理形式是一系列文件的集合。

包括：

组件主体×1（文件名必须为：*[组件标识].[规范扩展名]*）

组件描述文件×1（XML文件，文件名必须为：*[组件标识]*.tcm.xml）

另外，建议其他相关文件的命名按照以下规范：

其他相关文件×n（任意文件，文件名为：*[组件标识].[可选功能标识]*.tcm.*[扩展名]*）

组件标识只能包含字符集[A-Za-z0-9._]，不能为空。

当前应用规范不限定组件标识的唯一性。

组件主体与组件配置文件必须位于同一目录。

虽然组件文档不是组件的必须组成，

但是强烈建议在开发或移植组件时，同步编写相关组件文档。

逻辑构成

一个组件中可以包含多个逻辑功能（function），数量不能超过 2e30（1073741824）。

每个功能必须使用一个组件内唯一的正整数作为标识（从 1 开始）。

每个功能可以具有多个参数（parameter），数量不能超过 2e30。

每个参数必须使用一个功能内唯一的正整数作为标识（从 1 开始）。

TCM 调用组件功能时，严格遵循配置中的参数约定构造参数信封。

一个设计良好的组件应当保证：在独立于其他组件的情况下正常工作。

请根据需要决定组件中功能的划分粒度。

拆分复杂的功能可以提高复用的可能性，但是可能增加管理的复杂度。

开发的第一步

开发TCM组件需要引用必要的依赖。

使用C++开发需要引用头文件tcm_dev_lib.h。

使用.NET开发需要引用程序集tcm_xbridge.dll。

组件开发的依赖默认可能不会启用其他目的的开发接口（如任务、驱动）。

开发人员可以手工引入其他依赖获得更多的功能。

请注意，出于安全性、稳定性的考虑，

TCM在2.X及后续版本中引入了调用信任机制。

该机制会限制部分API的调用方式，隔离不同的开发目的。

TCM的所有功能都封装在“tcm”或“TCM”命名空间内。

强烈建议开发过程中考虑支持Unicode。

TCM默认字符串是Unicode字符串。

下文内容多以C++为默认开发语言，仅用于示意，具体语言参考请查阅API文档。

接口规范

由于 TCM 的非侵入式设计，组件只需实现一个规范要求的专用入口点。

这个入口点接受 3 个参数，分别是：功能标识、参数信封对象、上下文对象。

这个入口点中应当实现且仅实现组件的内部功能跳转。

强烈建议不要在入口点内直接实现具体功能。

入口点返回时应当使用 TCM 定义的返回码。

请注意，入口点的返回值所提供的返回码在 TCM 中仅起到建议的作用。

TCM定义的返回码可以描述功能的执行情况，不是功能本身的输出参数。

最终的功能返回码以上下文对象中的返回码的值为准。

当组件的入口**正常**返回时（不是返回“正常”），

执行器会根据最终情况将合适的返回码写入上下文对象。

以下是TCM为返回码定义的含义：

标识	返回值	含义
ERROR	0	未定义的错误
NORMAL	1	正常执行完成
CANCELBYMSG	2	用户取消，通知方式，组件主动
CANCELBYFORCE	3	用户取消，强制方式，执行器主动
NULLENTRY	4	没有目标功能
NULLTASK	5	没有执行任何功能
…	[6, INT32_MAX]	保留，用于TCM未来的扩展定义

1) C++组件

C++组件其入口点应当是一个 C 风格导出函数。

功能的代码风格不受限制，可以使用纯 C 或 C++进行组件的功能开发。

函数签名：*int Run (int function, Envelope* envelope, Context* context)*

完整的写法如下（仅供参考）：

```
#include "tcm_dev_lib.h"
using namespace tcm;

extern "C" __declspec(dllexport)
int Run(int function, Envelope* envelope, Context* context);
```

2) .NET 组件

.NET 组件其入口点应当是一个静态公有方法。

该方法必须位于由组件标识命名的命名空间中的公有类 Program 中。

函数签名：*ReturnCode Run (int function, Envelope envelope, Context context)*

完整的写法如下（仅供参考，假设组件标识为 Sample）：

```
using TCM.Runtime;

namespace Sample
{
    public class Program
    {
        public ReturnCode Run(int function, Envelope envelope, Context context)
        {
            //在这里实现入口的功能跳转逻辑
        }
    }
}
```

避免重复异步

TCM 规范建议每一次功能调用都是一次性的。

这意味着功能一旦返回将无法再处理自身订阅的异步通知。

而在 TCM 中，组件的调度已经妥善地使用了异步机制。

建议在组件的功能实现中，**避免**使用异步代码。

如果存在 IO 操作，一般建议使用同步 IO。

如果一定要在 IO 时，异步执行计算任务，可以将 IO 与计算分离成不同的功能。

或自行在组件中实现同步等待。

提示

多线程技术在许多情况下**几乎不能**提高计算过程本身的效率。

如果需要通过并行策略优化运算效率，可以专门设计针对特定硬件的并行计算方案。

考虑到TCM本身可能会引入并行优化机制，

因此建议开发人员仅对核心计算代码进行并行改造。

组件状态通信

组件使用上下文对象（Context）操作执行状态。

上下文对象是线程安全的。

通过上下文对象，组件的功能可以广播进度的更新，也可以监听平台发布的控制码。

控制码是TCM规范中的一种“君子协定”，

控制码的实质是来自调用方（应用）的控制消息，本身没有强制能力。

组件应当主动监听并响应控制码，以实现合理的受控行为。

注意，TCM提供具有强制能力的运行控制API，这些API大多具有时间参数。

如果超过应用指定的时间，组件没有做出正确的响应，TCM将执行强制控制。

强制措施容易导致功能执行期间表现异常。

所以，建议主动实现对控制码的响应。

上下文对象中定义了4个控制码，如下表，表中也给出了建议动作。

标识

标识	值	含义
NULL	0	没有控制请求，控制码默认值，建议组件保持当前状态
PAUSE	1	请求暂停，建议组件切换到睡眠状态，睡眠期间请监听恢复请求
RESUME	2	请求恢复，建议组件切换回运行状态，继续执行功能
CANCEL	3	请求取消，建议组件停止功能，执行清理并返回
RESTART	4	请求重启，建议组件停止功能，执行清理并重新启动功能

上下文对象面向组件开发提供一些常用方法。

签名	说明
`int GetCtrlCode()`	获取来自调用方的控制码
`float GetProgress()`	获取当前执行进度
`void SetProgress(float value)`	设置当前执行进度。value范围：[0,100]
`void ReplyCtrlCode()`	广播“已经响应控制请求/没有控制则循环状态机”

示例：

监听并响应取消请求

```
if(context->GetCtrlCode() == TCM_CTRL_CANCEL) {
    //停止任务
    //执行清理
    //执行终结代码
}
```

监听并响应暂停请求

```
if(context->GetCtrlCode() == TCM_CTRL_PAUSE) {
    context->ReplyCtrlCode();
    while(true) {
        Sleep(100);
        if(context->GetCtrlCode() == TCM_CTRL_RESUME) {
            context->ReplyCtrlCode();
            break;
        }
        //执行暂停期间其他操作
    }
}
```

设置和获取进度

```
int total = 10000;
for(int i=0; i<total; i++) {
    context->SetProgress(i / (float)total * 100.0f);
}
float prog = context->GetProgress(); //prog = 100.0f;
```

切换正常运行与UI运行状态

```
//从2.0.0.6开始引入UI状态，用于区分后台运行和UI运行两种运行期状态
//通过答复空控制码可以进行UI状态切换
//当前状态是BUSY，将切换到UI
context->ReplyCtrlCode();
//当前状态是UI，将切换到BUSY
context->ReplyCtrlCode();
```

组件参数通信

组件使用信封对象（Envelope）进行参数传递。

信封对象由TCM构造。

大多数情况下，调用方（应用）会通过TCM，生成符合功能的参数约定的参数信封。

功能的参数所需要的内存托管于TCM。

TCM支持全部数据类型，但是不支持保留结构信息的对象传递。

具体见下表（其中标识以 C++为例，仅供示意，具体请参考）。

标识	值	描述
POINTER	0	指针
INT8	1	8位整数
UINT8	5	无符号8位整数
INT16	2	16位整数
UINT16	6	无符号16位整数
INT32	3	32位整数
UINT32	7	无符号32位整数
INT64	4	64位整数
UINT64	8	无符号64位整数
REAL32	10	32位浮点数
REAL64	11	64位浮点数
BOOL	20	布尔值
STRING	21	字符串

关于用户对象和数组

用户的自定义对象和数组应当使用指针替代。

向信封对象写入这些数据时，**可以**同时提供该对象的实际大小（以字节为单位）。

当开发人员提供对象大小（非0）时，TCM将执行**浅拷贝**，在内存中生成对应对象的浅表副本，并记录此指针。这样，即便原有对象由于某些原因失效，写入信封中的对象依然有效。但是TCM并不关心原先的对象，所以一些开发失误可能导致内存泄露。

如果开发人员不提供对象的实际大小（0），TCM将仅写入引用。

这种情况下请自行保证该对象的生命周期和引用安全。

这时，如果外部在TCM依然占据该对象引用时释放了对象，则可能引发严重访问冲突。

关于转型读写支持

TCM提供有限的转型读写支持，可以在读写参数时 序列化到字符串 或 从字符串反序列化数据。

但是，组件开发API中默认屏蔽了这一功能。

这要求组件开发过程中使用明确类型的读写方法，避免使用转型读写。

信封对象面向组件开发人员提供以下常用方法。

签名	说明
`int GetParamTotal()`	获取参数数量
`bool GetInState(int id)`	获取参数方向
`T Read(int index)`	读取参数
`void Write(int index, T value)`	写入参数
`void Write(int index, LPVOID value, int size)`	写入对象（重载）
`void Write(int index, PCSTR value)`	写入多字节常字符串（重载）
`void Write(int index, PCWSTR value)`	写入宽字节常字符串（重载）

示例：

从信封中读出索引为 0 的 32位整型参数

```
int a = envelope->Read<int>(0);
```

向信封中写入索引为 5 的 32位浮点型参数

```
float a = 1.5f;
envelope->Write(5,a);
```

向信封中用两种方式写入一个对象，索引为 3 和 4

```
class TestA {
public:
    int a;
    float b;
    long c;
};

void Func(Envelope* envelope) {
    TestA* obj_a = new TestA();
    envelope->Write(3,obj_a, sizeof(obj_a)); //写入对象的浅表副本
    envelope->Write(4,obj_a, NULL); //仅写入对象的引用
}
```

3. 驱动

定义

在 TCM 技术范畴内，驱动特指为指定运行环境开发的 TCM 接口以及该运行环境下的 TCM 基础库。

物理形式

驱动的物理形式是一个匹配 Bridge 的语言环境的二进制文件。

[.tcm.driver]*（驱动，×1）

4. 任务

定义

在 TCM 技术范畴内，任务特指为一次完整独立的组件调用过程。

物理形式

任务的物理形式是一个描述功能调用全过程的 XML 文件。

任务的本质是一次完整的远程功能调用过程。

TCM 通过宿主执行任务，并提供一些基本调用不具备的高级特性。

包含完整预设参数、参数与通讯注入、指定目标

首先驱动加载，

然后，一个任务表示一个完整过程，包括：

任务清单解析，

模块加载，执行器构建，相关对象构建，

执行，依赖注入解决

跨线程通信仅用于宿主与外界程序通信

依赖注入可以通过多种方式

反馈

控制注入

任务由宿主支持

其他常用函数

TCM为开发人员提供了一系列工具函数。

签名	说明
`PCWSTR MbToWc(PCSTR src, UINT codepage)`	转换多字节字符串到宽字节字符串
`PCSTR WcToMb(PCWSTR src, UINT codepage)`	转换宽字节字符串到多字节字符串
`PCSTR ValueToStrA(T value)`	转换数值到字符串
`PCWSTR ValueToStrW(T value)`	转换数值到字符串（Unicode）
`T* VectorToArray(vector<T>& src)`	转换vector容器到定长（堆）数组
`int GetTypeUnit(int type)`	获得TCM类型的单位大小
`void Assert(bool state, LPVOID clear)`	根据断言结果执行清理
`PCSTR CopyStrA(PCSTR src)`	复制一个字符串
`PCWSTR CopyStrW(PCWSTR src)`	复制一个字符串（Unicode）
`PCSTR GetRandomStrA(int len)`	获得一个随机HEX字符串
`PCWSTR GetRandomStrW(int len)`	获得一个随机HEX字符串（Unicode）
`void ShowMsgbox(T value, PCWSTR caption)`	显示简易的信息框，以数值为内容
`void ShowMsgbox(PCSTR value, PCSTR caption)`	显示简易的信息框，自定义内容
`void ShowMsgbox(PCWSTR value, PCWSTR caption)`	显示简易的信息框，自定义内容（Unicode）
`PCWSTR GetAppDir()`	获取当前应用程序目录
`PCWSTR GetAppName()`	获取当前应用程序名称

运行时解耦

TCM提供一个宿主程序（tcm_host.exe）调用具体的TCM组件功能。

通过使用宿主程序调用TCM组件功能，您可以获得以下好处：

i) 调用方不依赖tcm_bridge.dll

ii) 调用方不会因为组件功能的崩溃而崩溃

iii) 调用方可以更方便的定制调度逻辑

通过使用宿主程序调用TCM组件功能，会导致以下问题：

i) 调用方与组件之间更难进行复杂对象通信

ii) 调用方与组件之间的通信效率根据具体的方案不同会有不同程度降低

使用宿主可以调用命令行：tcm_host <任务描述文件>

更丰富的命令行开关可以通过tcm_host本身查看。

任务描述文件是描述一次远程调用的全部信息的文件。

具体可以参考任务描述文件规范，也可以使用SDK的工具生成任务描述文件。

多语言支持

管理语言包

扩展数据结构

标志和字典

实用工具集

组件开发实用工具集随 TCM SDK 提供。

工具集启动器是组件开发实用工具集的快速启动目录。

Windows 环境的工具集启动器是 tcm_launch.exe

组件发布器可以配置组件的基本信息、图标组、功能参数表，并提供方便的发布功能。

Windows 环境的组件发布器是 tcm_pubish.exe。

组件测试器可以根据组件的配置进行自动化测试。

Windows 环境的组件测试器是 tcm_test.exe

组件任务助手可以通过简单操作快速生成调用任务并执行。

Windows 环境的组件任务助手是 tcm_task.exe

5. 模型

定义

在 TCM 技术范畴内，模型特指基于组件，通过流程图描述的一个完整、独立的业务。

物理形式

流程模型图文件以及依赖的组件构成一个TCM模型

模型通过模型设计器可以运行、调试

经过模型设计器发布，可以得到可执行模型包

可执行模型包可以脱离设计器在任何具备基础运行环境的计算机上运行

逻辑构成

模型通过标准流程图的特化版本进行描述。

图中应包含节点和关联。

节点表示具体的执行单位，具有多种类型。

关联标识节点间关联和依赖关系，对应图论的有向边。

节点说明

执行节点（Process）是基本的功能执行节点，是使用最频繁的节点，

其中需要指定该节点执行的具体组件的具体功能。

决策节点（Decision）是条件分支的起始，可以根据运行时情况动态选择执行路径。

决策节点是专家工具，需要用户了解具体的脚本编程。

变量节点（Variable）是提供数据的节点，本身没有任何实际功能，

变量节点是专家工具，需要用户自行设置需要的数据槽。

批量节点（Batch）是提供批量数据分发的节点，本身没有任何实际功能，

批量节点是专家工具，需要用户自行设置需要的数据槽以及数据分发方式。

可以执行的模型需要具备以下条件：

不存在无起点循环

所有节点参数完备，即初值或关联映射至少存在一项

节点的参数关联是一对一映射，不是多对一

节点的执行条件

具备有效执行体

参数必须有效

前向节点必须保证最长路径原则

启动节点条件

具备有效执行体

不具有优先权时不能有前向依赖

输入参数必须有值

设计原则

模型的基础描述语言是图，不是树。

所以原则上模型允许任意循环的存在。

为了能够被识别并执行，模型要求解决二义性循环描述。

存在二义性循环时，必须指定循环中某个节点为无视依赖启动。

目前允许执行、决策、代码

设置优先启动的节点将不检测其源节点（前向依赖）。

如果将循环中多个节点设置为源节点，将产生多重循环。

从一个节点执行到另一个节点必须保证最长路径能够执行完毕，短路不优先

执行策略

扫描所有节点，已经具备。。。

关联：

存在意义是

1. 可以使连接线与其他图元具有公平的数据模型绑定资格
2. 可以帮助序列化画布的连接线
3. 可以帮助实现参数的多对一输入（该条废弃）

不适用“最长路径”原则的原因：

前向可能存在循环

导致最长路径失效

两个特殊策略：

启动优先权：用于解决循环起点无法确定的问题

强依赖：用于解决节点由于多重路由导致的执行过载

强依赖仅能选择一个，某节点若要执行必须保证强依赖具有所有前向节点中的最大阶段数

执行过载：

多重执行路径都能经过某个节点（可能用户也会希望这个几乎是bug的特性）