**开发集成手册**

**简介**

引擎平台通过加载器对网络、日志、数据处理、分发策略、业务逻辑等模块进行统一封装, 并抽象了统一的链路交互协议及接口, 为AI引擎开发者提供了整套可便捷集成使用的AI引擎云服务化的基础设施及配套辅助系统。

引擎开发者通过实现平台加载器插件的方式,将AI能力以插件形式接入引擎平台从而对外提供分布式网络服务, 使内核引擎开发人员具备在线网络服务开发维护的能力. 用户仅需实现特定语言及平台的加载器插件接口,即可通过提供插件包的形式提交至引擎平台上线运行, 而不必关注在线服务所包含的网络传输、数据排序、分布式负载均衡、链路排障、数据收集等底层实现细节.

**适用对象**

\* 引擎内核开发人员

\* 引擎服务开发人员

\* 其他AI能力提供者

**快速集成**

用户通过实现特定语言的加载器插件接口，获取特定平台的加载器调试包及通用测试工具进行线下调试, 测试完成后即可提交产物至引擎平台进行上线审核发布.

**发布流程-集成调试**

1. 获取插件对应接口(c/c++,go,python)，及加载器对应平台调试包. [链接跳转](https://cloud.iflytek.com/aimarket/ai/index)

2. 实现对应插件接口并输出插件包(.so|.py)拷贝至平台调试包中.

3. 运行平台调试包中start.sh脚本启动加载器.

4. 按照测试调试说明文档，使用通用测试工具xtest进行测试. [链接跳转](https://cloud.iflytek.com/aimarket/ai/index)

5. 测试流程无异常则进行下一流程：审核发布.

**发布流程-审核发布**

1. 登录引擎平台上传引擎组件，资源模型，平台插件，测试报告等

2. 提交相关插件配置，标准输入输出及插件错误码描述等信息

3. 选择对应的加载器稳定版本基础镜像

4. 提交上线审核，审核通过后即可对外提供服务

**插件实现**

加载器通过事件注册及接口回调实现用户与云端计算引擎之间的数据交互, 通过对引擎交互流程及输入输出进行抽象定义出如下插件接口, 并将插件对应接口注册不同事件, 实现前端请求事件与加载器插件回调接口之间的映射. 实现平台前端、插件、计算引擎之间的数据交互.

**C/C++**

c/c++版本插件以c动态库方式实现. 插件接口依据不同类型的服务特性(会话/非会话,同步/异步等)提供了较为丰富的接口, 用户在集成过程中可按需实现服务对应的部分接口即可, 其他运行过程无需调用的接口也需进行空实现导出接口符号, 防止插件load过程失败. 如下接口按照*1.全局接口 2.会话接口 3.非会话接口* 进行分块说明.

1. *全局接口*

*/\*  
 wrapper插件初始化接口, 完成插件自身部分初始化工作, 进程启动加载插件后调用1次  
 @param cfg 服务层配置对, 由用户提交(离线调试通过aiges.toml [wrapper]配置)  
\*/  
int WrapperAPI wrapperInit(pConfig cfg);*

*/\*  
 wrapper服务层逆初始化, 进程结束退出时调用1次  
\*/  
int WrapperAPI wrapperFini();*

*/\*  
 获取插件错误码对应的描述信息   
 @param errNum 插件接口抛出的异常错误码  
 @return 插件错误码errNum对应的错误描述信息  
\*/*const char\* WrapperAPI wrapperError(int errNum);

*/\*  
 获取插件版本信息, 加载器通过该接口返回的版本信息请求配置中心获取配置  
 @return 插件版本信息  
\*/*const char\* WrapperAPI wrapperVersion();

*/\*  
 个性化数据加载接口, 加载器传入模型及模型映射id, 后续计算过程传入需要使用的模型id  
 @param perData 个性化模型数据实体  
 @param resId 个性化数据实体对应的id, 后续接口通过资源id标识对应的个性化数据  
 @return 接口错误码  
\*/  
int WrapperAPI wrapperLoadRes(pDataList perData, unsigned int resId);*

*/\*  
 个性化数据卸载接口  
 @param resId 个性化数据对应的映射id, 即wrapperLoadRes接口传入的resId  
 @return 接口错误码  
\*/  
int WrapperAPI wrapperUnloadRes(unsigned int resId);*

1. *会话接口*

*/\*  
 用于创建会话模式所需进行上下文关联的计算资源句柄   
 @param params 请求会话参数对, 即客户端调用上传的请求相关参数对, 传递至插件进行功能特性的控制  
 @param cb 加载器异步回调接口, 接口定义见头文件wrapper.h, 由加载器实现传入.*

*若配置同步模式则cb为null,输出数据通过wrapperRead接口传递*

*若配置异步模式则cb非null,输出数据通过回调接口cb返回至加载器;  
 @param psrIds 请求会话所需的个性化资源映射id, 即通过wrapperLoadRes接口加载的资源  
 @param psrCnt 会话个性化资源的数量, 即len(psrIds)  
 @param errNum 接口错误码[in/out]  
 @return 会话模式计算资源句柄,用于关联上下文;  
\*/  
const void\* WrapperAPI wrapperCreate(pParamList params, wrapperCallback cb, unsigned int psrIds[], int psrCnt, int\* errNum);*

*/\*  
 用于会话过程中写入计算数据至插件  
 @param handle 会话模式计算资源句柄,用于关联上下文;  
 @param reqData 写入插件的数据实体, 数据结构定义见头文件type.h  
 @return 接口错误码  
\*/  
int WrapperAPI wrapperWrite(const void\* handle, pDataList reqData);*

*/\*  
 读取计算结果, wrapperCreate接口cb为null场景, 加载器通过该接口获取输出数据  
 @param handle 会话模式计算资源句柄,用于关联上下文;  
 @param respData 同步读取输出数据实体  
 @return 接口错误码  
 @note respData内存由插件申请维护,在destroy阶段销毁  
\*/  
int WrapperAPI wrapperRead(const void\* handle, pDataList\* respData);*

*/\*  
 释放计算资源及会话过程临时数据  
 @param handle 会话模式计算资源句柄,用于关联上下文;  
\*/  
int WrapperAPI wrapperDestroy(const void\* handle);*

1. *非会话模式*

*/\*  
 非会话模式计算接口,对应oneShot请求,请求无上下文, 一次输入,一次输出  
 @param reqData 写入插件的数据实体  
 @param respData 插件输出的结果实体,内存由底层服务层申请维护,通过execFree()接口释放  
 @return 接口错误码  
 @note 同步操作接口  
\*/  
int WrapperAPI wrapperExec(pParamList params, pDataList reqData, pDataList\* respData);*

*/\*  
 非会话模式计算过程临时数据释放接口, 用于释放wrapperExec接口申请的respData及其他临时数据  
 @param respData 由同步接口wrapperExec获取的插件输出数据  
\*/  
int WrapperAPI wrapperExecFree(pDataList\* respData);*

*/\*  
 非会话模式计算异步接口,对应oneShot请求. 与wrapperExec接口类型, 差异在于数据通过wrapperCallback输出  
 @param handle 请求句柄,关联异步连接  
 @param reqData 写入插件的数据实体  
 @param callback 异步回调接口,用于异步输出计算结果. 框架实现,插件内部调用,定义见wrapper.h  
 @param timeout 异步超时时间,加载器等待插件异步输出的最大响应时间,ms;  
 @note 异步操作接口  
\*/  
int WrapperAPI wrapperExecAsync(const char\* handle, pParamList params, pDataList reqData, wrapperCallback callback, int timeout);*

**Golang**

Golang版本插件以go plugin方式实现. 插件plugIn按需实现如下部分方法, 注意方法名首字母需大写.相关结构体定义参见plugin.go. 如实现一个非会话模式不支持个性化的插件引擎, 仅需按需实现WrapperInit|WrapperFini|WrapperVersion|WrapperOnceExec 四个接口即可(其他接口亦需空实现).

*// plugin interface*type (

// 插件全局初始化, cfg及aiges.toml [wrapper]配置的相关配置对, 用于控制插件初始化行为  
 WrapperInit func(cfg map[string]string)(err error)

// 插件全局逆初始化, 释放插件资源  
 WrapperFini func()(err error)

// 插件版本信息, 加载器通过该版本信息请求配置中心获取相关配置文件  
 WrapperVersion func()(version string)

// 资源加载接口, 传递加载器管理的个性化资源及对应映射id至插件.

// 插件需实现引擎个性化加载并维护资源实体res与映射id resId的关系.

// 后续接口对个性化数据的交互均通过资源id resId标识(注：不支持个性化的引擎无需实现该接口)  
 WrapperLoadRes func(res wrapperData, resId int)(err error)

// 资源卸载接口, 资源id即为通过WrapperLoadRes加载的标记resId.  
 WrapperUnloadRes func(resId int)(err error)

// 会话模式计算资源句柄申请接口, 申请句柄hdl用于会话模式的上下文关联.

// 输入参数param为客户端上传的请求相关参数对,用于控制插件相关功能特性

// 输入参数prdIds为会话计算过程中需使用的个性化数据集合, 成员即WrapperLoadRes加载的resId

// 输入参数cb为异步模式下的计算结果消息回传接口.由加载器实现,插件调用输出结果使用(同步模式下传入nil)

// 输出参数hdl即为关联上下文的资源句柄, 若获取失败则通过err返回异常.  
 WrapperCreate func(param map[string]string, prsIds []int, cb CallBackPtr)(hdl interface{}, err error)

// 会话模式计算数据写入接口, hdl即计算资源句柄, req为会话过程写入数据实体  
 WrapperWrite func(hdl interface{}, req []wrapperData) (err error)

// 会话模式计算结果读取接口, hdl即计算资源句柄, resp为插件输出的计算结果.  
 WrapperRead func(hdl interface{})(resp []wrapperData, err error)

// 会话模式计算调试信息输出接口,用于加载器获取插件在计算过程中的关键调试信息.

WrapperDebugInfo func(hdl interface{})(debug string)

// 会话模式句柄释放接口, 用于会话生命周期结束后释放句柄关联的上下文,与WrapperCreate配套使用

WrapperDestroy func(hdl interface{}) (err error)

// 非会话模式计算接口, 用于无上下文会话的单次oneShot交互, 服务请求流程仅包含一次输入及一次输出.

// 输入参数param为客户端上传的请求相关参数对,用于控制插件相关功能特性

// 输入参数req为非会话模式计算过程写入数据实体.

// 输出参数resp为插件输出的计算结果,若计算异常则err非nil

WrapperExec func(param map[string]string, req []wrapperData)(resp []wrapperData, err error)  
 )

**Python**

Python版本插件以py模块方式实现. 模块名为”wrapper”, 集成方需提供wrapper.py，并按需实现如下部分定义接口.

*class EngineBase: # clase EngineBase(object): c call exception*

*# 类属性-版本号,子类需实现,上层通过版本号获取服务配置  
 version = '1.0.1'  
  
 def wrapperInit(self, \*\*kwargs):  
 '''  
 @初始化接口,上层逻辑限制,替代\_\_init\_\_接口  
 输入：配置对kwargs(dict)  
 输出：return int, 返回错误码  
 '''  
 pass  
  
 def wrapperFini(self):  
 '''  
 @逆初始化接口,替代\_\_del\_\_接口  
 输出：return int, 返回错误码  
 '''  
 pass  
  
 def wrapperCreate(self, \*args, \*\*kwargs):  
 '''  
 @实例创建接口  
 输入：个性化资源id列表args, 会话参数对kwargs  
 输出：成功则返回实例object inst 用于关联会话相关接口, 失败None  
 '''  
 pass  
  
 def wrapperDestroy(self, inst):  
 '''  
 @实例销毁接口  
 输入：由create接口申请所得实例object inst  
 输出：return int, 返回错误码  
 '''  
 pass  
  
 def wrapperError(self, code):  
 '''  
 @错误码描述转换接口  
 输入：error code(int)  
 输出：return str, 返回错误描述信息  
 '''  
 pass  
  
 def wrapperDebugInfo(self, inst):  
 '''  
 @调试日志接口  
 输入：由create接口申请所得实例object inst  
 输出：return str, 返回会话计算过程中的debug信息  
 '''  
 pass  
  
 def wrapperWrite(self, inst, dataInput):  
 '''  
 @数据写接口,由上层框架调用写数据  
 输入：引擎实例object inst, 用户请求数据[]  
 dataInput = [{  
 "Key":"",  
 "Data":"",  
 "Type":"",  
 "Status":"",  
 "Desc":"",  
 "Encoding":"",  
 }]  
 输出：return int, 返回错误码  
 '''  
 pass  
  
 def wrapperRead(self, inst):  
 '''  
 @数据读接口,上层框架调用读取引擎结果  
 输入：引擎实例object inst  
 输出：return [], int ;返回引擎计算结果dataOutput,错误信息err; 即tuple  
 dataOutput = [{  
 "Key":"",  
 "Data":"",  
 "Type":"",  
 "Status":"",  
 "Desc":"",  
 "Encoding":"",  
 }]  
 '''  
 pass  
  
 def wrapperExec(self, \*\*kwargs, msg):  
 '''  
 @非会话oneShot接口  
 输入：用户请求参数对kwargs, 用户请求数据msg []  
 dataInput = [{  
 "Key":"",  
 "Data":"",  
 "Type":"",  
 "Status":"",  
 "Desc":"",  
 "Encoding":"",  
 }]  
 输出：return int, [] ; 返回错误信息err, 引擎计算结果dataOutput; 即tuple  
 '''  
 pass  
  
 def wrapperLoadRes(self, id, res):  
 '''  
 @资源加载接口  
 输入：资源数据res, 资源id(int)  
 res = {  
 "Key":"",  
 "Data":"",  
 "Type":"",  
 "Status":"",  
 "Desc":"",  
 "Encoding":"",  
 }  
 输出：return int, 返回错误码  
 '''  
 pass  
  
 def wrapperUnloadRes(self, id):  
 '''  
 @资源卸载接口  
 输入：待卸载资源id(int)  
 输出：return int, 返回错误码  
 '''  
 pass  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 engine = EngineBase()  
 print(engine.version)*

**错误码说明**

**插件错误码**

插件返回错误码建议采用长度为6位的十进制uint值，与平台错误码以示区分.插件错误码及对应的异常描述信息将由集成方于平台上线流程进行提交审核.

**加载器错误码**

*AigesSuccess = 0*

*// 服务实例错误码*

*AigesErrorInvalidOp = 10003*

*AigesErrorInvalidSessMode = 10004*

*AigesErrorInvalidPara = 10007*

*AigesErrorInvalidHdl = 10008*

*AigesErrorInvalidData = 10009*

*AigesErrorLicNotEnough = 10010*

*AigesErrorDpInit = 10011*

*AigesErrorSessTimeout = 10019*

*AigesErrorEngInactive = 10101 // TODO 兼容保留*

*AigesErrorOnceTimeout = 10102*

*// 排序缓冲区错误*

*AigesErrorBufferEmpty = 10300*

*AigesErrorSeqChanClosed = 10301*

*// 协议错误*

*AigesErrorPbMarshal = 10400*

*AigesErrorPbUnmarshal = 10401*

*// 内部同步错误*

*AigesErrorFinRoutine = 10500*

*// 事件异常错误*

*AigesErrorNilEvent = 10600*

*// Auth计量授权*

*AigesErrorNotAuthority = 10700*

*// 个性化模块错误*

*AigesErrorNilResTags = 10800*

*AigesErrorResChanFull = 10801*

*// 插件接口异常*

*WrapperInitErr = 100001*

*WrapperFiniErr = 100002*

*WrapperLoadErr = 100003*

*WrapperUnloadErr = 100004*

*WrapperCreateErr = 100005*

*WrapperDestroyErr = 100006*

*WrapperWriteErr = 100007*

*WrapperReadErr = 100008*

*WrapperExecErr = 100009*

**集成样例**

[wrapper-iat-2s集成示例](https://git.iflytek.com/zsli/wrapper-iat-2s)

**调试测试**

参考调试测试文档，包含：

1. 测试环境说明
2. 启动参数说明
3. 加载器配置说明
4. 测试工具使用说明
5. 加载器参数说明

**技术支持**

邮箱: zsli@iflytek.com

微信: 1126544836

电话: 15656957584