**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++,python)，及加载器对应平台调试包.

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.

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

**插件开发**

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, 3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.

插件加载方式：

|  |  |
| --- | --- |
| 集成类型 | 加载方式 |
| c/c++ | c接口，调用dlopen/dlsym/dlclose相关接口集 |
| python | c接口，调用PyImport\_ImportModule/PyModule\_GetDict相关接口集 |

*注1：python版本要求2.7*

**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集, 用户在集成过程中可按需实现服务对应的部分接口即可,

*注：运行过程中无需调用的其他接口也需空实现导出接口符号, 防止插件load过程失败.*

**接口说明**

c/c++插件共提供13个接口，其中包含插件的初始化/资源操作/会话操作/非会话操作等相关接口，对于大部分服务类型，仅需实现其中部分接口即可. 如下对相关接口进行说明

1. *wrapperInit*

|  |  |
| --- | --- |
| 原型 | int WrapperAPI wrapperInit(pConfig cfg) |
| 参数 | 初始化配置对，由aiges.toml配置，其中k-v类型均为string |
| 返回值 | 插件初始化异常错误码，错误码用于快速定位异常原因，类型int  接口调用正常需返回0值. |
| 说明 | 全局调用一次，用于插件初始化及内核实例预申请等操作 |

*类型定义：pConfig*

*typedef struct ParamList{*

*char\* key;*

*char\* value;*

*unsigned int vlen;*

*struct ParamList\* next;*

*}\* pConfig;*

*配置示例：aiges.toml(若无该配置段,可新增)*

*[wrapper]*

*“common.lic” = “10”*

*“resource./resource/res.bin” = “WFST”*

*接口示例：*

*int maxLic = 10; // default concurrency*

*std::map<kernelInst, bool/\*idle\*/> hdlCache;*

*Int WrapperAPI wrapperInit(pConfig cfg){*

*while(cfg){*

*printf(“key:%s, value:%s\n”, cfg->key, cfg->value);*

*If (strcmp(cfg->key, “common.lic”) == 0){*

*maxLic = atoi(cfg->value);*

*}*

*cfg = cfg->next;*

*}*

*// 插件实例管理-预申请*

*for (int i=0; i< maxLic ){*

*kernelInst inst = new kernelInst;*

*hdlCache[inst] = true;*

*}*

*return 0;*

*}*

*示例输出：*

*> key:common.lic, value:10*

*> key:resource./home/res.bin, value:WFST*

1. *wrapperFini*

|  |  |
| --- | --- |
| 原型 | int WrapperAPI wrapperFini() |
| 参数 | 无 |
| 返回值 | 插件逆初始化异常错误码，错误码用于快速定位异常原因，类型int  接口调用正常需返回0值. |
| 说明 | 1. 全局调用一次，用于插件逆初始化及内核实例释放等操作 |

*接口示例：*

*int WrapperAPI wrapperFini(){*

*std::map<kernelInst, bool>::iterator ite = hdlCache.begin();*

*for ( ; ite != hdlCache.end() ; ite++){*

*delete ite->first;*

*}*

*return 0;*

*}*

1. *wrapperError*

|  |  |
| --- | --- |
| 原型 | const char\* WrapperAPI wrapperError(int errNum); |
| 参数 | 插件自定义的错误码，类型int |
| 返回值 | 插件自定义错误码对应的描述信息，类型const char\*，非NULL |
| 说明 | 1. 用于插件接口调用异常报错后，加载器通过调用该接口获取错误码描述. 2. 插件接口异常报错时调用 |

*接口示例：*

*static const struct errPair {*

*int err;*

*const char\* desc;*

*} errPairArr[] = {*

*{10010, “instance not enough”}，*

*{10011, “input invalid data”},*

*};*

*const char\* WrapperAPI wrapperError(int errNum){*

*int errSize = sizeof(errPairArr) / sizeof(errPair);*

*for (int i=0; i < errSize; i++){*

*If (errPairArr[i].err == errNum){*

*return errPairArr[i].desc;*

*}*

*}*

*return “general error”;*

*}*

*调用示例：*

*printf(“err: %s\n”, wrapperError(10010));*

*printf(“err: %s\n”, wrapperError(10011));*

*示例输出：*

*> err: instance not enough*

*> err: input invalid data*

1. *wrapperVersion*

|  |  |
| --- | --- |
| 原型 | const char\* WrapperAPI wrapperVersion(); |
| 参数 | 无 |
| 返回值 | 插件版本号，类型const char\*，非NULL |
| 说明 | 1. 版本号用于云服务化定位不同版本配置文件所用，标记不同服务版本 2. 全局启动过程中调用一次 |

*接口示例：*

*const char\* version = “1.0.1”;*

*const char\* WrapperAPI wrapperVersion(){*

*return version;*

*}*

1. *wrapperLoadRes*

|  |  |
| --- | --- |
| 原型 | int WrapperAPI wrapperLoadRes(pDataList perData, unsigned int resId); |
| 参数 | 1. 个性化数据实体perData. 2. 实体对应唯一标识resId |
| 返回值 | 个性化加载异常错误码，错误码用于快速定位异常原因，类型int  接口调用正常需返回0值. |
| 说明 | 加载器通过调用该接口加载用户上传的个性化资源，并分配一个唯一标识resId. 后续对该资源的所有操作均通过resId标识，故插件需维护资源-id的映射关系. |
| 备注 | 1. 若插件不支持个性化、可空实现直接返回0. 2. 存在个性化上传/更新/请求时调用. |

*类型定义：*

*typedef struct DataList{*

*char\* key; // 数据标识*

*void\* data; // 数据实体*

*unsigned int len; // 数据长度*

*char\* desc; // 数据描述*

*char\* encoding; // 数据编码*

*DataType type; // 数据类型*

*DataStatus status; // 数据状态*

*struct DataList\* next; // 链表指针*

*}\* pDataList;*

*补充说明：*

*(1). 资源加载接口实参perData链表长度固定为1.*

*(2). 资源加载场景，仅使用部分成员标记数据：perData->data，perData->len，perData->type*

*其他成员为无效数据，无需读取.*

1. *wrapperUnloadRes*

|  |  |
| --- | --- |
| 原型 | int WrapperAPI wrapperUnloadRes(unsigned int resId); |
| 参数 | 个接口说明  性化数据实体对应的唯一标识resId |
| 返回值 | 个性化卸载异常错误码，错误码用于快速定位异常原因，类型int  接口调用正常需返回0值. |
| 说明 | 卸载通过wrapperLoadRes接口加载的个性化数据 |
| 备注 | 1. 若插件不支持个性化、可空实现直接返回0. 2. 个性化缓存超时、个性化请求删除时调用. |

1. *wrapperCreate*

|  |  |
| --- | --- |
| 原型 | const void\* WrapperAPI wrapperCreate(pParamList params, wrapperCallback cb, unsigned int psrIds[], int psrCnt, int\* errNum); |
| 参数 | 1. 用户请求上传的参数对params，用于会话中控制插件功能 2. 用于异步请求的加载器回调接口cb，同步请求时为NULL 3. 会话依赖的个性化资源组psrIds[]，成员即*wrapperLoadRes传入id.* 4. 会话依赖的个性化资源组大小psrCnt，即psrIds数组大小. 5. 会话实例申请异常错误码errNum，类型int\*，接口调用正常需返回0值. |
| 返回值 | 会话实例申请的插件句柄，关联插件内核实例及后续会话相关插件接口.  接口调用异常需返回NULL. |
| 说明 | 加载器通过该接口申请插件内核实例句柄，用于关联会话上下文. |
| 备注 | 1. 对于会话模式请求，每次会话对应一次调用. 非会话模式不调用该接口. 2. 与wrapperDestroy接口成对使用. 3. 该接口存在外部并发调用可能性. |

*类型定义：*

*typedef struct ParamList{*

*char\* key;*

*char\* value;*

*unsigned int vlen;*

*struct ParamList\* next;*

*}\* pParamList;*

*// 回调接口，用于异步模式回传内核响应结果*

*// handle：wrapperCreate接口返回的插件实例句柄.*

*// respData：插件输出的会话请求计算结果.*

*typedef int(\*wrapperCallback)(const void\* handle, pDataList respData);*

*接口示例：*

*(1). 同步模式：外部加载器通过wrapperRead接口获取结果*

*const void\* WrapperAPI wrapperCreate(pParamList params, wrapperCallback cb, unsigned int psrIds[], int psrCnt, int\* errNum){*

*// hdlCache lock*

*std::map<kernelInst, bool>::iterator ite = hdlCache.begin();*

*for ( ; ite != hdlCache.end() ; ite++){*

*If (ite->second){*

*ite->second = false; \*errNum = 0;*

*// hdlCache unlock*

*return (void\*)ite->first;*

*}*

*}*

*// hdlCache unlock*

*\*errNum = 10010;*

*return NULL;*

*}*

*(2). 异步模式：插件通过调用加载器回调接口cb返回结果.*

*// func1*

*wrapperCallback funcCB;*

*const void\* WrapperAPI wrapperCreate(pParamList params, wrapperCallback cb, unsigned int psrIds[], int psrCnt, int\* errNum){*

*// hdlCache lock*

*std::map<kernelInst, bool>::iterator ite = hdlCache.begin();*

*for ( ; ite != hdlCache.end() ; ite++){*

*If (ite->second){*

*ite->second = false; \*errNum = 0;*

*// hdlCache unlock*

*funcCB = cb;*

*return (void\*)ite->first;*

*}*

*}*

*// hdlCache unlock*

*\*errNum = 10010;*

*return NULL;*

*}*

*// func2, internal private function*

*Int kernelCalc(kernelInst inst){*

*pDataList result = inst->calc(inputData);*

*funcCB(inst, result);*

*return 0;*

*}*

*补充说明：*

*(1). 若会话为同步模式，插件需实现create|write|read|destroy配套接口.(当前仅支持同步模式)*

*(2). 若会话为异步模式，插件需实现create|write|destroy接口，结果回传通过回调cb进行返回同步.*

*(3). 会话模式对应配置文件及控制项：aiges.toml*

*[aiges]*

*sessMode = 1 // 1:会话模式、0:非会话模式*

1. *wrapperWrite*

|  |  |
| --- | --- |
| 原型 | int WrapperAPI wrapperWrite(const void\* handle, pDataList reqData); |
| 参数 | 1. 会话实例申请所得插件句柄handle，wrapperCreate获取. 2. 会话请求上传的实体数据reqData，包含文本/音频/图像/视频等数据类型. |
| 返回值 | 写数据接口异常错误码，错误码用于快速定位异常原因，类型int  接口调用正常需返回0值. |
| 说明 | 会话模式写数据接口，用于将用户请求上传的会话数据传递至插件内核 |
| 备注 | 每次会话对应多次调用，非会话模式不调用 |

*接口示例：*

*int WrapperAPI wrapperWrite(const void\* handle, pDataList reqData){*

*kernelInst inst = (kernelInst)handle;*

*// 插件写入计算数据*

*int ret = Inst->dataWrite(reqData);*

*return ret;*

*}*

*补充说明：*

*(1). 用户请求上传的分块数据可能在加载器中进行合并.*

*(2). 对于单个具体插件实例，加载器单线程调用写数据接口，保证写入数据有序.*

1. *wrapperRead*

|  |  |
| --- | --- |
| 原型 | int WrapperAPI wrapperRead(const void\* handle, pDataList\* respData); |
| 参数 | 1. 会话实例申请所得插件句柄handle，wrapperCreate获取 2. 插件计算返回的结果respData |
| 返回值 | 读数据接口异常错误码，错误码用于快速定位异常原因，类型int  接口调用正常需返回0值. |
| 说明 | 会话模式读数据接口，用于获取插件内核计算结果，最终返回请求用户. |
| 备注 | 每次会话对应多次调用，非会话模式不调用 |

*类型定义：*

*typedef struct DataList{  
 char\* key; // 数据标识  
 void\* data; // 数据实体  
 unsigned int len; // 数据长度  
 char\* desc; // 数据描述  
 char\* encoding; // 数据编码  
 DataType type; // 数据类型  
 DataStatus status; // 数据状态  
 struct DataList\* next; // 链表指针  
}\* pDataList;*

*接口示例：*

*int WrapperAPI wrapperRead(const void\* handle, pDataList\* respData){*

*kernelInst inst = (kernelInst)handle;*

*// 插件查询计算结果*

*int ret = Inst->queryResult(respData);*

*return ret;*

*}*

*调用示例：*

*(1). 实时结果模式：会话过程中每次写数据后调用读接口查询实时结果*

*int aigesCall(){*

*void\* inst = wrapperCreate();*

*while(!input.end()){*

*wrapperWrite(inst, input);*

*wrapperRead(inst, output);*

*}*

*while(!output.end()){*

*wrapperRead(inst, output);*

*}*

*return 0;*

*}*

*(2). 非实时结果模式：会话过程中所有数据完整写入后调用读接口*

*int aigesCall(){*

*void\* inst = wrapperCreate();*

*while(!input.end()){*

*wrapperWrite(inst, input);*

*}*

*while(!output.end()){*

*wrapperRead(inst, output);*

*}*

*return 0;*

*}*

*补充说明：*

*(1). 对于单个具体插件实例，加载器单线程调用读数据接口*

*(2). 结果respData内存空间由插件申请及维护，可于destroy时释放.*

*(3). 会话模式返回结果状态respData->status取值DataBegin/DataContinue/DataEnd*

*(4). 实时结果及非实时结果模式依靠配置文件进行控制：aiges.toml*

*[aiges]*

*realTimeRlt = 1 // 1:实时模式、2:非实时模式*

1. *wrapperDestroy*

|  |  |
| --- | --- |
| 原型 | int WrapperAPI wrapperDestroy(const void\* handle); |
| 参数 | 会话实例申请所得插件句柄handle，wrapperCreate获取 |
| 返回值 | 插件实例句柄释放接口异常错误码，类型int  接口调用正常需返回0值. |
| 说明 | 1. 该接口与wrapperCreate成对使用. 用于句柄申请及释放. 2. 每次会话对应一次调用. |

*接口示例：*

*int WrapperAPI wrapperDestroy(const void\* handle){*

*// hdlCache lock;*

*std::map<kernelInst, bool>::iterator ite = hdlCache.find(handle);*

*If (ite != hdlCache.end()){*

*ite->second = true;*

*}*

*// hdlCache unlock;*

*return 0;*

*}*

1. *wrapperExec*

|  |  |
| --- | --- |
| 原型 | int WrapperAPI wrapperExec(pParamList params, pDataList reqData, pDataList\* respData); |
| 参数 | 1. 用户请求上传的参数对params，用于会话中控制插件功能 2. 用户请求上传的实体数据reqData，包含文本/音频/图像/视频等数据类型 3. 插件计算返回的结果respData |
| 返回值 | 插件非会话请求接口异常错误码，类型int  接口调用正常需返回0值. |
| 说明 | 1. 接口用于处理非会话模式请求(无上下文管理)，每次用户请求触发一次调用. 2. 该接口与wrapperExecFree接口成对使用. |

*类型定义：*

*typedef struct ParamList{*

*char\* key;*

*char\* value;*

*unsigned int vlen;*

*struct ParamList\* next;*

*}\* pParamList;*

*接口示例：*

*int WrapperAPI wrapperExec(pParamList params, pDataList reqData, pDataList\* respData){*

*return kernel->calc(params, reqData, respData);*

*}*

*补充说明：*

*(1). 结果respData内存空间由插件申请及维护，于wrapperExecFree时释放*

*(2). 非会话模式输出结果状态respData->status取值仅为DataOnce*

*(3). 非会话模式对应配置文件配置项，aiges.toml*

*[aiges]*

*sessMode = 0 // 1:会话模式、0:非会话模式*

1. *wrapperExecFree*

|  |  |
| --- | --- |
| 原型 | int WrapperAPI wrapperExecFree(pDataList\* respData); |
| 参数 | wrapperExec接口返回的非会话请求结果缓存指针. |
| 返回值 | 插件非会话结果缓存释放接口异常错误码，类型int  接口调用正常需返回0值. |
| 说明 | 1. 接口用于释放非会话接口的结果缓存，每次用户请求触发一次调用 2. 该接口与wrapperExec接口成对使用. |

*接口示例：*

*int WrapperAPI wrapperExecFree(pDataList\* respData){*

*return kernel->release(respData);*

*}*

1. *wrapperExecAsync*

|  |  |
| --- | --- |
| 原型 | int WrapperAPI wrapperExecAsync(const char\* handle, pParamList params, pDataList reqData, wrapperCallback callback, int timeout); |
| 参数 | 1. 加载器请求句柄，用于异步回调callback时回传以关联请求及结果. 2. 用户请求上传的参数对params，用于会话中控制插件功能 3. 用户请求上传的实体数据reqData，包含文本/音频/图像/视频等数据类型 4. 请求异步回调，用于回传请求数据计算结果至加载器. 5. 异步请求结果超时时间timeout，单位ms. |
| 返回值 | 插件非会话异步接口异常错误码，类型int  接口调用正常需返回0值. |
| 说明 | 该接口需实现为异步非阻塞模式. 请求结果通过加载器回调接口callback回传. |

*类型定义：*

*typedef int(\*wrapperCallback)(const void\* handle, pDataList respData);*

*接口示例：*

*// func1*

*int WrapperAPI wrapperExecAsync(const char\* handle, pParamList params, pDataList reqData, wrapperCallback callback, int timeout){*

*return = kernel->produce(handle, params, reqData, callback, timeout);*

*}*

*// func2, internal private function*

*Int kernelAsyncCalc(){*

*handle, params, reqData, callback, timeout = kernel->consume();*

*respData = kernel->calc(params, reqData, timeout);*

*return callback(handle, respData);*

*}*

**集成说明**

对应不同插件引擎的服务模式，插件仅需实现上述部分接口即可正常提供服务，其他接口可做空实现处理. 如下以不同服务模式进行说明.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 服务模式 | | | 对应接口集 | 对应配置项 | 备注 |
| 会话模式 | 同步模式 | 实时结果 | 1. wrapperCreate 2. wrapperWrite 3. wrapperRead 4. wrapperDestroy | [aiges]  sessMode = 1  realTimeRlt = 1 |  |
| 非实时结果 | [aiges]  sessMode = 1  realTimeRlt = 0 |  |
| 异步模式 | | 1. wrapperCreate 2. wrapperWrite 3. wrapperDestroy | [aiges]  sessMode = 1 | 暂未稳定 |
| 非会话模式 | 同步模式 | | 1. wrapperExec 2. wrapperExecFree | [aiges]  sessMode = 0 |  |
| 异步模式 | | 1. wrapperExecAysnc | [aiges]  sessMode = 0 | 暂未稳定 |

*注：*

1. *通用接口wrapperInit/wrapperFini/wrapperVersion/wrapperError，各类服务模式插件均需实现.*
2. *个性化接口wrapperLoadRes/wrapperUnloadRes由各类服务模式插件按需实现.*

**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