HUAWEI AOS 3.2

AUTOSAR AP 接口参考

文档版本	06
发布日期	2022-05-30

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

前言

概述

本文档详细的描述了Adaptive AUTOSAR组件支持的外部接口以及其描述，作为软件开发人员二次开发的手册。

读者对象

本文档适用通用软件开发人员阅读。开发人员必须具备以下经验和技能：

- 熟悉面向对象的C++编程语言。
- 熟悉车载Adaptive AUTOSAR软件标准。
- 了解面向服务的软件架构基本概念。

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
06	2022-05-30	第六次正式发布，文档内容更新如下： 新增功能描述： • 新增IAM接口说明：IAM。 勘误和内容优化： • CM章节2.2 Skeleton的Event通信方式中的Send接口，刷新使用说明：如果使用DDS协议，需为DDS协议头预留1KB空间。 • Tysnc章节中SynchSlaveTB接口刷新函数功能描述。
05	2022-03-30	第五次正式发布，文档内容更新如下： 新增功能描述： • EM新增operator相关接口说明：4.2 SetState。 • 新增Tsync功能涉及的接口说明：9 Tsync。 勘误和内容优化： • 7 PER中7.1 OpenKeyValueStorage~7.3 GetCurrentKeyValueStorageSize入参kvs增加const修饰。 • 8 DM中更正8.2.2 ReentrancyType所在头文件、8.4.12.3 DiagnosticManagerConfig::GetPerSpecifier函数原型中补充括号。

文档版本	发布日期	修改说明
04	2022-01-30	第四次正式发布，文档内容更新如下： 新增功能描述： • CoreTpye新增Variant类型：**3.18 Variant**。 • 新增PER接口说明：**7 PER**。 • 新增DM接口说明：**8 DM**。 • EM新增GetState接口：**4.10 GetState**。 勘误和内容优化： • **5 LOG**中刷新所有接口的函数原型表达式、接口名称UnregisterBackends变更为Unregister、Initialize/Deinitialize/InitLogging接口注意事项中刷新可重入为不可重入。 • **3 CoreType**新增Initialize and Shutdown接口说明。
03	2021-11-30	第三次正式发布，文档内容更新如下： 新增功能描述： • 无 勘误和内容优化： • **3.5 CoreErrorDomain**优化目录结构，删除父目录CoreErrorDomain下的CoreErrorDomain子目录（目录重名），将内容移动到父目录CoreErrorDomain下，无接口内容变更。 • **5 LOG**章节优化接口功能描述及使用说明等。 • **5.2.22 remoteClientState**修改接口名称RemoteClientState为remoteClientState。 • **6.12 RegisterRecoveryCallback**优化参数IN的说明。 • 刷新本文档02版本修订记录，补充**•5.3-LogStream Class中修改接...**。
02	2021-10-30	第二次正式发布，文档内容更新如下： • **6.27 MakeErrorCode**增加使用说明。 • **5.3 LogStream Class**中修改接口名称operator拼写错误。
01	2021-08-30	第一次正式发布。

目 录

1 注意事项

本文的所有API仅能在main()中调用，尤其注意禁止使用全局变量或静态变量实例化AP接口类，以防在main()结束后，全局或静态变量析构时调用AP接口。

2 CM

2.1 数据类型

2.1.1 ConstructionToken

接口类定义

类名	ara::com::ConstructionToken
命名空间	namespace ara::com
语法	class ConstructionToken{...}
头文件	生成文件NAMESPACE/XXX_skeleton.h / NAMESPACE/XXX_proxy.h
说明	创建Skeleton与Proxy的构造令牌，预构建功能使用

成员函数定义

函数原型	ConstructionToken(const ConstructionToken& other) = delete;	
函数功能	ConstructionToken类拷贝构造函数	
参数（IN）	other	其他已存在的ConstructionToken实例
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	None	-
使用说明	ConstructionToken类禁用拷贝构造函数	
注意事项	无	

函数原型	ConstructionToken& operator=(const ConstructionToken& other) = delete;	
函数功能	ConstructionToken类拷贝赋值运算符	
参数（IN）	other	其他已存在的ConstructionToken实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ConstructionToken&	ConstructionToken实例
使用说明	ConstructionToken类禁用拷贝赋值运算符	
注意事项	无	

函数原型	ConstructionToken(ConstructionToken&& other);	
函数功能	ConstructionToken类移动构造函数	
参数（IN）	other	其他已存在的ConstructionToken实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	ConstructionToken& operator=(ConstructionToken&& other);
函数功能	ConstructionToken类移动赋值运算符

参数（IN）	other	其他已存在的ConstructionToken实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ConstructionToken&	ConstructionToken实例
使用说明	无	
注意事项	无	

2.1.2 InstanceIdentifier

接口类定义

类名	ara::com::InstanceIdentifier
命名空间	namespace ara::com
语法	class InstanceIdentifier {...}
头文件	#include "ara/com/types.h"
说明	服务实例的标识符

成员函数定义

函数原型	explicit InstanceIdentifier(ara::core::StringView value);	
函数功能	InstanceIdentifier类构造函数	
参数（IN）	value	Instance Id
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	ara::core::StringView ToString() const;
函数功能	以StringView的形式返回InstanceIdentifier包含的Instance Id信息

参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::StringView	Instance Id
使用说明	无	
注意事项	• 线程安全 • 由于是以StringView的形式返回，使用时应考虑InstanceIdentifier的生命周期	

函数原型	bool operator==(const InstanceIdentifier& other) const;	
函数功能	InstanceIdentifier类比较运算符	
参数（IN）	other	其他InstanceIdentifier实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	true代表是相同的InstanceIdentifier，false代表两个InstanceIdentifier存在差异
使用说明	无	
注意事项	无	

函数原型	bool operator<(const InstanceIdentifier& other) const;	
函数功能	InstanceIdentifier类比较运算符, 按字典序进行大小比较，返回比较结果	
参数（IN）	other	其他InstanceIdentifier实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	true代表当前实例小于输入的InstanceIdentifier实例，false代表当前实例大于等于输入的InstanceIdentifier实例
使用说明	无	

注意事项	无

函数原型	InstanceIdentifier& operator=(const InstanceIdentifier& other);	
函数功能	InstanceIdentifier类拷贝赋值运算符	
参数（IN）	other	其他InstanceIdentifier实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	InstanceIdentifier&	InstanceIdentifier实例
使用说明	无	
注意事项	无	

2.1.3 InstanceIdentifierContainer

接口类定义

类名	ara::com::InstanceIdentifierContainer
命名空间	namespace ara::com
语法	using InstanceIdentifierContainer = ara::core::Vector<InstanceIdentifier>;
头文件	#include "ara/com/types.h"
说明	包含InstanceIdentifier的列表

2.1.4 HandleType

接口类定义

类名	ara::com::HandleType
命名空间	namespace ara::com
语法	class HandleType {...}
头文件	#include "ara/com/types.h"
说明	包含创建ServiceProxy所需的特定服务实例的信息

成员函数定义

函数原型	HandleType(HandleType&& other);	
函数功能	HandleType类移动构造函数	
参数（IN）	other	其他HandleType实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	HandleType(const HandleType& other);	
函数功能	HandleType类拷贝构造函数	
参数（IN）	other	其他HandleType实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	HandleType& operator=(HandleType&& other);	
函数功能	HandleType类移动赋值函数	
参数（IN）	other	其他HandleType实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	HandleType&	HandleType实例
使用说明	无	
注意事项	无	

函数原型	HandleType& operator=(const HandleType& other);	
函数功能	HandleType类拷贝赋值函数	
参数（IN）	other	其他HandleType实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	HandleType&	HandleType实例
使用说明	无	
注意事项	无	

函数原型	bool operator==(const HandleType& other) const;	
函数功能	HandleType类等于比较运算符，判定是否为同一特定服务	
参数（IN）	other	其他HandleType实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	true代表相同服务，false代表不同服务
使用说明	无	
注意事项	无	

函数原型	bool operator<(const HandleType& other) const;	
函数功能	HandleType类小于比较运算符	
参数（IN）	other	其他HandleType实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	true代表当前实例小于输入的HandleType实例，false代表当前实例大于等于输入的HandleType实例
使用说明	无	

注意事项	无

函数原型	const ara::com::InstanceIdentifier& GetInstanceId() const;	
函数功能	获取HandleType实例中的InstanceIdentifier	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	const ara::com::InstanceIdentifier&	该HandleType所包含的IntanceIdentifier的常量引用
使用说明	无	
注意事项	• 线程安全	

2.1.5 SampleAllocateePtr

接口类定义

类名	ara::com::SampleAllocateePtr
命名空间	namespace ara::com
语法	template <typename T> using SampleAllocateePtr = std::unique_ptr<T>
头文件	#include "ara/com/types.h"
说明	指针指向由CM分配数据的地址

2.1.6 MethodCallProcessingMode

接口类定义

类名	ara::com::MethodCallProcessingMode
命名空间	namespace ara::com

语法	enum class MethodCallProcessingMode: uint8_t {
	kPoll = 0,
	kEvent = 1,
	kEventSingleThread = 2
	};

头文件	#include "ara/com/types.h"
说明	定义服务server端的Method处理模式： kPoll：轮询模式 kEvent：事件驱动模式，多线程处理 kEventSingleThread：事件驱动模式，顺序处理

2.1.7 FindServiceHandle

接口类定义

类名	ara::com::FindServiceHandle
命名空间	namespace ara::com
语法	struct FindServiceHandle {...}
头文件	#include "ara/com/types.h"
说明	由StartFindService返回的，用于停止服务发现的对象

成员函数定义

函数原型	bool operator==(const FindServiceHandle& other) const;	
函数功能	FindServiceHandle类等于比较运算符，比较是否为相同的FindServiceHandle	
参数（IN）	other	其他FindServiceHandle实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	true代表相同的FindServiceHandle，false代表不同的FindServiceHandle
使用说明	无	
注意事项	无	

函数原型	bool operator<(const FindServiceHandle& other) const;	
函数功能	FindServiceHandle类小于比较运算符	
参数（IN）	other	其他FindServiceHandle实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	true代表当前实例小于输入的FindServiceHandle实例，false代表当前实例大于等于输入的FindServiceHandle实例
使用说明	无	
注意事项	无	

函数原型	FindServiceHandle& operator=(const FindServiceHandle& other);	
函数功能	InstanceIdentifier类拷贝赋值运算符	
参数（IN）	other	其他FindServiceHandle实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	FindServiceHandle&	FindServiceHandle实例
使用说明	无	
注意事项	无	

2.1.8 ServiceHandleContainer

接口类定义

类名	ara::com::ServiceHandleContainer
命名空间	namespace ara::com
语法	template <typename T> using ServiceHandleContainer = std::vector<T>;
头文件	#include "ara/com/types.h"
说明	包含服务句柄的列表，用作FindService方法的返回值

2.1.9 FindServiceHandler

接口类定义

类名	ara::com::FindServiceHandler
命名空间	namespace ara::com
语法	template <typename T> using FindServiceHandler = std::function<void(ServiceHandleContainer<T>, FindServiceHandle)>
头文件	#include "ara/com/types.h"
说明	调用StartFindService后，所发现的服务列表的回调处理函数

2.1.10 EventReceiveHandler

接口类定义

类名	ara::com::EventReceiveHandler
命名空间	namespace ara::com
语法	using ara::com::EventReceiveHandler = std::function<void()>;
头文件	#include "ara/com/types.h"
说明	收到Event之后的回调处理函数

2.1.11 SubscriptionState

接口类定义

类名	ara::com::SubscriptionState
命名空间	namespace ara::com

语法	enum class SubscriptionState : uint8_t {
	kSubscribed,
	kNotSubscribed,
	kSubscriptionPending
	};

头文件	#include "ara/com/types.h"

说明	定义事件的订阅状态： kSubscribed：已订阅 kNotSubscribed：未订阅 kSubscriptionPending：等待订阅

2.1.12 SubscriptionStateChangeHandler

接口类定义

类名	ara::com::SubscriptionStateChangeHandler
命名空间	namespace ara::com
语法	using ara::com::SubscriptionStateChangeHandler = std::function<void(SubscriptionState)>;
头文件	#include "ara/com/types.h"
说明	Event订阅状态改变之后的回调处理函数

2.1.13 SamplePtr

接口类定义

类名	ara::com::SamplePtr
命名空间	namespace ara::com
语法	template< typename T > class SamplePtr;
头文件	#include "ara/com/types.h"
说明	指向接收到的数据的指针，具有独占语义，类似std::unique_ptr

函数原型	constexpr SamplePtr() noexcept;	
作用域	ara::com	
函数功能	SamplePtr类构造函数	
参数（IN）	None	-
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	默认SamplePtr值为nullptr	

函数原型	SamplePtr (const SamplePtr&) = delete;	
作用域	ara::com	
函数功能	禁用拷贝构造函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	SamplePtr(SamplePtr&& other) noexcept;	
作用域	ara::com	
函数功能	SamplePtr类移动构造函数	
参数（IN）	other	被移动的另一个SamplePtr
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	SamplePtr& operator=(const SamplePtr&) = delete;
作用域	ara::com
函数功能	禁用拷贝赋值运算符

参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	SamplePtr	SamplePtr实例
使用说明	无	
注意事项	无	

函数原型	SamplePtr& operator=(SamplePtr&&) noexcept;	
作用域	ara::com	
函数功能	SamplePtr类移动赋值运算符	
参数（IN）	other	被移动的另一个SamplePtr
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	SamplePtr	SamplePtr实例
使用说明	无	
注意事项	无	

函数原型	T& operator*() const noexcept;	
作用域	ara::com	
函数功能	获取SamplePtr指向的数据	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&	SamplePtr指向的数据
使用说明	无	
注意事项	无	

函数原型	T* operator->() const noexcept;	
作用域	ara::com	
函数功能	获取指向SamplePtr指向的数据的指针	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T*	指向SamplePtr指向的数据的指针
使用说明	无	
注意事项	无	

函数原型	explicit operator bool () const noexept;	
作用域	ara::com	
函数功能	检查SamplePtr是否为空	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	true 代表SamplePtr不为空，false代表SamplePtr为空
使用说明	无	
注意事项	无	

函数原型	void Swap（SamplePtr& other）noexcept;	
作用域	ara::com	
函数功能	交换SamplePtr指向的内容	
参数（IN）	other	被交换内容的另一个SamplePtr
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-

使用说明	无
注意事项	• 线程不安全

函数原型	void Reset (T* ptr) ;	
作用域	ara::com	
函数功能	销毁内部数据并接受新的数据的所有权	
参数（IN）	ptr	指向新数据的指针
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程不安全	

函数原型	T* Get () const noexcept;	
作用域	ara::com	
函数功能	获取指向数据的指针	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T*	指向数据的指针
使用说明	无	
注意事项	• 线程安全	

2.1.14 ComErrc

类名	ara::com::ComErrc
命名空间	ara::com

语法	enum class ComErrc : ara::core::ErrorDomain::CodeType {
	kServiceNotAvailable = 1,
	kMaxSamplesExceeded = 2,
	kNetworkBindingFailure = 3
	};

头文件	#include "ara/com/com_error_domain.h"
说明	CM的错误码： kServiceNotAvailable：表示服务不在线 kMaxSamplesExceeded：表示应用程序持有的SamplePtrs达到了订阅接口设置的缓存的最大值 kNetworkBindingFailure：表示网络绑定时存在本地故障

2.2 Skeleton

Skeleton相关接口头文件均为NAMESPACE/XXX_skeleton.h，其中，“NAMESPACE”表示生成代码中Skeleton类所属的命名空间，“XXX”表示生成代码中定义的服务名称，服务名称（即ARXML中的ServiceInterface的ShortName标签）按照规范用户应以大驼峰风格定义。

函数原型	XXXSkeleton(ara::com::InstanceIdentifier instance, ara::com::MethodCallProcessingMode mode = ara::com::MethodCallProcessingMode::kEvent);	
作用域	NAMESPACE::skeleton::XXXSkeleton（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	Skeleton服务端使用InstanceIdentifier构造函数	
参数（IN）	instance	创建的服务实例ID（ara::com::InstanceIdentifier）
参数（IN）	mode	服务处理method调用的模式，取值范围为{ kPoll、kEvent、kEventSingleThread }，默认kEvent模式
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	XXXSkeleton	服务实例的对象
异常	ara::com::ComException，且其错误码为ara::com::ComErrc::kNetworkBindingFailure	

使用说明	服务类的构造函数，必须继承生成代码中的基类定义子类，并在子类中实现基类中定义的method等函数。该种构造方式构造Skeleton失败会抛出类型为ara::com::ComException的异常。
	示例：
	class RadarImpl : public ara::com::sample::skeleton::RadarSkeleton {
	// 实现基类中定义的method和field getter/setter等函数。}
	std::shared_ptr<RadarImpl> radar1 =
	std::make_shared<RadarImpl>(ara::com::InstanceIdentifier(“1”),
	ara::com::MethodCallProcessingMode::kEvent);
	method模式参数：
	● kEvent：CM server端将开启多线程并发处理所有method请求。
	● kEventSingleThread：CM server端将使用单线程处理所有method请求。
	● kPoll：该参数需配合ProcessNextMethodCall使用。kPoll模式下，CM仅负责接收method请求，但不负责method请求的触发处理，用户需调用ProcessNextMethodCall来触发method请求处理。该模式下，CM将所有收到的method请求存放于队列中，用户每调用一次ProcessNextMethodCall，CM将从队列中取出一个method请求进行处理。“ProcessNextMethodCall”返回值为“ara::core::Future<bool>”。其中，若“future”返回“true”，表明队列中仍存在未处理method请求；若返回“false”，表明队列中无未处理请求。

注意事项	无

函数原型	XXXSkeleton(ara::core::InstanceSpecifier instanceSpec, ara::com::MethodCallProcessingMode mode = ara::com::MethodCallProcessingMode::kEvent);	
作用域	NAMESPACE::skeleton::XXXSkeleton（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	Skeleton服务端使用InstanceSpecifier构造函数，用于创建的某一Port下所有instance实例	
参数（IN）	instanceSpec	Port对应ara::core::InstanceSpecifier InstanceSpecifier的全路径应为executable名/RootSWC名/复合SWC名/Port名称
	mode	服务处理method调用的模式，取值范围为{ kPoll、kEvent、kEventSingleThread }
参数（INOUT）	None	-

参数（OUT）	None	-
异常	ara::com::ComException，且其错误码为ara::com::ComErrc::kNetworkBindingFailure	
返回值	XXXSkeleton	服务实例的对象
使用说明	不支持instanceSpec所表示的Port对于同一种协议绑定两个及以上的ProvidedApServiceInstance（DdsProvidedServiceInstance或ProvidedSomeipServiceInstance）	
注意事项	无	

函数原型	XXXSkeleton(ConstructionToken&&) noexcept;	
作用域	NAMESPACE::skeleton::XXXSkeleton（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	Skeleton服务端使用Construcion构造函数	
参数（IN）	Construction	预构建返回值，通过该值可以创建Skeleton
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	XXXSkeleton	服务实例的对象
使用说明	用户先调用预购建Preconstruct函数，在无异常情况下创建ConstructionToken并预先创建Skeleton，通过返回值ConstructionToken可以用来创建Skeleton。	
注意事项	无	

函数原型	static ara::core::Result<ConstructionToken> ServiceSkeleton::Preconstruct(ara::com::InstanceIdentifier instanceID, ara::com::MethodCallProcessingMode mode = ara::com::MethodCallProcessingMode::kEvent);	
作用域	NAMESPACE::skeleton::XXXSkeleton（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	创建Skeleton前预构建Skeleton	
参数（IN）	InstanceIdentifier	创建的服务实例ID（ara::com::InstanceIdentifier）

参数（IN）	mode	服务处理method调用的模式，取值范围为{ kPoll、kEvent、kEventSingleThread }
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	XXXSkeleton	服务实例的对象
使用说明	Result的value为可以构建Skeleton的ConstructionToken，error为创建失败Skeleton的返回码ara::com::ComErrc::kNetworkBindingFailure。	
注意事项	● 线程安全	

函数原型	static ara::core::Result<ConstructionToken> ServiceSkeleton::Preconstruct(ara::core::InstanceSpecifier instanceSpec, ara::com::MethodCallProcessingMode mode = ara::com::MethodCallProcessingMode::kEvent);	
作用域	NAMESPACE::skeleton::XXXSkeleton（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	创建Skeleton前预构建Skeleton	
参数（IN）	InstanceSpec	Port对应InstanceSpecifier
	mode	服务处理method调用的模式，取值范围为{ kPoll、kEvent、kEventSingleThread }
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	XXXSkeleton	服务实例的对象
使用说明	Result的value为可以构建Skeleton的ConstructionToken，error为创建失败Skeleton的返回码ara::com::ComErrc::kNetworkBindingFailure。 不支持instanceSpec所表示的Port对于同一种协议绑定两个及以上的ProvidedApServiceInstance（DdsProvidedServiceInstance或ProvidedSomeipServiceInstance）	
注意事项	● 线程安全	

2.2.1 Offering Service

函数原型	void OfferService();	
作用域	NAMESPACE::skeleton::XXXSkeleton（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	创建Skeleton并发布服务	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	创建的Skeleton发布服务，OfferService后可被客户端发现服务。 若Skeleton包含Method，OfferService后，CM将启动1个或多个线程用于处理Proxy发送的method请求，关闭以上线程需要用户调用StopOfferService。因此用户在主程序退出或其他不需要服务的时刻，必须调用StopOfferService，防止在某些服务所需数据已无效，或Skeleton已析构后，Method请求回调仍被调用，导致出现已析构或无效变量被错误调用情况。	
注意事项	• 线程不安全	

函数原型	void StopOfferService();	
作用域	NAMESPACE::skeleton::XXXSkeleton（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	停止Skeleton服务发布	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	服务端接口，停止服务。StopOfferService后不可被客户端服务发现。	
注意事项	• 线程不安全 • 该接口需用户在skeleton对象析构前调用，保证资源释放，否则产生未定义行为。	

2.2.2 Method

函数原型	ara::com::Future<bool> ProcessNextMethodCall();	
作用域	NAMESPACE::skeleton::XXXSkeleton（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	服务端接口，处理队列中的下一个Method调用请求。 该接口仅在kPoll模式下使用，该模式下CM仅负责接收Method请求，不负责Method请求的触发处理，用户需调用ProcessNextMethodCall来触发Method请求处理。该模式下，CM将所有收到的Method请求存放于队列中，用户每调用一次ProcessNextMethodCall，CM将从队列中取出一个Method请求进行处理。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Future<bool>	处理完当前请求后，若队列中仍有待处理的Method请求，则返回值为true，否则为false。
使用说明	无	
注意事项	• 线程安全	

函数原型	bool SetMethodThreadNumber(const std::uint16_t& number, const std::uint16_t& queueSize);	
作用域	NAMESPACE::skeleton::XXXSkeleton（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	设置Method线程数	
参数（IN）	Number	线程数目
	queueSize	队列长度
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	true表示设置成功，false表示设置失败

使用说明	该函数仅在kEvent模式有效，用于用户自定义Method处理线程数目。 用户可不调用该接口。默认情况下，线程数目为Method数目+1。但最大不超过32，默认任务队列长度为1024，若任务队列满，接收到的Method请求被丢弃。 注： 1. 用户可设最大线程数不可超过32，若超过32，CM将调用ara::core::Abort(...)，终止程序。 2. 该函数仅在OfferService前调用有效。
注意事项	● 线程不安全

2.2.3 Event

函数原型	void Send(const SampleType &data)	
命名空间	NAMESPACE::skeleton::XXXSkeleton::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::skeleton::events::<VariableDataPrototype.shortName>	
函数功能	Event数据发送	
参数（IN）	data	发送的数据包
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	服务端接口，发送Event。	
注意事项	● 线程安全 ● 用户在外部分配的内存，调用Send接口时，以引用的方式将该内存地址传递给了CM。在Send执行过程中，CM会复制该内存数据，因此在Send返回之前，调用者需保证不会修改和删除发送的数据内存。	

函数原型	ara::com::SampleAllocateePtr<SampleType> Allocate();

作用域	NAMESPACE::skeleton::XXXSkeleton::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::skeleton::events::<VariableDataPrototype.shortName>	
函数功能	分配要发送数据的内存	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::com::SampleAllocateePtr <SampleType>	指向成功分配的内存的指针

使用说明	服务端接口，为发送的Event分配内存，返回的内存可按照该Event引用的数据结构直接赋值（详见具体类型的生成代码）。
	示例：
	auto l_sampleRadar = radar1.XXXEvent.Allocate();
	l_sampleRadar.member1 = 1;
	l_sampleRadar.member2 = 2;	

注意事项	• 线程安全	

函数原型	void Send(ara::com::SampleAllocateePtr <SampleType> dataPtr);	
作用域	NAMESPACE::skeleton::XXXSkeleton::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::skeleton::events::<VariableDataPrototype.shortName>	
函数功能	Event数据发送	
参数（IN）	ara::com::SampleAllocateePtr <SampleType> dataPtr	存放需要发送的数据的指针
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	None	-
使用说明	针对CM模块的事件发送方法，CM将dataPtr中的数据发送至订阅者，发送过程中，应用程序禁止进行对应event数据的修改。 注：该函数需配合Allocate()函数使用，该函数入参为Allocate的返回值，数据量最大为序列化后16M，如果使用DDS协议，需为DDS协议头预留1KB空间。 示例： radar1. <VariableDataPrototype.ShortName>.Send(...);	
注意事项	• 线程安全	

2.2.4 Field

函数原型	void RegisterGetHandler(std::function<ara::core::Future<FieldType>()> getHandler);	
作用域	NAMESPACE::skeleton::XXXSkeleton::<Field.shortName>（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称，<Field.shortName>表示ARXML中定义的Field的ShortName，另根据规范该ShortName应为大驼峰风格） Field数据类型：NAMESPACE::skeleton::fields::<Field.shortName>	
函数功能	服务端接口，注册具体Field的Get请求的处理函数	
参数（IN）	getHandler	Get请求的处理函数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	注册Get请求的处理函数，收到Get请求时会触发	
注意事项	• 线程不安全	

函数原型	void RegisterSetHandler(std::function<ara::core::Future<FieldType>(const FieldType& value)> setHandler);

作用域	NAMESPACE::skeleton::XXXSkeleton::<Field.shortName>（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称，<Field.shortName>表示ARXML中定义的Field的ShortName，另根据规范该ShortName应为大驼峰风格） Field数据类型：NAMESPACE::skeleton::fields::<Field.shortName>	
函数功能	服务端接口，注册具体Field的Set请求的处理函数	
参数（IN）	setHandler	Set请求的处理函数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	若Field.hasSetter=true，用户在OfferService前未调用RegisterSetHandler，则CM将调用ara::core::Abort终止程序。 SetHandler完成后，CM会将SetHandler返回结果返回至Set请求方，并以notification方式将发送至所有已订阅的Proxy。	
注意事项	● 线程不安全	

函数原型	void Update(const FieldType &value);	
作用域	NAMESPACE::skeleton::XXXSkeleton::<Field.shortName>（其中NAMESPACE表示生成代码中Skeleton类所属命名空间，XXX表示生成代码中定义的服务名称，<Field.shortName>表示ARXML中定义的Field的ShortName，另根据规范该ShortName应为大驼峰风格） Field数据类型：NAMESPACE::skeleton::fields::<Field.shortName>	
函数功能	服务端接口，更新Field的值。该接口类似于Event的Send接口，会向订阅的客户端发送新的Field值。	
参数（IN）	const FieldType &value	待更新Field值
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	若Field.hasNotifier = true或者Field.hasGetter = true且GetHandler未注册，用户调用OfferService前应调用Update对Field赋初值，否则CM将调用ara::core::Abort终止程序。 若Update传入的值与上一次Update传入值相同，CM不会再次发送Update传入值。	

注意事项	● 线程安全 ● 用户在外部分配的内存，调用Update接口时，以引用的方式将该内存地址传递给了CM;在Update执行过程中，CM会复制该内存数据，因此在Update返回之前，调用者需保证不会修改和删除发送的数据内存。

2.3 Proxy

函数原型	explicit XXXProxy(const HandleType &handle);	
作用域	NAMESPACE::proxy::XXXProxy（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	Proxy构造函数	
参数（IN）	handle	具体服务实例的句柄，该句柄是通过StartFindService接口（异步）或FindService接口（同步）获取。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	XXXProxy	服务代理实例的对象。
异常	ara::com::ComException，且其错误码为ara::com::ComErrc::kNetworkBindingFailure	
使用说明	客户端接口，创建指向某个服务实例的代理（Proxy），通过该代理实现具体的数据通信功能。 示例： `void serviceAvailabilityCallback(ara::com::ServiceHandleContainer<Proxy::HandleType> handles, ara::com::FindServiceHandle handler){ for(auto handle : handles) { if(handle.GetInstanceId() == 1 && radarProxy1 == nullptr) { std::shared_ptr<Proxy> radarProxy1 = std::make_shared<Proxy>(handle); } } }`	
注意事项	● 对于多重绑定场景，不支持在同一个进程内使用分别绑定了两种协议的两个Proxy与绑定了两种协议的同一个Skeleon通信，即，server进程内Skeleton绑定了DDS和SOMEIP两种协议，client仅可使用DDS或SOMEIP其中一种协议创建一个Proxy与server端Skeleton通信。	

函数原型	static ara::core::Result<ConstructionToken> ServiceProxy::Preconstruct(HandleType &handle);	
作用域	NAMESPACE::Proxy::XXXProxy（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	通过预构建方式创建用来创建Preconstruct的ConstructionToken	
参数（IN）	handle	具体服务实例的句柄，该句柄是通过StartFindService接口（异步）或FindService接口（同步）获取。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Result	返回的数据类型Result的Value为创建成功的ConstructionToken，errorcode为创建失败的错误码ara::com::ComErrc::kNetworkBindingFailure
使用说明	通过预构建方式创建Proxy，若在创建Proxy过程中失败，则会返回为Result的错误码（kNetworkBindingFailure）。创建成功Result内会存储ConstructionToken可以直接用来创建Proxy实例。	
注意事项	• 线程安全	

函数原型	explicit XXXProxy(ConstructionToken&&) noexcept;	
作用域	NAMESPACE::proxy::XXXProxy（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	通过预构建方式创建Proxy	
参数（IN）	ConstructionToken	通过预构建创建的ConstructionToken可以通过该入参无异常创建Proxy。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	XXXProxy	服务代理实例的对象
使用说明	通过预构建的Token创建XXXProxy实例。	

注意事项	• 线程安全 • 对于多重绑定场景，不支持在同一个进程内使用分别绑定了两种协议的两个Proxy与绑定了两种协议的同一个Skeleton通信，即，server进程内Skeleton绑定了DDS和SOMEIP两种协议，client仅可使用其中一种协议DDS或SOMEIP创建一个Proxy与server端Skeleton通信。

2.3.1 Finding Services

函数原型	static ara::com::FindServiceHandle StartFindService(ara::com::FindServiceHandler<XXXProxy::HandleType> handler, ara::com::InstanceIdentifier instanceId);	
作用域	NAMESPACE::proxy::XXXProxy（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	注册服务发现回调并开始进行服务发现	
参数（IN）	handler	应用向CM组件注册的监听服务变化的回调函数。
	instanceId	应用指定期望发现的服务实例ID（ara::com::InstanceIdentifier）
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	FindServiceHandle	对应此次服务发现请求的句柄，该句柄用于后续停止服务发现监测。
使用说明	客户端接口，启动服务发现持续监测，当服务状态发生变化时调用handler函数。 注意： 由于StartFindService调用后，CM将启动服务发现回调线程，用于将发现的服务以调用用户注册的回调函数的方式返回至上层。用户退出程序前应调用对应StopFindService停止服务发现回调的触发及对应线程，防止在某些服务所需数据已无效，但仍然触发服务发现回调，导致程序异常。	
注意事项	• 线程安全	

函数原型	static ara::com::FindServiceHandle StartFindService(ara::com::FindServiceHandler<XXXProxy::HandleType> handler, ara::core::InstanceSpecifier instance);

作用域	NAMESPACE::proxy::XXXProxy（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	注册服务发现回调并开始进行服务发现，使用InstanceSpecifier，指定发现某一Port对应的所有的InstanceIdentifier实例	
参数（IN）	handler	应用向CM组件注册的监听服务变化的回调函数。
	instance	Port对应InstanceSpecifier。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	FindServiceHandle	对应此次服务发现请求的句柄，该句柄用于后续停止服务发现监测。
使用说明	客户端接口，启动服务发现持续监测，当服务状态发生变化时调用handler函数。 注意： 由于StartFindService调用后，CM将启动服务发现回调线程，用于将发现的服务以调用用户注册的回调函数的方式返回至上层。用户退出程序前应调用对应StopFindService停止服务发现回调的触发及对应线程，防止在某些服务所需数据已无效，但仍然触发服务发现回调，导致程序异常。	
注意事项	• 线程安全	

函数原型	static ara::com::ServiceHandleContainer<XXXProxy::HandleType> FindService(ara::com::InstanceIdentifier instanceId);	
作用域	NAMESPACE::proxy::XXXProxy（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	一次性服务发现	
参数（IN）	instanceId	应用指定期望发现的服务实例ID（ara::com::InstanceIdentifier）。若该参数为空，则表示发现该服务下的所有实例。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ServiceHandleContainer	返回当前可用的服务实例列表。

使用说明	客户端接口，“一次性”服务发现请求，向CM组件查询当前可用的服务实例，并立即返回。若当前无可用服务实例，则返回的列表为空。
注意事项	● 线程安全

函数原型	static ara::com::ServiceHandleContainer<XXXProxy::HandleType> FindService(ara::core::InstanceSpecifier instance);	
作用域	NAMESPACE::proxy::XXXProxy（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	一次性服务发现。使用InstanceSpecifier，指定发现某一Port对应的所有当前存在的InstanceIdentifier实例	
参数（IN）	Instance	Port对应InstanceSpecifier
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ServiceHandleContainer	返回当前可用的服务实例列表
使用说明	客户端接口，“一次性”服务发现请求，向CM组件查询当前可用的服务实例，并立即返回。若当前无可用服务实例，则返回的列表为空。	
注意事项	● 线程安全	

函数原型	void StopFindService(ara::com::FindServiceHandle handle);	
作用域	NAMESPACE::proxy::XXXProxy（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	停止服务发现检测	
参数（IN）	handle	服务发现请求的句柄，填入StartFindService接口的返回值。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-

使用说明	客户端接口，停止服务发现持续监测。若StopFindService在非服务发现回调函数中被调用，可保证StopFindService函数调用结束后，对应注册的服务发现回调函数已经调用结束且不会被再次触发。若StopFindService在服务发现回调函数中调用，仅保证服务发现回调函数不会再次被触发，但StopFindService前已经触发的服务发现回调无法保证已经处理结束，因此不建议在服务发现回调中调用StopFindService。
注意事项	• 线程安全

函数原型	HandleType GetHandle();	
作用域	NAMESPACE::proxy::XXXProxy（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称）	
函数功能	获取Proxy创建所用HandleType	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	HandleType	Proxy创建所用HandleType
使用说明	无	
注意事项	• 线程安全	

2.3.2 Event

函数原型	void Subscribe(size_t maxSampleCount);	
作用域	NAMESPACE::proxy::XXXProxy::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::proxy::events::<VariableDataPrototype.shortName>	
函数功能	订阅Event事件，并根据参数设置最大预分配内存数量	
参数（IN）	maxSampleCount	订阅该Event时分配的最大数据槽位，CM会自动按照maxSampleCount预先分配内存空间。

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	maxSampleCount取值范围为(0，1000]。 若Subscribe设置值不在取值范围内，或小于等于1000但内存分配失败则CM调用ara::core::Abort，终止程序。	
注意事项	● 线程不安全	

函数原型	void Unsubscribe();	
作用域	NAMESPACE::proxy::XXXProxy::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::proxy::events::<VariableDataPrototype.shortName>	
函数功能	取消Event的订阅	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	客户端接口，取消订阅event 注意： 如正在执行回调函数，则该接口会同步等待正在执行的回调函数执行结束	
注意事项	● 线程不安全	

函数原型	bool IsSubscribed();

作用域	NAMESPACE::proxy::XXXProxy::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::proxy::events::<VariableDataPrototype.shortName>	
函数功能	判断该Event是否已订阅	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	true表示已订阅，false表示未订阅。
使用说明	客户端接口，获取event的订阅状态。	
注意事项	• 线程安全	

函数原型	ara::core::Result<size_t> GetNewSamples(F&& f, size_t maxNumberOfSamples = std::numeric_limits<size_t>::max());	
作用域	NAMESPACE::proxy::XXXProxy::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::proxy::events::<VariableDataPrototype.shortName>	
函数功能	使用接收到的数据更新CM缓冲区，并将缓冲区数据反馈至用户	
参数（IN）	f	数据处理回调函数，该回调函数为void(ara::com::SamplePtr<SampleType const>)

	maxNumberOfSamples	一次GetNewSamples调用取出的最大数据个数，默认值std::numeric_limits<size_t>::max()。若使用默认值，将取当前缓冲区中的所有数据；若使用者不使用默认值，则会触发多次f回调直到到达maxNumberOfSamples或者取空缓冲区数据。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Result<size_t>	返回值value为本次调用接收到的数据个数，errorcode可能为kMaxSamplesReached，表示目前缓冲区已经达到Subscribe订阅最大数据槽位。
使用说明	客户端接口，从缓冲区获取收到的服务端数据。CM将调用f函数，将缓冲区获取到的数据逐个反馈至用户	
注意事项	• 线程不安全	

函数原型	size_t GetFreeSampleCount() const noexcept;	
作用域	NAMESPACE::proxy::XXXProxy::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::proxy::events::<VariableDataPrototype.shortName>	
函数功能	查询目前可用SamplePtr数量	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	size_t	目前可使用的SamplePtr最大数据槽位
使用说明	返回值为目前可使用的SamplePtr最大数据槽位	
注意事项	• 线程安全	

函数原型	void SetReceiveHandler(ara::com::EventReceiveHandler handler);	
作用域	NAMESPACE::proxy::XXXProxy::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::proxy::events::<VariableDataPrototype.shortName>	
函数功能	设置收数回调函数	
参数（IN）	handler	使用者注册的当CM接收到对端数据时会触发的回调函数。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	客户端接口，设置Event数据接收回调函数。CM组件会保证该回调函数串行调用，无需考虑多线程调用。 如果调用SetReceiveHandler，则再次调用SetReceiveHandler函数无法覆盖第一次设置的回调，需要调用UnsetReceiveHandler接口解除回调后再调用SetReceiveHandler设置新回调。	
注意事项	● 线程安全	

函数原型	void UnsetReceiveHandler()	
作用域	NAMESPACE::proxy::XXXProxy::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::proxy::events::<VariableDataPrototype.shortName>	
函数功能	取消收数回调函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-

使用说明	客户端接口，取消Event数据接收回调函数。
注意事项	● 线程安全

函数原型	ara::com::SubscriptionState GetSubscriptionState()	
作用域	NAMESPACE::proxy::XXXProxy::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::proxy::events::<VariableDataPrototype.shortName>	
函数功能	获得目前订阅状态	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::com::SubscriptionState	有三个值分别为kSubscribed，kNotSubscribed，kSubscriptionPending。 分别是订阅，未订阅与悬挂状态。
使用说明	获取目前订阅状态。	
注意事项	● 线程安全 注意：server端的停止响应，导致客户端在服务超时期间，无法感知对端是否在线。导致Event的订阅状态不会从kSubscribed状态切换到kSubscriptionPending状态。其中DDS异常下线时间超时通知最长为10s。Someip应用异常下线会立即通知，若someipd被kill掉则不会通知。	

函数原型	void SetSubscriptionStateChangeHandler(ara::com::SubscriptionStateChangeHandler handler)

作用域	NAMESPACE::proxy::XXXProxy::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::proxy::events::<VariableDataPrototype.shortName>	
函数功能	设置状态改变回调函数	
参数（IN）	handler	当订阅状态改变时，CM触发的使用者注册的回调函数。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	void	-
使用说明	注册订阅状态改变时的回调函数	
注意事项	● 线程安全 注意：server端停止响应，导致客户端在服务超时期间，无法感知对端是否在线。导致Event的订阅状态不会从kSubscribed状态切换到kSubscriptionPending状态。其中DDS异常下线时间超时通知最长为10s。Someip应用异常下线会立即通知，若someipd被kill掉则不会通知。	

函数原型	void UnsetSubscriptionStateChangeHandler()	
作用域	NAMESPACE::proxy::XXXProxy::<VariableDataPrototype.ShortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<VariableDataPrototype.ShortName>表示ARXML中定义的Event的ShortName，另根据规范该ShortName应为大驼峰风格） Events数据类型：NAMESPACE::proxy::events::<VariableDataPrototype.shortName>	
函数功能	注销订阅状态改变回调函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明		

注意事项	● 线程安全

2.3.3 Method

函数原型	ara::core::Future<Output> operator()(TypeInputParameter1 input1,TypeInputParameter2 input2,...);	
函数功能	客户端接口，调用具体method。该接口调用会立即返回，method执行结果从返回值的ara::com::Future中获取。	
作用域	NAMESPACE::proxy::XXXProxy::<ClientServerOperation.shortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<ClientServerOperation.shortName>表示ARXML中定义的Method的ShortName，另根据规范该ShortName应为大驼峰风格） method数据类型：NAMESPACE::proxy::methods::<ClientServerOperation.shortName>	
参数（IN）	input1/input2/...	Method对应参数序列，可为void类型
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Future<Output> （如果为fire&forget方式，返回值为void）	Method对应返回值构成的结构体，可为void类型
使用说明	该函数可能返回正常值，或错误码kNetworkBindingFailure（服务双方method数据类型不符导致反序列化失败，或协议层内部错误）、kServiceNotAvailable（server调用了StopOfferService停止提供服务） 注意： 1. server端停止响应，导致的method应答消息丢失不能引起kServiceNotAvailable的返回，反而会导致client端method请求超时。如果调用future的get或GetResult方法等待，异常下线超时后，CM会返回错误码。其中DDS超时时间最长为10s。Someip应用异常下线会立即通知，若someipd被kill掉则不会通知。 2. 如果method设置为fire&forget模式，返回值为void，不是ara::core::Future<void>。	
注意事项	● 线程安全	

2.3.4 Field

函数原型	void Subscribe(size_t maxSampleCount);	
作用域	NAMESPACE::proxy::XXXProxy::<Field.shortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<Field.shortName>表示ARXML中定义的Field的ShortName，另根据规范该ShortName应为大驼峰风格） Field数据类型：NAMESPACE::proxy::fields::<Field.shortName>	
函数功能	客户端接口，订阅Field，该接口类似于Event的Subscribe接口。	
参数（IN）	size_t maxSampleCount	订阅时分配的最大数据槽位，CM会自动按照maxSampleCount预先分配内存空间。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程不安全 • maxSampleCount最大值为1000	

函数原型	void Unsubscribe();	
作用域	NAMESPACE::proxy::XXXProxy::<Field.shortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<Field.shortName>表示ARXML中定义的Field的ShortName，另根据规范该ShortName应为大驼峰风格） Field数据类型：NAMESPACE::proxy::fields::<Field.shortName>	
函数功能	客户端接口，取消订阅Field，该接口类似于Event的Unsubscribe接口。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程不安全	

函数原型	bool IsSubscribed();	
作用域	NAMESPACE::proxy::XXXProxy::<Field.shortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<Field.shortName>表示ARXML中定义的Field的ShortName，另根据规范该ShortName应为大驼峰风格） Field数据类型：NAMESPACE::proxy::fields::<Field.shortName>	
函数功能	判断该Field是否已订阅，该接口类似于Event的IsSubscribed接口。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	true表示已订阅，false表示未订阅
使用说明	无	
注意事项	• 线程安全	

函数原型	ara::core::Result<size_t> GetNewSamples(F&& f, size_t maxNumberOfSamples = std::numeric_limits<size_t>::max());	
作用域	NAMESPACE::proxy::XXXProxy::<Field.shortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<Field.shortName>表示ARXML中定义的Field的ShortName，另根据规范该ShortName应为大驼峰风格） Field数据类型：NAMESPACE::proxy::fields::<Field.shortName>	
函数功能	客户端接口，从缓冲区获取收到的数据	
参数（IN）	f	数据处理回调函数，该回调函为void(ara::com::SamplePtr<SampleType const>)
	maxNumberOfSamples	一次GetNewSamples调用取出的最大数据个数 默认值std::numeric_limits<size_t>::max()，若使用默认值，将取当前缓冲区中的所有数据
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	ara::core::Result<size_t>	本次调用接收到的数据个数，errorcode可能为kMaxSamplesReached，表示目前缓冲区已经达到Subscribe订阅最大数据槽位。
使用说明	客户端接口，从缓冲区获取收到的服务端数据。CM将调用f函数，将缓冲区获取到的数据逐个反馈至用户	
注意事项	● 线程不安全	

函数原型	void SetReceiveHandler(ara::com::EventReceiveHandler handler);	
作用域	NAMESPACE::proxy::XXXProxy::<Field.shortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<Field.shortName>表示ARXML中定义的Field的ShortName，另根据规范该ShortName应为大驼峰风格） Field数据类型：NAMESPACE::proxy::fields::<Field.shortName>	
函数功能	客户端接口，设置Field数据接收回调函数，该接口类似于Event的SetReceiveHandler接口。	
参数（IN）	ara::com::EventReceiveHandler handler	客户端接收到Field消息的回调函数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	● 线程安全	

函数原型	void UnsetReceiveHandler()	
作用域	NAMESPACE::proxy::XXXProxy::<Field.shortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<Field.shortName>表示ARXML中定义的Field的ShortName，另根据规范该ShortName应为大驼峰风格） Field数据类型：NAMESPACE::proxy::fields::<Field.shortName>	
函数功能	客户端接口，取消Field数据接收回调函数，该接口类似于Event的UnsetReceiveHandler接口。	
参数（IN）	None	-

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程安全	

函数原型	ara::com::Future<FieldType> Get();	
作用域	NAMESPACE::proxy::XXXProxy::<Field.shortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<Field.shortName>表示ARXML中定义的Field的ShortName，另根据规范该ShortName应为大驼峰风格） Field数据类型：NAMESPACE::proxy::fields::<Field.shortName>	
函数功能	客户端接口，获取Field值。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::com::Future<FieldType>	含获取到的Field值的future
使用说明	该函数可能返回错误码kNetworkBindingFailure（服务双方Method数据类型不符导致反序列化失败，或协议层内部错误）、kServiceNotAvailable（server调用了StopOfferService停止提供服务）	
注意事项	• 线程安全	

函数原型	ara::com::Future<FieldType> Set(const FieldType& value);	
作用域	NAMESPACE::proxy::XXXProxy::<Field.shortName>（其中NAMESPACE表示生成代码中Proxy类所属命名空间，XXX表示生成代码中定义的服务名称，<Field.shortName>表示ARXML中定义的Field的ShortName，另根据规范该ShortName应为大驼峰风格） Field数据类型：NAMESPACE::proxy::fields::<Field.shortName>	
函数功能	客户端接口，设置Field值。	
参数（IN）	const FieldType& value	待设置Field值
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	ara::com::Future<FieldType>	Field实际设置结果
使用说明	该函数可能返回错误码kNetworkBindingFailure（服务双方Method数据类型不符导致反序列化失败或协议层内部错误）、kServiceNotAvailable（server调用了StopOfferService停止提供服务）	
注意事项	• 线程安全	

3 CoreType

3.1 ErrorDomain

本章节描述ara::core::ErrorDomain数据类型，该数据类型是构造错误域类的抽象基类，为字面类型。

接口类定义

类名	ara::core::ErrorDomain
命名空间	namespace ara::core
语法	class ErrorDomain {...};
头文件	#include "ara/core/error_domain.h"
说明	封装了一个错误域（ErrorDomain）。错误域是错误码值的控制实体，它定义了错误码值到文本表示的映射。具体的错误域类应从此虚基类继承实现

类内数据类型定义

类型名	定义	说明
IdType	std::uint64_t	ErrorDomain标识符类型别名，不同ErrorDomain具有唯一标识符
CodeType	std::int32_t	ErrorDomain内错误码数据类型别名
SupportDataType	std::int32_t	补充数据类型别名（由于1911规范定义该类型为IMPLEMENTATION_DEFINED，此处延用1810规范定义的数据类型std::int32_t）

成员函数定义

函数原型	ErrorDomain(ErrorDomain const &) = delete;	
函数功能	ErrorDomain禁用拷贝构造函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	None	
注意事项	无	

函数原型	ErrorDomain(ErrorDomain &&) = delete;	
函数功能	ErrorDomain禁用移动构造函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	None	
注意事项	无	

函数原型	explicit constexpr ErrorDomain (IdType id) noexcept;	
函数功能	使用给定的标识符构造一个ErrorDomain实例	
参数（IN）	id	ErrorDomain唯一的标识符
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数为protected类型，仅用在子类ErrorDomain中，初始化基类ErrorDomain。	
注意事项	每种ErrorDomain的id值在系统范围内应该是唯一的	

函数原型	~ErrorDomain()=default;	
函数功能	析构函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	None	
注意事项	无	

函数原型	ErrorDomain& operator= (ErrorDomain const &)=delete;	
函数功能	ErrorDomain禁用拷贝赋值	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	None	
注意事项	无	

函数原型	ErrorDomain& operator= (ErrorDomain &&)=delete;	
函数功能	ErrorDomain禁用移动赋值	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	None	
注意事项	无	

函数原型	constexpr bool operator== (ErrorDomain const &other) const noexcept;	
函数功能	操作符重载，同另一个ErrorDomain实例进行比较。如果相同返回true，否则返回false。	
参数（IN）	other	另一个ErrorDomain实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	相同返回true，否则返回false
使用说明	无	
注意事项	无	

函数原型	constexpr bool operator!= (ErrorDomain const &other) const noexcept;	
函数功能	操作符重载，同另一个ErrorDomain实例进行比较。如果不相同返回true，否则返回false。	
参数（IN）	other	另一个ErrorDomain实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	不相同返回true，否则返回false
使用说明	无	
注意事项	无	

函数原型	constexpr IdType Id () const noexcept;	
函数功能	返回唯一的ErrorDomain识别码。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Id	返回唯一的ErrorDomain识别码。
使用说明	无	
注意事项	• 线程安全	

函数原型	virtual char const* Name () const noexcept=0;	
函数功能	返回错误域的名称，返回值不会是nullptr。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	const char*	返回错误域的名称，返回值不会是nullptr
使用说明	该函数为纯虚函数，语法见子类，如CoreErrorDomain	

注意事项	无

函数原型	virtual char const* Message (CodeType errorCode) const noexcept=0;	
函数功能	返回给定错误码对应的文本描述	
参数（IN）	errorCode	特定错误域的错误码
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	char const*	返回给定错误码对应的文本描述
使用说明	该函数为纯虚函数，语法见子类，如CoreErrorDomain	
注意事项	无	

函数原型	virtual void ThrowAsException (ErrorCode const &errorCode) const noexcept(false)=0;	
函数功能	根据指定的ErrorCode抛出异常	
参数（IN）	errorCode	指定错误码
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数为纯虚函数，语法见子类，如CoreErrorDomain	
注意事项	无	

3.2 ErrorCode

本章节描述ara::core::ErrorCode数据类型，该数据类型根据特定的域进行描述错误。

接口类定义

类名	ara::core::ErrorCode
命名空间	namespace ara::core
语法	class ErrorCode final {...};

头文件	#include "ara/core/error_code.h"
说明	ErrorCode包含原始错误码值和错误域。原始错误码值为特定于此错误域。

成员函数定义

函数原型	template <typename EnumT> constexpr ErrorCode (EnumT e, ErrorDomain::SupportDataType data=ErrorDomain::SupportDataType()) noexcept;	
函数功能	ErrorCode构造函数	
参数（IN）	e	错误码对应枚举类型
	data	补充数据（可选）
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	ErrorCode构造函数，输入枚举类型，需具备对应错误域定义、对应MakeErrorCode定义，如future_errc、CoreErrc等	
注意事项	无	

函数原型	constexpr ErrorCode (ErrorDomain::CodeType value, ErrorDomain const &domain, ErrorDomain::SupportDataType data=ErrorDomain::SupportDataType()) noexcept;	
函数功能	ErrorCode构造函数	
参数（IN）	value	域内指定错误码对应数值
	domain	指定错误域
	data	补充数据（可选）
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	ErrorCode构造函数，输入value与domain需具备关联关系，如future_errc与FutureErrorDomain，CoreErrc与CoreErrorDomain	
注意事项	无	

函数原型	constexpr ErrorDomain::CodeType Value () const noexcept;	
函数功能	获取原始错误码的值	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ErrorDomain::CodeType	原始错误码值
使用说明	获取原始错误码的值	
注意事项	• 线程安全	

函数原型	constexpr ErrorDomain const& Domain () const noexcept;	
函数功能	获取该ErrorCode所关联的错误域	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ErrorDomain const&	该ErrorCode所关联的错误域
使用说明	获取该ErrorCode所关联的错误域	
注意事项	• 线程安全	

函数原型	constexpr ErrorDomain::SupportDataType SupportData () const noexcept;	
函数功能	获取该ErrorCode实例所包含的用户自定义补充数据	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ErrorDomain::SupportDataType	用户自定义补充数据
使用说明	无	
注意事项	• 线程安全	

函数原型	StringView Message () const noexcept;	
函数功能	根据ErrorCode实例所包含的错误码，返回对应的字符串文本说明	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	StringView	错误码对应文字说明
使用说明	该错误码值在对应错误域应具备实际意义	
注意事项	• 线程安全	

函数原型	void ThrowAsException () const;	
函数功能	抛出错误码对应异常	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	此函数将决定该错误码对应的异常类型，并抛出它，抛出的异常类型包含该错误码	
注意事项	• 线程安全	

3.2.1 全局函数定义

函数原型	constexpr bool operator== (ErrorCode const &lhs, ErrorCode const &rhs) noexcept;	
函数功能	ErrorCode支持==运算符	
参数（IN）	lhs	要比较的ErrorCode的实例1
	rhs	要比较的ErrorCode的实例2
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	bool	比较的结果，如果两个实例相等，返回true；否则，返回false。
使用说明	如果两个ErrorCode的Value()及Domain()结果相等，则认为两个ErrorCode相等，SupportData()结果是否相等，不予考虑。	
注意事项	无	

函数原型	constexpr bool operator!= (ErrorCode const &lhs, ErrorCode const &rhs) noexcept;	
函数功能	ErrorCode支持!=运算符	
参数（IN）	lhs	要比较的ErrorCode的实例1
	rhs	要比较的ErrorCode的实例2
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	比较的结果，如果两个实例不相等，返回true；否则，返回false。
使用说明	如果两个ErrorCode的Value()及Domain()结果相等，则认为两个ErrorCode相等，SupportData()结果是否相等，不予考虑。	
注意事项	无	

3.3 Exception

接口类定义

类名	ara::core::Exception
命名空间	namespace ara::core
语法	class Exception : public std::exception {...};
头文件	#include "ara/core/exception.h"
说明	ara::core::Exception继承std::exception

成员函数定义

函数原型	explicit Exception (ErrorCode err) noexcept;	
函数功能	构造函数	
参数（IN）	err	错误码
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	根据指定的ErrorCode构造一个新的异常实例	
注意事项	无	

函数原型	char const* what () const noexcept override;	
函数功能	根据实例化时输入的ErrorCode实例，以字符串的形式返回ErrorCode所包含的错误码的含义。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	char const*	异常文字说明
使用说明	该函数重写std::exception的what函数，详细用法参考std::exception::what	
注意事项	• 线程安全	

函数原型	ErrorCode const& Error () const noexcept;	
函数功能	获取异常对应ErrorCode实例	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ErrorCode const&	异常对应ErrorCode
使用说明	无	
注意事项	• 线程安全	

3.4 Result

本章节描述ara::core::Result<T,E>数据类型，该数据类型要么包含一个T类型的值，要么包含一个E类型的错误。

接口类定义

类名	ara::core::Result
命名空间	namespace ara::core
语法	template <typename T, typename E = ErrorCode> class Result final {...};
头文件	#include "ara/core/result.h"
说明	错误类型仅支持ErrorCode

类内数据类型定义

类型名	定义	说明
value_type	T（输入模板类）	该Result包含的数值类型，支持T为void类型
error_type	E（默认为ErrorCode，仅支持ErrorCode数据类型）	该Result包含的错误类型

成员函数定义

函数原型	ara::core::Result< T, E >::Result (T const &t);	
函数功能	构造函数，根据传入的T类型的左值构造一个新的Result实例	
参数（IN）	t	T类型左值
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	构造含数值t的Result	
注意事项	无	

函数原型	ara::core::Result< T, E >::Result (T &&t);	
函数功能	构造函数，根据传入的T类型的右值构造一个新的Result实例	
参数（IN）	t	T类型右值
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	构造含数值t的Result	
注意事项	无	

函数原型	explicit ara::core::Result< T, E >::Result (E const &e);	
函数功能	构造函数，根据传入的E类型的左值构造一个新的Result实例	
参数（IN）	e	E类型左值
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	构造含错误码e的Result	
注意事项	无	

函数原型	explicit ara::core::Result< T, E >::Result (E &&e);	
函数功能	构造函数，根据传入的E类型的右值构造一个新的Result实例	
参数（IN）	e	E类型右值
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	构造含错误码e的Result	
注意事项	无	

函数原型	ara::core::Result< T, E >::Result (Result const &other);	
函数功能	拷贝构造函数	
参数（IN）	other	左值Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	ara::core::Result< T, E >::Result (Result &&other) noexcept(std::is_ nothrow_move_constructible< T >::value &&std::is_nothrow_move_ constructible< E >::value);	
函数功能	移动构造函数	
参数（IN）	other	右值Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	ara::core::Result< T, E >::~Result ();	
函数功能	析构函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	

注意事项	无

函数原型	static Result ara::core::Result< T, E >::FromValue (T const &t);	
函数功能	根据传入的T类型实例的左值返回一个含有T类型实例的Result实例	
参数（IN）	t	T类型左值
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result	构造出的Result
使用说明	无	
注意事项	• 线程安全	

函数原型	static Result ara::core::Result< T, E >::FromValue (T &&t);	
函数功能	根据传入的T类型实例的右值返回一个含有T类型实例的Result实例。	
参数（IN）	t	T类型右值
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result	构造出的Result
使用说明	无	
注意事项	• 线程安全	

函数原型	template <typename... Args> static Result ara::core::Result< T, E >::FromValue (Args &&... args);	
函数功能	使用T的构造函数入参构造含T类型数值的Result实例	
参数（IN）	args	T构造函数入参列表
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	Result	构造出的Result

使用说明	args…为T的构造函数入参。
	示例：
	struct fromValue {
	fromValue (int param1, int param2) {}
	};
	int test=1;
	int test1=1;
	Result<fromValue>
	resultFromValue=Result<fromValue>::FromValue(test,test1)	

注意事项	• 线程安全	

函数原型	static Result ara::core::Result< T, E >::FromError (E const &e);	
函数功能	根据传入的E类型左值返回一个含有E类型实例的Result实例	
参数（IN）	e	错误码
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result	包含E类型实例的Result实例

使用说明	示例：
	const ErrorCode errorcode(CoreErrc::kInvalidArgument, 12);
	auto result = Result<int>::FromError(errorcode);
	注意：该接口暂不支持不含默认构造函数的T类型Result<T,E>的构造，即
	Result<T>::FromError(errorcode); // 若T无默认构造函数，该语句将编译失败。
	此时需通过如下方式构造该Result：
	Result<T> result(T(…)); // 调用T构造函数先构造reuslt
	result.EmplaceError(CoreErrc::kInvalidArgument, 12); // 再调用EmplaceError将result转为Error类型	

注意事项	• 线程安全	

函数原型	static Result ara::core::Result< T, E >::FromError (E &&e);	
函数功能	根据传入的E类型右值返回一个含有E类型实例的Result实例	
参数（IN）	e	错误码
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	Result	包含ErrorCode的Result实例

使用说明	注意：该接口暂不支持不含默认构造函数的T类型Result<T,E>的构造，即
	Result<T>::FromError(errorcode); // 若T无默认构造函数，该语句将编译失败。
	此时需通过如下方式构造该Result：
	Result<T> result(T(…)); // 调用T构造函数先构造reuslt
	result.EmplaceError(CoreErrc::kInvalidArgument, 12); // 再调用EmplaceError将result转为Error类型	

注意事项	• 线程安全	

函数原型	template <typename... Args> static Result ara::core::Result< T, E >::FromError (Args &&... args);	
函数功能	使用ErrorCode的构造函数构造含ErrorCode的Result	
参数（IN）	args	ErrorCode构造函数入参
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result	包含ErrorCode的Result实例

使用说明	示例：
	auto resultFromError=Result<int>::FromError(CoreErrc::kInvalidArgument, 12);
	注意：该接口暂不支持不含默认构造函数的T类型Result<T,E>的构造，即
	Result<T>::FromError(errorcode); // 若T无默认构造函数，该语句将编译失败。
	此时需通过如下方式构造该Result：
	Result<T> result(T(…)); // 调用T构造函数先构造reuslt
	result.EmplaceError(CoreErrc::kInvalidArgument, 12); // 再调用EmplaceError将result转为Error类型	

注意事项	• 线程安全	

函数原型	Result& ara::core::Result< T, E >::operator= (Result const &other);	
函数功能	拷贝赋值	
参数（IN）	other	左值Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result&	与入参Result相同的Result
使用说明	无	
注意事项	无	

函数原型	Result& ara::core::Result< T, E >::operator= (Result &&other) noexcept(std::is_nothrow_move_constructible< T >::value &&std::is_ nothrow_move_assignable< T >::value &&std::is_nothrow_move_ constructible< E >::value &&std::is_nothrow_move_assignable< E >::value);	
函数功能	移动赋值	
参数（IN）	other	右值Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result&	与入参Result相同的Result
使用说明	无	
注意事项	无	

函数原型	template <typename... Args> void ara::core::Result< T, E >::EmplaceValue (Args &&... args);	
函数功能	使用T的构造函数入参对Result重新赋值	
参数（IN）	args	Value构造函数入参
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	None	-
使用说明	原Result若含T，则其内部值改为新值，原Result若含E，则Result改为含T的Result	
注意事项	• 线程安全	

函数原型	template <typename... Args> void ara::core::Result< T, E >::EmplaceError (Args &&... args);	
函数功能	使用ErrorCode的构造函数入参对Result重新赋值	
参数（IN）	args	ErrorCode构造函数入参
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	原Result若含T，则Result改为含T的Result，原Result若含E，则其错误码改为新错误码	
注意事项	• 线程安全	

函数原型	void ara::core::Result< T, E >::Swap (Result &other) noexcept(std::is_ nothrow_move_constructible< T >::value &&std::is_nothrow_move_ assignable< T >::value &&std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_move_assignable< E >::value);	
函数功能	与传入的另一个Result实例的内容进行交换	
参数（IN）	other	另一个Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	示例： Result<int> result1(1); Result<int> result2(2); result1.Swap(result2);	

注意事项	• 线程安全

函数原型	bool ara::core::Result< T, E >::HasValue () const noexcept;	
函数功能	判断Result实例是否含有值类型。如果包含则返回true，否则返回false。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	如果包含值类型则返回true，否则返回false。
使用说明	无	
注意事项	• 线程安全	

函数原型	explicit ara::core::Result< T, E >::operator bool () const noexcept;	
函数功能	bool类型转换符重载，判断Result实例是否含有值类型。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	如果包含值类型则返回true，否则返回false。
使用说明	示例： Result<int> result(1); bool b = bool(result); // b == true	
注意事项	无	

函数原型	T const& ara::core::Result< T, E >::operator * () const &;	
函数功能	获取该Result的T值	
参数（IN）	None	-
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	T const&	Result包含的T值
使用说明	若该Result不含T，含E，则导致Abort（规范中表示为未定义行为，实际实现为程序Abort）。 示例： Result<int> result(1); int a = *result; //a == 1;	
注意事项	无	

函数原型	T&& ara::core::Result< T, E >::operator * () &&;	
函数功能	获取该Result的T值	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&&	Result包含的T值
使用说明	若该Result不含T，含E，则导致Abort（规范中表示为未定义行为，实际实现为程序Abort）。	
注意事项	无	

函数原型	T const* ara::core::Result< T, E >::operator-> () const;	
函数功能	重载，返回指向该Result内T值的指针	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T const*	指向该Result内T值的指针

使用说明	若该Result不含T，含E，则导致Abort（规范中表示为未定义行为，实际实现为程序Abort）。
	示例：
	Class Test {
	void Play();
	};
	Result<Test> result;
	result->Play();	

注意事项	无

函数原型	T const& ara::core::Result< T, E >::Value () const &;	
函数功能	获取该Result实例所包含的T类型的值	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T const&	该Result实例所包含的T类型的值
使用说明	若该Result不含T，含E，则导致Abort（规范中表示为未定义行为，实际实现为程序Abort）。	
注意事项	• 线程安全	

函数原型	T&& ara::core::Result< T, E >::Value () &&;	
函数功能	获取该Result实例所包含的T类型的值	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T &&	该Result实例所包含的T类型的值
使用说明	若该Result不含T，含E，则导致Abort（规范中表示为未定义行为，实际实现为程序Abort）。	
注意事项	• 线程安全	

函数原型	E const& ara::core::Result< T, E >::Error () const &;	
函数功能	获取该Result实例所包含的ErrorCode	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	E const&	Result所含E值
使用说明	若该Result不含E，含T，则导致Abort（规范中表示为未定义行为，实际实现为程序Abort）。	
注意事项	• 线程安全	

函数原型	E&& ara::core::Result< T, E >::Error () &&;	
函数功能	获取该Result实例所包含的ErrorCode	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	E &&	Result所含E值
使用说明	若该Result不含E，含T，则导致Abort（规范中表示为未定义行为，实际实现为程序Abort）。	
注意事项	• 线程安全	

函数原型	template <typename U> T ara::core::Result< T, E >::ValueOr (U &&defaultValue) const &;	
函数功能	获取该Result所含T值，如果该Result实例存在值，则返回该Result实例所包含的值，否则返回传入的U类型的值（被强制转换为T类型返回）。	
参数（IN）	defaultValue	如果*this不包含value，则使用这个value
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T	返回value
使用说明	无	
注意事项	• 线程安全	

函数原型	template <typename U> T ara::core::Result< T, E >::ValueOr (U &&defaultValue) &&;

函数功能	获取该Result所含T值，如果该Result实例存在值，则返回该Result实例所包含的值，否则返回传入的U类型的值（被强制转换为T类型返回）	
参数（IN）	defaultValue	如果*this不包含value，则使用这个value
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T	返回value
使用说明	无	
注意事项	• 线程安全	

函数原型	template <typename G> E ara::core::Result< T, E >::ErrorOr (G &&defaultError) const;	
函数功能	获取该Result所含E值，如果该Result实例存在E值，则返回该Result实例所包含的E值，否则返回传入的G类型的值（被强制转换为E类型返回）	
参数（IN）	defaultError	若E不存在，函数将返回的值
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	E	实际E值或强转后的默认值
使用说明	若E不存在，G类型值将被强制转换为E类型返回，因此要求G类型具备强制转换为E类型的条件	
注意事项	• 线程安全	

函数原型	template <typename G> bool ara::core::Result< T, E >::CheckError (G &&error) const;	
函数功能	检查Result所含错误类型	
参数（IN）	error	被检测错误类型
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	检测结果

使用说明	如果该Result实例包含值则返回false；如果该Result实例所包含的错误实例和传入的G类型的错误实例一致，则返回true，否则返回false。 要求：G类型具备强转为E类型的条件
注意事项	• 线程安全

函数原型	T const& ara::core::Result< T, E >::ValueOrThrow () const & noexcept(false);	
函数功能	返回该Result的T值，若Result不含T，则根据E类型的错误抛出ara::core::Exception类型的异常	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T const&	T类型的值
使用说明	无	
注意事项	• 线程安全	

函数原型	T&& ara::core::Result< T, E >::ValueOrThrow () &&noexcept(false);	
函数功能	返回该Result的T值，若Result不含T，则根据E类型的错误抛出ara::core::Exception类型的异常	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&&	T类型的值
使用说明	无	
注意事项	• 线程安全	

函数原型	template <typename F> T ara::core::Result< T, E >::Resolve (F &&f) const;

函数功能	如果该Result实例包含值，则返回该值，否则返回传入函数的返回值。传入的函数的形参为E类型的错误。	
参数（IN）	f	F类型的函数指针
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T	返回对应的Value
使用说明	传入F类型应为：T f(E const&); 示例： int funcResolve(ErrorCode test){return 10;} Result<int> resultError(CoreErrc::kInvalidArgument, 10); resultError.Resolve(funcResolve);	
注意事项	● 线程安全	

函数原型	template <typename F> auto ara::core::Result< T, E >::Bind (F &&f) const -> SEE_BELOW;	
函数功能	对Result内T值，执行f操作，并使用f的返回值构造新的Result返回	
参数（IN）	f	可调用的F类型的函数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	SEE_BELOW	可被转换的数据类型的新的Result实例

使用说明	F函数类型可为以下两种：
	1. Result<XXX, E> f(T const&);
	2. XXX f(T const&)
	即F既可为返回Result<XXX>的函数，又可为返回直接返回XXX的类型。
	如果原Result不含T，含E，则Result<XXX,E>会被构造并返回，且原错误码被复制到新的Result里。
	注意：若T == void，则该函数不可用
	示例1：
	`std::string funcBind(int test1)`
	`{`
	`return "test";`
	`}`
	`Result<int> result(1);`
	`// bindResult == Result<std::string>("test")`
	`auto bindResult = result.Bind(funcBind);`
	示例2：
	`Result<std::string> funcBind2(int test1)`
	`{`
	`return std::string("test2");`
	`}`
	`Result<int> result(1);`
	`// bindResult == Result<std::string>("test2")`
	`auto bindResult = result.Bind(funcBind2)`

注意事项	● 线程安全

3.4.1 Result<void, E> 模版特化

此小节介绍Result的void值的特化类型。

接口类定义

类名	ara::core::Result< void, E >
命名空间	namespace ara::core
语法	template <typename E> class Result<void, E> final {...};
头文件	#include "ara/core/result.h"
说明	模板类Result的void值的特化类型

类内数据类型定义

类型名	定义	说明
value_type	void	该Result包含的数值类型

类型名	定义	说明
error_type	E（默认为ErrorCode，仅支持ErrorCode数据类型）	该Result包含的错误类型

成员函数定义

函数原型	Result () noexcept;	
函数功能	构造函数	
参数（IN）	None	None
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	构造出一个value是void的Result	
注意事项	无	

函数原型	explicit Result (E const &e);	
函数功能	构造函数，根据传入的E类型的左值构造一个新的Result实例	
参数（IN）	e	E类型左值
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	构造含错误码e的Result	
注意事项	无	

函数原型	explicit Result (E &&e);	
函数功能	构造函数，根据传入的E类型的右值构造一个新的Result实例	
参数（IN）	e	E类型右值
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	None	-
使用说明	构造含错误码e的Result	
注意事项	无	

函数原型	Result (Result const &other);	
函数功能	拷贝构造函数	
参数（IN）	other	左值Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	Result (Result &&other) noexcept (std::is_nothrow_move_constructible<E>::value);	
函数功能	移动构造函数	
参数（IN）	other	右值Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	~Result ();	
函数功能	析构函数	
参数（IN）	None	-
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	static Result FromValue ();	
函数功能	构造一个Value是void类型的Result实例	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result	构造出的Result
使用说明	无	
注意事项	• 线程安全	

函数原型	static Result FromError (E const &e);	
函数功能	根据传入的E类型左值返回一个含有E类型实例的Result实例	
参数（IN）	e	错误码
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result	包含E类型实例的Result实例
使用说明	无	
注意事项	• 线程安全	

函数原型	static Result FromError (E &&e);	
函数功能	根据传入的E类型右值返回一个含有E类型实例的Result实例	
参数（IN）	e	错误码
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	Result	包含ErrorCode的Result实例
使用说明	无	
注意事项	• 线程安全	

函数原型	template <typename... Args> static Result FromError (Args &&... args);	
函数功能	使用ErrorCode的构造函数构造含ErrorCode的Result	
参数（IN）	args	ErrorCode构造函数入参
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result	包含ErrorCode的Result实例
使用说明	无	
注意事项	• 线程安全	

函数原型	Result& operator= (Result const &other);	
函数功能	拷贝赋值	
参数（IN）	other	左值Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result&	与入参Result相同的Result
使用说明	无	
注意事项	无	

函数原型	Result& operator= (Result &&other) noexcept(std::is_nothrow_move_ constructible< E >::value &&std::is_nothrow_move_assignable< E >::value);
函数功能	移动赋值

参数（IN）	other	右值Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result&	与入参Result相同的Result
使用说明	无	
注意事项	• 无	

函数原型	template <typename... Args> void EmplaceValue (Args &&... args);	
函数功能	使用T的构造函数入参对Result重新赋值	
参数（IN）	args	Value构造函数入参
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程安全	

函数原型	template <typename... Args> void EmplaceError (Args &&... args);	
函数功能	使用ErrorCode的构造函数入参对Result重新赋值	
参数（IN）	args	ErrorCode构造函数入参
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程安全	

函数原型	void Swap (Result &other) noexcept(std::is_nothrow_move_constructible< E >::value &&std::is_nothrow_move_assignable< E >::value);	
函数功能	与传入的另一个Result实例的内容进行交换	
参数（IN）	other	另一个Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程不安全	

函数原型	bool HasValue () const noexcept;	
函数功能	判断Result实例是否含有值类型。如果包含则返回true，否则返回false。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	如果包含值类型则返回true，否则返回false。
使用说明	无	
注意事项	• 线程安全	

函数原型	explicit operator bool () const noexcept;	
函数功能	bool类型转换符重载，判断Result实例是否含有值类型。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	如果包含值类型则返回true，否则返回false
使用说明	无	

注意事项	无

函数原型	void operator * () const;	
函数功能	None，仅是为了generic programming	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该接口仅是为了支撑泛型编程，并且如果*this不包含值，该函数功能行为未定义。	
注意事项	无	

函数原型	void Value () const;	
函数功能	None，仅是为了generic programming	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程安全	

函数原型	E const& Error () const &;	
函数功能	获取该Result实例所包含的ErrorCode	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	E const&	Result所含E值

使用说明	无
注意事项	• 线程安全

函数原型	E&& Error () &&;	
函数功能	获取该Result实例所包含的ErrorCode	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	E &&	Result所含E值
使用说明	无	
注意事项	• 线程安全	

函数原型	template <typename U> void ValueOr (U &&defaultValue) const;	
函数功能	None	
参数（IN）	defaultValue	如果*this不包含value，则使用这个value
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程安全	

函数原型	template <typename G> E ErrorOr (G &&defaultError) const;	
函数功能	获取该Result所含E值，如果该Result实例存在E值，则返回该Result实例所包含的E值，否则返回传入的G类型的值（被强制转换为E类型返回）	
参数（IN）	defaultError	若E不存在，函数将返回的值

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	E	实际E值或强转后的默认值
使用说明	若E不存在，G类型值将被强制转换为E类型返回，因此要求G类型具备强制转换为E类型的条件	
注意事项	• 线程安全	

函数原型	template <typename G> bool CheckError (G &&error) const;	
函数功能	检查Result所含错误类型	
参数（IN）	error	被检测错误类型
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	检测结果
使用说明	如果该Result实例包含值则返回false；如果该Result实例所包含的错误实例和传入的G类型的错误实例一致，则返回true，否则返回false。 要求：G类型具备强转为E类型的条件。	
注意事项	• 线程安全	

函数原型	void ValueOrThrow () const noexcept(false);	
函数功能	返回该Result的T值，若Result不含T，则根据E类型的错误抛出ara::core::Exception类型的异常	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程安全	

函数原型	template <typename F> void Resolve (F &&f) const;	
函数功能	None	
参数（IN）	f	F类型的函数指针
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程安全	

3.4.2 全局函数定义

函数原型	template <typename T, typename E> bool operator== (Result< T, E > const &lhs, Result< T, E > const &rhs);	
函数功能	==操作符重载函数	
参数（IN）	lhs	用于比较的Result实例1
	rhs	用于比较的Result实例2
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	比较的结果
使用说明	比较两个Result实例是否相等。如果一个Result包含了Value，它不等于任何包含Error的Result。当且仅当两个Result的实例都包含同样的数据类型，比如都是ErrorCode或者都是Value，并且包含的该类型的值相等，这两个Result才相等。	
注意事项	无	

函数原型	template <typename T, typename E> bool operator!= (Result< T, E > const &lhs, Result< T, E > const &rhs);	
函数功能	!=操作符重载函数	
参数（IN）	lhs	用于比较的Result实例1

	rhs	用于比较的Result实例2
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	比较的结果
使用说明	比较两个Result实例是否不相等。如果一个Result包含了Value，它不等于任何包含Error的Result。当且仅当两个Result的实例都包含同样的数据类型，比如都是ErrorCode或者都是Value，并且包含的该类型的值相等，这两个Result才相等，其他场景均为不相等。	
注意事项	无	

函数原型	template <typename T, typename E> bool operator== (Result< T, E > const &lhs, T const &rhs);	
函数功能	==操作符重载函数	
参数（IN）	lhs	用于比较的Result实例
	rhs	用于比较的Value
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	比较的结果
使用说明	比较Result和Value是否相等。一个不包含Value的Result不等于任何Value。当且仅当一个包含Value的Result与待比较的Value有相同的数据类型，并且Result包含的Value等于待比较的Value，它们才相等。	
注意事项	无	

函数原型	template <typename T, typename E> bool operator== (T const &lhs, Result< T, E > const &rhs);	
函数功能	==操作符重载函数	
参数（IN）	lhs	用于比较的Value
	rhs	用于比较的Result实例
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	bool	比较的结果
使用说明	比较Result和Value是否相等。一个不包含Value的Result不等于任何Value。当且仅当一个包含Value的Result与待比较的Value有相同的数据类型，并且Result包含的Value等于待比较的Value，它们才相等。	
注意事项	无	

函数原型	template <typename T, typename E> bool operator!= (Result< T, E > const &lhs, T const &rhs);	
函数功能	!=操作符重载函数	
参数（IN）	lhs	用于比较的Result实例
	rhs	用于比较的Value
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	比较的结果
使用说明	比较Result和Value是否不相等。一个不包含Value的Result不等于任何Value。当且仅当一个包含Value的Result与待比较的Value有相同的数据类型，并且Result包含的Value等于待比较的Value，它们才相等，其他场景均为不相等。	
注意事项	无	

函数原型	template <typename T, typename E> bool operator!= (T const &lhs, Result< T, E > const &rhs);	
函数功能	!=操作符重载函数	
参数（IN）	lhs	用于比较的Value
	rhs	用于比较的Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	比较的结果

使用说明	比较Result和Value是否不相等。一个不包含Value的Result不等于任何Value。当且仅当一个包含Value的Result与待比较的Value有相同的数据类型，并且Result包含的Value等于待比较的Value，它们才相等，其他场景均为不相等。
注意事项	无

函数原型	template <typename T, typename E> bool operator== (Result< T, E > const &lhs, E const &rhs);	
函数功能	==操作符重载函数	
参数（IN）	lhs	用于比较的Result实例
	rhs	用于比较的Error
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	比较的结果
使用说明	比较Result和Error是否相等。一个不包含Error的Result不等于任何Error。当且仅当一个包含Error的Result与待比较的Error有相同的数据类型，并且Result包含的Error的值等于待比较的Error的值，它们才相等。	
注意事项	无	

函数原型	template <typename T, typename E> bool operator== (E const &lhs, Result< T, E > const &rhs);	
函数功能	==操作符重载函数	
参数（IN）	lhs	用于比较的Error
	rhs	用于比较的Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	比较的结果
使用说明	比较Result和Error是否相等。一个不包含Error的Result不等于任何Error。当且仅当一个包含Error的Result与待比较的Error有相同的数据类型，并且Result包含的Error的值等于待比较的Error的值，它们才相等。	

注意事项	无

函数原型	template <typename T, typename E> bool operator!= (Result< T, E > const &lhs, E const &rhs);	
函数功能	!=操作符重载函数	
参数（IN）	lhs	用于比较的Result实例
	rhs	用于比较的Error
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	比较的结果
使用说明	比较Result和Error是否不相等。一个不包含Error的Result不等于任何Error。当且仅当一个包含Error的Result与待比较的Error有相同的数据类型，并且Result包含的Error的值等于待比较的Error的值，它们才相等，其他场景均为不相等。	
注意事项	无	

函数原型	template <typename T, typename E> bool operator!= (E const &lhs, Result< T, E > const &rhs);	
函数功能	!=操作符重载函数	
参数（IN）	lhs	用于比较的Error
	rhs	用于比较的Result实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	比较的结果
使用说明	比较Result和Error是否不相等。一个不包含Error的Result不等于任何Error。当且仅当一个包含Error的Result与待比较的Error有相同的数据类型，并且Result包含的Error的值等于待比较的Error的值，它们才相等，其他场景均为不相等。	
注意事项	无	

函数原型	template <typename T, typename E> void swap (Result< T, E > &lhs, Result< T, E > &rhs) noexcept(noexcept(lhs.Swap(rhs)));	
函数功能	交换Promise实例内容	
参数（IN）	lhs	Result实例1
	rhs	Result实例2
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	交换两个Result的实例	
注意事项	• 线程不安全	

3.5 CoreErrorDomain

CoreErrorDomain继承自ara::core::ErrorDomain，其定义了CORE功能模块的错误类型。

类名	ara::core::CoreErrorDomain
命名空间	namespace ara::core
语法	class CoreErrorDomain final : public ErrorDomain {...};
头文件	#include "ara/core/core_error_domain.h"
说明	CORE类型错误对应错误域 CORE错误域ID为0x8000'0000'0000'0014

类内数据类型定义

类型名	定义	说明
Errc	CoreErrc	错误码的枚举值
Exception	CoreException	异常的类型

成员函数定义

函数原型	constexpr CoreErrorDomain () noexcept;

函数功能	构造函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	char const* Name () const noexcept override;	
函数功能	返回CoreErrorDomain文字表示	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	char const*	返回“Core”
使用说明	无	
注意事项	• 线程安全	

函数原型	char const* Message (ErrorDomain::CodeType errorCode) const noexcept override;	
函数功能	返回错误码对应文字信息	
参数（IN）	errorCode	Core类型错误码
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	char const*	错误码对应错误类型
使用说明	若错误码非core已定义错误码，则程序Abort	
注意事项	• 线程安全	

函数原型	void ThrowAsException (ErrorCode const &errorCode) const override;	
函数功能	抛出错误码对应CoreException	
参数（IN）	errorCode	Core错误码
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程安全	

3.5.1 CoreErrorCode

CoreError错误码枚举定义

类名	ara::core::CoreErrc
命名空间	namespace ara::core
语法	enum class CoreErrc : ErrorDomain::CodeType {...};
头文件	#include "ara/core/core_error_domain.h"

说明	该类型定义了CORE模块所有错误类型，含以下值：
	kInvalidArgument= 22 // 函数收到无效入参
	kInvalidMetaModelShortname= 137 // string为无效模型元素的shortname
	kInvalidMetaModelPath= 138 // 错误模型元素路径

3.5.2 CoreException

接口类定义

类名	ara::core::CoreException
命名空间	namespace ara::core
语法	class CoreException : public Exception {...};
头文件	#include "ara/core/core_error_domain.h"
说明	CORE类型错误对应异常

成员函数定义

函数原型	explicit CoreException (ErrorCode err) noexcept;	
函数功能	构造函数	
参数（IN）	err	ErrorCode
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

3.5.3 GetCoreErrorDomain

函数原型	constexpr ErrorDomain const& GetCoreErrorDomain () noexcept;	
函数功能	返回全局CoreErrorDomain的引用	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ErrorDomain const&	返回全局CoreErrorDomain的引用
使用说明	无	
注意事项	• 线程安全	

3.5.4 MakeErrorCode

函数原型	constexpr ErrorCode MakeErrorCode (CoreErrc code, ErrorDomain::SupportDataType data) noexcept;	
函数功能	创建CoreErrorDomain内的ErrorCode	
参数（IN）	code	CoreErrc错误码
	data	补充数据类型

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ErrorCode	一个新的ErrorCode的实例
使用说明	该函数通常为ErrorCode构造函数内部使用，用户不直接使用。	
注意事项	• 线程安全	

3.6 Future and Promise

3.6.1 future_errc

类名	ara::core::future_errc
命名空间	namespace ara::core
语法	enum class future_errc : int32_t {...};
头文件	#include "ara/core/future_error_domain.h"

说明	该类型定义了future模块所有错误类型，含以下值：
	broken_promise = 101, // 异步任务抛弃其共享状态
	future_already_retrieved = 102, // 共享状态的内容已通过ara::core::Future访问
	promise_already_satisfied = 103, // 试图两次存储值于共享状态
	no_state = 104, // 试图访问无关联共享状态的std::promise或std::future
	以上定义同std::future_errc

3.6.2 FutureException

接口类定义

类名	ara::core::FutureException
命名空间	namespace ara::core
语法	class FutureException : public Exception {...};
头文件	#include "ara/core/future_error_domain.h"
说明	Future、Promise抛出异常类型

成员函数定义

函数原型	explicit FutureException (ErrorCode err) noexcept;	
函数功能	构造函数，根据特定的ErrorCode实例构造FutureException实例。	
参数（IN）	err	ErrorCode实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

3.6.3 FutureErrorDomain

接口类定义

类名	ara::core::FutureErrorDomain
命名空间	namespace ara::core
语法	class FutureErrorDomain final : public ErrorDomain {...};
头文件	#include "ara/core/future_error_domain.h "
说明	Future、Promise类型错误对应错误域 CORE错误域ID为0x8000'0000'0000'0013

类内数据类型定义

类型名	定义	说明
Errc	future_errc	错误码枚举值
Exception	FutureException	异常的类型

成员函数定义

函数原型	constexpr FutureErrorDomain () noexcept;	
函数功能	默认构造函数	
参数（IN）	None	-

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	char const* Name () const noexcept override;	
函数功能	返回FutureErrorDomain文字表示	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	char const*	返回“Future”
使用说明	无	
注意事项	• 线程安全	

函数原型	char const* Message (ErrorDomain::CodeType errorCode) const noexcept override;	
函数功能	返回错误码对应文字信息	
参数（IN）	errorCode	Future、Promise类型错误码
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	char const*	错误码对应错误类型文字表示
使用说明	若错误码非Future、Promise已定义错误码，则程序Abort	
注意事项	• 线程安全	

函数原型	void ThrowAsException (ErrorCode const &errorCode) const override;

函数功能	抛出错误码对应FutureException	
参数（IN）	errorCode	一个ErrorCode的实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	• 线程安全	

3.6.4 GetFutureErrorDomain

函数原型	constexpr ErrorDomain const& GetFutureErrorDomain () noexcept;	
函数功能	返回全局CoreErrorDomain的引用	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ErrorDomain const&	返回全局FutureErrorDomain的引用
使用说明	无	
注意事项	• 线程安全	

3.6.5 MakeErrorCode

函数原型	constexpr ErrorCode MakeErrorCode (future_errc code, ErrorDomain::SupportDataType data) noexcept;	
函数功能	创建FutureErrorDomain内的ErrorCode	
参数（IN）	code	future_errc错误码
	data	补充数据类型
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	ErrorCode	一个新的ErrorCode的实例
使用说明	无	
注意事项	• 线程安全	

3.6.6 future_status

枚举类型名	ara::core::future_status
命名空间	namespace ara::core
语法	enum class future_status : uint8_t {...};
头文件	#include "ara/core/future.h"
枚举值	ready // 共享状态已准备好 timeout // 共享状态在超时前仍未准备好
说明	该枚举值表示Future的当前状态，为wait_for和wait_until的返回值 状态定义与std::future_status相同，但无对应std::future_status::deferred定义

3.6.7 Future

接口类定义

类名	ara::core::Future
命名空间	namespace ara::core
语法	template <typename T, typename E = ErrorCode> class Future final {...};
头文件	#include "ara/core/future.h"
模板参数	typename T // future所含值类型，支持void类型 typename E = ErrorCode // 错误类型
说明	Future类用于获取异步调用结果

成员函数定义

函数原型	ara::core::Future< T, E >::Future () noexcept=default;
函数功能	默认构造函数

参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	ara::core::Future< T, E >::Future (Future const &)=delete;	
函数功能	禁用拷贝构造函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	禁用拷贝构造函数	
注意事项	无	

函数原型	ara::core::Future< T, E >::Future (Future &&other) noexcept;	
函数功能	移动构造函数	
参数（IN）	other	右值Future实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	支持移动构造	
注意事项	无	

函数原型	ara::core::Future< T, E >::~Future ();
函数功能	析构函数

参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	析构函数	
注意事项	无	

函数原型	Future& ara::core::Future< T, E >::operator= (Future &other) = delete;	
函数功能	禁用拷贝赋值	
参数（IN）	other	Future实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Future&	赋值后Future实例
使用说明	禁用拷贝赋值	
注意事项	无	

函数原型	Future& ara::core::Future< T, E >::operator= (Future &&other) noexcept;	
函数功能	移动赋值	
参数（IN）	other	右值Future实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Future&	赋值后Future实例
使用说明	支持移动赋值	
注意事项	无	

函数原型	T ara::core::Future< T, E >::get ();

函数功能	获取异步调用结果	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T	Future所含T类型值
使用说明	此调用将被阻塞，直到共享状态准备好	
注意事项	• 该函数行为与std::future.get()函数行为一致，可能会抛出异常	

函数原型	Result<T, E> ara::core::Future< T, E >::GetResult () noexcept;	
函数功能	获取异步调用结果	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result<T, E>	异步调用结果
使用说明	与get()类似，此调用将被阻塞，直到该值或错误可用为止。但是，此调用永远不会引发异常，即该函数将会捕获get()所抛出的异常，并以含E Result方式返回。	
注意事项	无	

函数原型	bool ara::core::Future< T, E >::valid () const noexcept;	
函数功能	检查Future是否有效，即是否含共享状态	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	Future可用返回true，否则返回false
使用说明	该函数使用方法同std::future::valid()	
注意事项	无	

函数原型	void ara::core::Future< T, E >::wait () const;	
函数功能	等待值或错误可用	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数使用方法同std::future函数	
注意事项	无	

函数原型	template <typename Rep, typename Period> future_status ara::core::Future< T, E >::wait_for (std::chrono::duration< Rep, Period > const &timeoutDuration) const;	
函数功能	等待给定时间，期间若值或错误可用则直接返回	
参数（IN）	timeoutDuration	最大等待时间
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	future_status	超时返回timeout，否则返回ready
使用说明	该函数使用方法同std::future函数	
注意事项	无	

函数原型	template <typename Clock, typename Duration> future_status ara::core::Future< T, E >::wait_until (std::chrono::time_point< Clock, Duration > const &deadline) const;	
函数功能	等待至给定时间，期间若值或错误可用则直接返回	
参数（IN）	deadline	最大等待时间节点
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	future_status	超时返回timeout，否则返回ready
使用说明	该函数使用方法同std::future函数	
注意事项	无	

函数原型	template <typename F> auto ara::core::Future< T, E >::then (F &&func) -> Future< SEE_BELOW>;	
函数功能	注册回调函数，当Future准备好时，回调将会被调用	
参数（IN）	func	回调函数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Future<SEE_BELOW>	新的Future实例，具体类型见使用说明
使用说明	注册回调函数，当Future准备好时，回调将会被调用。 当回调被调用时，回调函数中的get()、GetResult()将不会表现为阻塞。 func将会被在then()调用处、Promise::set_value()调用处、Promise::SetError()调用处、或其他地方被调用。 返回值类型取决于func返回值。 func的入参应为Future<T, E>类型。使U表示func返回值类型，若U为Future<T2, E2>，则then返回值类型为Future<T2, E2>，若U为Result<T2, E2>，则then返回值类型为Future<T2, E2>，其他情况，then返回值为Future<U, E>	
注意事项	无	

函数原型	bool ara::core::Future< T, E >::is_ready () const;	
函数功能	查询是否异步调用已结束	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	如果Future含值或error则返回true，否则返回false

使用说明	如果该函数返回true，则get()，GetResult()，wait_for(…)，wait_until(…)不会阻塞
注意事项	无

3.6.7.1 Future<void, E> 模版特化

此小节介绍Future的void值的特化类型。

接口类定义

类名	ara::core::Future< void, E >
命名空间	namespace ara::core
语法	template <typename E> class Future< void, E > final {...};
头文件	#include "ara/core/future.h"
模板参数	typename E = ErrorCode // 错误类型
说明	模板类Future的void值的特化类型

成员函数定义

函数原型	Future () noexcept;	
函数功能	默认构造函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	Future (Future const & other)=delete;	
函数功能	禁用拷贝构造函数	
参数（IN）	None	-

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	禁用拷贝构造函数	
注意事项	无	

函数原型	Future (Future &&other) noexcept;	
函数功能	移动构造函数	
参数（IN）	other	右值Future实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	支持移动构造	
注意事项	无	

函数原型	~Future ();	
函数功能	析构函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	析构函数	
注意事项	无	

函数原型	Future& operator= (Future const &other) = delete;	
函数功能	禁用拷贝赋值	
参数（IN）	other	Future实例

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Future&	赋值后Future实例
使用说明	禁用拷贝赋值	
注意事项	无	

函数原型	Future& operator= (Future &&other) noexcept;	
函数功能	移动赋值	
参数（IN）	other	右值Future实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Future&	赋值后Future实例
使用说明	支持移动赋值	
注意事项	无	

函数原型	T get ();	
函数功能	获取异步调用结果	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T	Future所含T类型值
使用说明	此调用将被阻塞，直到共享状态准备好	
注意事项	• 该函数行为与std::future.get()函数行为一致，可能会抛出异常	

函数原型	Result<void, E> GetResult () noexcept;	
函数功能	获取异步调用结果	
参数（IN）	None	-

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result<void, E>	异步调用结果
使用说明	与get()类似，此调用将被阻塞，直到该值或错误可用为止。但是，此调用永远不会引发异常，即该函数将会捕获get()所抛出的异常，并以含E Result方式返回。	
注意事项	无	

函数原型	bool valid () const noexcept;	
函数功能	检查Future是否有效，即是否含共享状态	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	Future可用返回true，否则返回false
使用说明	该函数使用方法同std::future::valid()	
注意事项	无	

函数原型	void wait () const;	
函数功能	等待值或错误可用	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数使用方法同std::future函数	
注意事项	无	

函数原型	template <typename Rep, typename Period> future_status wait_for (std::chrono::duration< Rep, Period > const &timeoutDuration) const;	
函数功能	等待给定时间，期间若值或错误可用则直接返回	
参数（IN）	timeoutDuration	最大等待时间
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	future_status	超时返回timeout，否则返回ready
使用说明	该函数使用方法同std::future函数	
注意事项	无	

函数原型	template <typename Clock, typename Duration> future_status wait_until (std::chrono::time_point< Clock, Duration > const &deadline) const;	
函数功能	等待至给定时间，期间若值或错误可用则直接返回	
参数（IN）	deadline	最大等待时间节点
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	future_status	超时返回timeout，否则返回ready
使用说明	该函数使用方法同std::future函数	
注意事项	无	

函数原型	template <typename F> auto then (F &&func) -> Future< SEE_BELOW>;	
函数功能	注册回调函数，当Future准备好时，回调将会被调用	
参数（IN）	func	回调函数
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	Future<SEE_BELOW>	新的Future实例，具体类型见使用说明
使用说明	注册回调函数，当Future准备好时，回调将会被调用。 当回调被调用时，回调函数中的get()、GetResult()将不会表现为阻塞。 func将会被在then()调用处、Promise::set_value()调用处、Promise::SetError()调用处、或其他地方被调用。 返回值类型取决于func返回值。 func的入参应为Future<T, E>类型。使用U表示func返回值类型，若U为Future<T2, E2>，则then返回值类型为Future<T2, E2>，若U为Result<T2, E2>，则then返回值类型为Future<T2, E2>，其他情况，then返回值为Future<U, E>	
注意事项	无	

函数原型	bool is_ready () const;	
函数功能	查询是否异步调用已结束	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	如果Future含值或error则返回true，否则返回false
使用说明	如果该函数返回true，则get()，GetResult()，wait_for(…)，wait_until(…)不会阻塞	
注意事项	无	

3.6.8 Promise

接口类定义

类名	ara::core::Promise
命名空间	namespace ara::core
语法	template <typename T, typename E = ErrorCode> class Promise {...};
头文件	#include "ara/core/promise.h"

模板参数	typename T // Promise所含值类型，支持void类型（Promise的void特化类是final类） typename E = ErrorCode // 错误类型
说明	Promise类用于设置异步调用结果

成员函数定义

函数原型	ara::core::Promise< T, E >::Promise ();	
函数功能	默认构造函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	ara::core::Promise< T, E >::Promise (Promise &&other) noexcept;	
函数功能	移动构造函数	
参数（IN）	other	右值promise
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	支持移动构造	
注意事项	无	

函数原型	ara::core::Promise< T, E >::Promise (Promise const &)=delete;	
函数功能	禁用拷贝构造函数	
参数（IN）	None	-
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	None	-
使用说明	禁用拷贝构造函数	
注意事项	无	

函数原型	ara::core::Promise< T, E >::~Promise ();	
函数功能	析构函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	Promise& ara::core::Promise< T, E >::operator= (Promise &&other) noexcept;	
函数功能	移动赋值	
参数（IN）	other	右值promise
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Promise&	移动后promise
使用说明	支持移动赋值	
注意事项	无	

函数原型	Promise& ara::core::Promise< T, E >::operator= (Promise const&)=delete;	
函数功能	禁用拷贝赋值	
参数（IN）	None	-

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	禁用拷贝赋值	
注意事项	无	

函数原型	void ara::core::Promise< T, E >::swap (Promise &other) noexcept;	
函数功能	交换Promise实例内容	
参数（IN）	other	Promise实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数使用方法同std::promise函数	
注意事项	• 线程安全	

函数原型	Future<T, E> ara::core::Promise< T, E >::get_future ();	
函数功能	返回关联future	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Future<T, E>	返回一个future
使用说明	由于不允许一个Promise对应多个Future，该函数仅能调用一次	
注意事项	• 该函数使用方法同std::promise函数	

函数原型	void ara::core::Promise< T, E >::set_value (T const &value);	
函数功能	存储value到共享状态，并令状态就绪	
参数（IN）	value	待存储共享状态value

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	存储value到共享状态，并令状态就绪；	
注意事项	• 该函数使用方法同std::promise函数，可能会抛出异常	

函数原型	void ara::core::Promise< T, E >::set_value (T &&value);	
函数功能	存储value到共享状态，并令状态就绪	
参数（IN）	value	待存储共享状态value
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	存储value到共享状态，并令状态就绪；	
注意事项	• 该函数使用方法同std::promise函数，可能会抛出异常	

函数原型	void ara::core::Promise< T, E >::SetError (E &&error);	
函数功能	存储error到共享状态，并令状态就绪	
参数（IN）	error	待存储共享状态error
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	存储error到共享状态，并令状态就绪	
注意事项	• 该函数的行为与set_value相似，可能会抛出异常	

函数原型	void ara::core::Promise< T, E >::SetError (E const &error);	
函数功能	存储error到共享状态，并令状态就绪	
参数（IN）	error	待存储共享状态error

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	存储error到共享状态，并令状态就绪	
注意事项	• 该函数的行为与set_value相似，可能会抛出异常	

3.6.8.1 Promise<void, E> 模版特化

此小节介绍Promise的void值的特化类型。

接口类定义

类名	ara::core::Promise< void, E >
命名空间	namespace ara::core
语法	template <typename E> class Promise<void, E> final{...};
头文件	#include "ara/core/promise.h"
模板参数	typename E // 错误类型
说明	模板类Promise的void值的特化类型

成员函数定义

函数原型	Promise ();	
函数功能	默认构造函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	Promise (Promise &&other) noexcept;

函数功能	移动构造函数	
参数（IN）	other	右值promise
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	支持移动构造	
注意事项	无	

函数原型	Promise (Promise const &)=delete;	
函数功能	禁用拷贝构造函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	禁用拷贝构造函数	
注意事项	无	

函数原型	~Promise ();	
函数功能	析构函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	Promise& operator= (Promise &&other) noexcept;

函数功能	移动赋值	
参数（IN）	other	右值promise
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Promise&	移动后promise
使用说明	支持移动赋值	
注意事项	无	

函数原型	Promise& operator= (Promise const&)=delete;	
函数功能	禁用拷贝赋值	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	禁用拷贝赋值	
注意事项	无	

函数原型	void swap (Promise &other) noexcept;	
函数功能	交换Promise实例内容	
参数（IN）	other	Promise实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数使用方法同std::promise函数	
注意事项	• 线程安全	

函数原型	Future<void, E> get_future ();

函数功能	返回关联future	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Future<void, E>	返回一个future
使用说明	由于不允许一个Promise对应多个Future，该函数仅能调用一次	
注意事项	• 该函数使用方法同std::promise函数相同	

函数原型	void set_value ();	
函数功能	设置共享状态，并令状态就绪	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	此函数仅设置共享状态，并令状态就绪	
注意事项	• 该函数使用方法同std::promise函数，可能会抛出异常	

函数原型	void SetError (E &&error);	
函数功能	存储error到共享状态，并令状态就绪	
参数（IN）	error	待存储共享状态error
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	存储error到共享状态，并令状态就绪	
注意事项	• 该函数的行为与set_value相似，可能会抛出异常	

函数原型	void SetError (E const &error);

函数功能	存储error到共享状态，并令状态就绪	
参数（IN）	error	待存储共享状态error
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	存储error到共享状态，并令状态就绪	
注意事项	• 该函数的行为与set_value相似，可能会抛出异常	

3.7 Array

接口类定义

类名	Array
命名空间	namespace ara::core
语法	template <typename T, std::size_t N> class Array { ... };
头文件	#include "ara/core/array.h"
说明	此类的所有成员和支持的结构（如全局关系运算符）应遵循std::array。

成员函数定义

函数原型	template <typename T, std::size_t N> void swap(Array<T, N>& lhs, Array<T, N>& rhs);	
函数功能	交换函数	
参数（IN）	None	-
参数（INOUT）	lhs	要交换内容的容器1
	rhs	要交换内容的容器2
参数（OUT）	None	-
返回值	None	-
使用说明	交换lhs和rhs的内容	
注意事项	• 线程不安全	

3.8 Vector

接口类定义

类名	Vector
命名空间	namespace ara::core
语法	template <typename T, typename Allocator = /* implementation-defined */> class Vector { ... };
头文件	#include "ara/core/vector.h"
说明	此类的所有成员和支持的结构（如全局关系运算符）应遵循std::vector，但从Allocator是由用户定义的。 但是，由于规范对于Allocator未定义，实际使用std::allocator<T>。

全局函数定义

函数原型	template <typename T, typename Allocator> bool operator==(Vector<T, Allocator> const& lhs, Vector<T, Allocator> const& rhs);	
函数功能	比较函数	
参数（IN）	None	-
参数（INOUT）	lhs	要比较内容的容器1
	rhs	要比较内容的容器2
参数（OUT）	None	-
返回值	bool	true：两个容器里的内容相等 false：两个容器里的内容不相等
使用说明	检查lhs与rhs的内容是否相等，即它们是否拥有相同数量的元素且lhs中每个元素与rhs的同位置元素比较相等。	
注意事项	无	

函数原型	template <typename T, typename Allocator> bool operator!=(Vector<T, Allocator> const& lhs, Vector<T, Allocator> const& rhs);

函数功能	比较函数	
参数（IN）	None	-
参数（INOUT）	lhs	要比较内容的容器1
	rhs	要比较内容的容器2
参数（OUT）	None	-
返回值	bool	true：两个容器里的内容不相等 false：两个容器里的内容相等
使用说明	检查lhs与rhs的内容是否不相等，即它们是否拥有不同数量的元素或lhs中某个元素与rhs的同位置元素不相等。	
注意事项	无	

函数原型	template <typename T, typename Allocator> bool operator<(Vector<T, Allocator> const& lhs, Vector<T, Allocator> const& rhs);	
函数功能	比较函数	
参数（IN）	None	-
参数（INOUT）	lhs	要比较内容的容器1
	rhs	要比较内容的容器2
参数（OUT）	None	-
返回值	bool	true: lhs < rhs false: lhs >= rhs
使用说明	检查lhs是否小于rhs。按字典序比较lhs与rhs的内容。	
注意事项	无	

函数原型	template <typename T, typename Allocator> bool operator<=(Vector<T, Allocator> const& lhs, Vector<T, Allocator> const& rhs);	
函数功能	比较函数	
参数（IN）	None	-
参数（INOUT）	lhs	要比较内容的容器1
	rhs	要比较内容的容器2

参数（OUT）	None	-
返回值	bool	true: lhs < =rhs false: lhs > rhs
使用说明	检查lhs是否小于等于rhs。按字典序比较lhs与rhs的内容。	
注意事项	无	

函数原型	template <typename T, typename Allocator> bool operator>(Vector<T, Allocator> const& lhs, Vector<T, Allocator> const& rhs);	
函数功能	比较函数	
参数（IN）	None	-
参数（INOUT）	lhs	要比较内容的容器1
	rhs	要比较内容的容器2
参数（OUT）	None	-
返回值	bool	true: lhs >rhs false: lhs <=rhs
使用说明	检查lhs是否大于rhs。按字典序比较lhs与rhs的内容。	
注意事项	无	

函数原型	template <typename T, typename Allocator> bool operator>=(Vector<T, Allocator> const& lhs, Vector<T, Allocator> const& rhs);	
函数功能	比较函数	
参数（IN）	None	-
参数（INOUT）	lhs	要比较内容的容器1
	rhs	要比较内容的容器2
参数（OUT）	None	-
返回值	bool	true: lhs >=rhs false: lhs <=rhs
使用说明	检查lhs是否大于等于rhs。按字典序比较lhs与rhs的内容。	
注意事项	无	

函数原型	template <typename T, typename Allocator> void swap (Vector<T, Allocator> const& lhs, Vector<T, Allocator> const& rhs);	
函数功能	交换函数	
参数（IN）	None	-
参数（INOUT）	lhs	要交换内容的容器1
	rhs	要交换内容的容器2
参数（OUT）	None	-
返回值	None	-
使用说明	该函数交换lhs和rhs的状态	
注意事项	• 线程不安全	

3.9 Map

接口类定义

类名	Map
命名空间	namespace ara::core
语法	template <typename K, typename V, typename C = std::less<K>, typename Allocator = /* implementation-defined */ > class Map { ... };
头文件	#include "ara/core/map.h"
说明	此类的所有成员和支持的结构（如全局关系运算符）应遵循std::map，但从Allocator是由用户定义的。 但是，由于规范对于Allocator未定义，实际使用std::pmr::polymorphic_allocator<std::pair<const Key,T>>。

全局函数定义

函数原型	template <typename K, typename V, typename C, typename Allocator> void swap(Map<K, V, C, Allocator>& lhs, Map<K, V, C, Allocator>& rhs);

函数功能	交换函数	
参数（IN）	None	-
参数（INOUT）	lhs	要交换内容的容器1
	rhs	要交换内容的容器2
参数（OUT）	None	-
返回值	None	-
使用说明	交换lhs和rhs的状态。	
注意事项	• 线程不安全	

3.10 StringView

接口类定义

类名	StringView
命名空间	namespace ara::core
语法	class StringView { ... };
头文件	#include "ara/core/string_view.h"
说明	该类型构建一个字符序列的只读视图，该字符序列的生命周期由使用的对象负责保证。 此类的所有成员和支持的结构（如全局关系运算符）应遵循C++17的std::string_view，但从不使用constexpr声明非const成员函数。

3.11 String

接口类定义

类名	ara::core:: BasicString
命名空间	namespace ara::core
语法	template <typename Allocator = /* implementation-defined */> class BasicString { ... };
头文件	#include "ara/core/string.h"
说明	描述了AUTOSAR字符串类型，行为类似std::string CharT = char, Traits = std::char_traits<char>。 Allocator规范未定义，实际实现为std::allocator<char>。

类名	ara::core::String
命名空间	namespace ara::core
语法	using String = BasicString<>;
头文件	#include "ara/core/string.h"
说明	描述了AUTOSAR字符串类型。 CharT = char, Traits = std::char_traits<char>。

成员函数定义

函数原型	operator StringView() const noexcept;	
函数功能	隐式把BasicString转换成StringView	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该操作符用于隐式转换成StringView类型	
注意事项	无	

函数原型	explicit BasicString(StringView sv);	
函数功能	构造函数	
参数（IN）	sv	初始化string所用的对象
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	构造函数	
注意事项	无	

函数原型	template <typename T> BasicString(T const& t, size_type pos, size_type n);

函数功能	构造函数	
参数（IN）	t	初始化string所用的对象
	pos	要包含的首字符位置
	n	以pos开始的字符数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	构造函数	
注意事项	无	

函数原型	BasicString& operator=(StringView sv);	
函数功能	赋值操作符	
参数（IN）	sv	用于赋值的内容
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	BasicString&	当前的内容，即*this
使用说明	以sv的副本替换内容	
注意事项	无	

函数原型	BasicString& assign(StringView sv);	
函数功能	赋值函数	
参数（IN）	sv	用于赋值的内容
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	BasicString&	当前的内容，即*this
使用说明	通过StringView给String赋值	
注意事项	无	

函数原型	template <typename T> BasicString& assign(T const& t, size_type pos, size_type n = npos);	
函数功能	赋值函数	
参数（IN）	t	用于赋值的内容
	pos	要取的字符下标
	n	以pos开始的字符数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	BasicString&	当前的内容，即*this
使用说明	以来自t的子视图[pos，pos+n)的字符替换内容	
注意事项	无	

函数原型	BasicString& operator+=(StringView sv);	
函数功能	+=操作符	
参数（IN）	sv	要附加的内容
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	BasicString&	当前字符串拼接后的内容，即*this
使用说明	后附字符到结尾	
注意事项	无	

函数原型	BasicString& append(StringView sv);	
函数功能	拼接函数	
参数（IN）	sv	要附加的内容
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	BasicString&	当前字符串拼接后的内容，即*this
使用说明	后附额外字符到字符串	
注意事项	无	

函数原型	template <typename T> BasicString& append(T const& t, size_type pos, size_type n = npos);	
函数功能	拼接函数	
参数（IN）	t	要附加的内容
	pos	要后附的首个字符下标
	n	要后附的字符数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	BasicString&	当前字符串拼接后的内容，即*this
使用说明	后附来自t的子视图[pos，pos+n)的字符	
注意事项	无	

函数原型	BasicString& insert(size_type pos, StringView sv);	
函数功能	插入函数	
参数（IN）	pos	插入的位置
	sv	要插入的内容
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	BasicString&	当前字符串插入后的内容，即*this
使用说明	在位置pos插入sv	
注意事项	无	

函数原型	template <typename T> BasicString& insert(size_type pos1, T const& t, size_type pos2, size_type n = npos);	
函数功能	插入函数	
参数（IN）	pos1	插入内容到的位置
	t	要插入的字符来源对象
	pos2	待插入字符子串中首字符位置
	n	要插入的字符数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	BasicString&	当前字符串插入后的内容，即*this
使用说明	可用于插入任何可隐式转换成StringView的类型	
注意事项	无	

函数原型	BasicString& replace(size_type pos1, size_type n1, StringView sv);	
函数功能	替换函数	
参数（IN）	pos1	将被替换的子串起始位置
	n1	将被替换的子串长度
	sv	用于替换的StringView
参数（IN-OUT）	None	-
参数（OUT）	None	-
返回值	BasicString&	当前字符串替换后的内容，即*this
使用说明	可用于替换StringView类型的子序列	
注意事项	无	

函数原型	template <typename T> BasicString& replace(size_type pos1, size_type n1, T const& t, size_type pos2, size_type n2 = npos);

函数功能	替换函数	
参数（IN）	pos1	将被替换的子串起始位置
	n1	将被替换的子串长度
	t	拥有用于替换的字符的对象
	pos2	用于替换的子串起始位置
	n2	用于替换的字符数
参数（IN-OUT）	None	-
参数（OUT）	None	-
返回值	BasicString&	当前字符串替换后的内容，即*this
使用说明	可用于替换任何可隐式转换StringView的类型的子序列。	
注意事项	无	

函数原型	BasicString& replace(const_iterator i1, const_iterator i2, StringView sv);	
函数功能	替换函数	
参数（IN）	i1	将被替换的字符起始位置
	i2	将被替换的字符结束位置
	sv	用于替换的StringView
参数（IN-OUT）	None	-
参数（OUT）	None	-
返回值	BasicString&	当前字符串替换后的内容，即*this
使用说明	可用于StringView的内容替换迭代器绑定的子序列	
注意事项	无	

函数原型	size_type find(StringView sv, size_type pos = 0) const noexcept;	
函数功能	查找函数	
参数（IN）	sv	要搜索的StringView
	pos	开始搜索的位置

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	size_type	找到的字符位置，或若找不到这种字符则为npos
使用说明	前向查找StringView的内容	
注意事项	无	

函数原型	size_type rfind(StringView sv, size_type pos = npos) const noexcept;	
函数功能	查找函数	
参数（IN）	sv	要搜索的StringView
	pos	开始搜索的位置
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	size_type	找到的字符位置，或若找不到这种字符则为npos
使用说明	反向查找StringView的内容	
注意事项	无	

函数原型	size_type find_first_of(StringView sv, size_type pos = 0) const noexcept;	
函数功能	前向查找函数	
参数（IN）	sv	要搜索的StringView
	pos	开始搜索的位置
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	size_type	找到的字符位置，或若找不到这种字符则为npos
使用说明	前向查找StringView类型指定的字符集	
注意事项	无	

函数原型	size_type find_last_of(StringView sv, size_type pos = npos) const noexcept;	
函数功能	查找函数	
参数（IN）	sv	鉴别要搜索的字符的StringView
	pos	搜索结束的位置
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	size_type	找到的字符位置，或若找不到这种字符则为npos
使用说明	反向查找StringView类型指定的字符集	
注意事项	无	

函数原型	size_type find_first_not_of(StringView sv, size_type pos = 0) const noexcept;	
函数功能	查找函数	
参数（IN）	sv	鉴别要搜索的字符的StringView
	pos	搜索结束的位置
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	size_type	找到的字符位置，或若找不到这种字符则为npos
使用说明	前向查找不在StringView中的字符集	
注意事项	无	

函数原型	size_type find_last_not_of(StringView sv, size_type pos = npos) const noexcept;	
函数功能	查找函数	
参数（IN）	sv	鉴别要搜索的字符的StringView
	pos	搜索结束的位置
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	size_type	找到的字符位置，或若找不到这种字符则为npos
使用说明	反向查找不在StringView中的字符集	
注意事项	无	

函数原型	int compare(StringView sv) const noexcept;	
函数功能	比较函数	
参数（IN）	sv	要比较的另一个StringView
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	int	• 若*this在字典序中先出现于参数所指定的字符序列，则为正值。 • 若两个序列比较等价，则为零。 • 若*this在字典序中后出现于参数所指定的字符序列，则为负值。
使用说明	比较StringView的内容	
注意事项	无	

函数原型	int compare(size_type pos1, size_type n1, StringView sv) const;	
函数功能	比较函数	
参数（IN）	pos1	要比较的首字符的位置
	n1	要比较的字符数
	sv	要比较的另一StringView
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	int	• 若*this在字典序中先出现于参数所指定的字符序列，则为正值。 • 若两个序列比较等价，则为零。 • 若*this在字典序中后出现于参数所指定的字符序列，则为负值。
使用说明	比较StringView中某个子序列内容	
注意事项	无	

函数原型	template <typename T> int compare(size_type pos1, size_type n1, T const& t, size_type pos2, size_type n2 = npos) const;	
函数功能	比较函数	
参数（IN）	pos1	要比较的首字符的位置
	n1	要比较的字符数
	t	要比较的另一个可转换为StringView的类型数据
	pos2	给定字符串的要比较的首字符位置
	n2	给定字符串要比较的字符数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	int	• 若*this在字典序中先出现于参数所指定的字符序列，则为正值。 • 若两个序列比较等价，则为零。 • 若*this在字典序中后出现于参数所指定的字符序列，则为负值。
使用说明	比较可隐式转换成StringView类型的任意类型的子序列内容。	
注意事项	无	

全局函数定义

函数原型	void swap(BasicString<Allocator>& lhs, BasicString<Allocator>& rhs);	
函数功能	交换函数	
功能安全等级	不涉及	
参数（IN）	None	-
参数（INOUT）	lhs	要交换内容的容器1
	rhs	要交换内容的容器2
参数（OUT）	None	-
返回值	None	-
使用说明	该函数交换lhs和rhs的状态	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

3.12 Span

ara::core::Span所描述的对象能指代对象的相接序列，ara::core::Span相比于std::span，包含以下几点改动：

1. span.objectrep相关符号已删除
2. span.tuple相关符号已删除
3. font()、back()、const_pointer、const_reference已删除
4. 友元函数begin()、end()已删除
5. 所有对std::array的引用替换为ara::core::Array
6. constexpr类型赋值运算已删除
7. 增加Span::size_type
8. 增加MakeSpan重载

全局常量定义

类型名	定义	说明
dynamic_extent	std::numeric_limits<std::size_t>::max()	Span动态长度定义变量，创建动态长度Span所需常量

接口类定义

类名	ara::core::Span

命名空间	namespace ara::core
语法	template <typename T, std::size_t Extent = dynamic_extent> class Span {...};
模板参数	typename T Span元素类型
	std::size_t Extent = dynamic_extent Span内元素数量
头文件	#include "ara/core/span.h"
说明	表示连续序列元素

类内数据类型定义

类型名	定义	说明
element_type	T	Span内元素类型别名
value_type	typename std::remove_cv<element_type>::type	Span内元素值类型别名
index_type	std::size_t	Span索引类型别名
difference_type	std::ptrdiff_t	Span索引差值类型别名
size_type	index_type	Span内元素数量表示类型别名
pointer	element_type*	指向Span内元素的指针类型别名
reference	element_type&	指向Span内元素的引用类型别名
iterator	element_type*	指向Span内元素的迭代器类型别名，规范中指定为implementation_defined
const_iterator	element_type const *	指向Span内元素的常迭代器类型别名，规范中指定为implementation_defined
reverse_iterator	std::reverse_iterator<iterator>	指向Span内元素的反向迭代器类型别名
const_reverse_iterator	std::reverse_iterator<const_iterator>	指向Span内元素的常反向迭代器类型别名
extent	Extent	反映Span内元素数量的常量

成员函数定义

函数原型	constexpr ara::core::Span< T, Extent >::Span () noexcept;	
函数功能	默认构造函数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	默认构造函数	
注意事项	无	

函数原型	constexpr ara::core::Span< T, Extent >::Span (pointer ptr, index_type count);	
函数功能	使用给定指针和元素数量构造Span	
参数（IN）	ptr	连续内存元素组头指针
	count	元素个数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr ara::core::Span< T, Extent >::Span (pointer firstElem, pointer lastElem);	
函数功能	使用连续内存元素组的头指针和尾指针构造Span	
参数（IN）	pointer firstElem	连续内存元素组的头指针
	pointer lastElem	连续内存元素组的尾指针
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-

使用说明	该函数使用方法同std::span
注意事项	无

函数原型	template <std::size_t N> constexpr ara::core::Span< T, Extent >::Span (element_type(&arr)[N]) noexcept;	
函数功能	使用定长数组构造Span	
参数（IN）	arr	定长数组，长度为N
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	template <std::size_t N> constexpr ara::core::Span< T, Extent >::Span (Array< value_type, N > &arr) noexcept;	
函数功能	使用ara::core::Array<value_type, N>构造Span	
参数（IN）	arr	ara::core::Array
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数使用方法同使用std::array构造std::span	
注意事项	无	

函数原型	template <std::size_t N> constexpr ara::core::Span< T, Extent >::Span (Array< value_type, N > const &arr) noexcept;	
函数功能	使用ara::core::Array<value_type, N>构造Span	
参数（IN）	arr	ara::core::Array

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数使用方法同使用std::array构造std::span	
注意事项	无	

函数原型	template <typename Container> constexpr ara::core::Span< T, Extent >::Span (Container &cont);	
函数功能	使用给定容器构造Span	
参数（IN）	cont	容器
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	[ara::core::data(cont), ara::core::data(cont) + ara::core::size(cont))应该为有效范围。如果Extent与dynamic_extent不相等，ara::core::size(cont)应该与Extent相等。	
注意事项	无	

函数原型	template <typename Container> constexpr ara::core::Span< T, Extent >::Span (Container const &cont);	
函数功能	使用给定容器构造Span	
参数（IN）	cont	容器
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	[ara::core::data(cont), ara::core::data(cont) + ara::core::size(cont))应该为有效范围。如果Extent与dynamic_extent不相等，ara::core::size(cont)应该与Extent相等。	
注意事项	无	

函数原型	constexpr ara::core::Span< T, Extent >::Span (Span const &other) noexcept=default;	
函数功能	拷贝构造函数	
参数（IN）	other	另一个Span实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	template <typename U, std::size_t N> constexpr ara::core::Span< T, Extent >::Span (Span< U, N > const &s) noexcept;	
函数功能	从另一Span s的转换构造函数	
参数（IN）	s	另一个Span实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	Span& ara::core::Span< T, Extent >::operator= (Span const &other) noexcept=default;	
函数功能	拷贝赋值运算	
参数（IN）	other	另一个Span实例
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	Span&	新Span
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	template <std::size_t Count> constexpr Span<element_type, Count> ara::core::Span< T, Extent >::first () const;	
函数功能	获取仅含前count个元素的子Span	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Span<element_type, Count>	仅包含原Span的前Count个元素的子Span
使用说明	返回子Span，该子Span中仅包含原Span的前Count个元素 该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr Span<element_type, dynamic_extent> ara::core::Span< T, Extent >::first (index_type count) const;	
函数功能	获取仅含前count个元素的子Span	
参数（IN）	count	子Span所包含的原Span的元素个数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Span< element_type, dynamic_extent>	子Span
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	template <std::size_t Count> constexpr Span<element_type, Count> ara::core::Span< T, Extent >::last() const;	
函数功能	获取仅含后count个元素的子Span	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Span<element_type, Count>	仅包含后Span的前Count个元素的子Span
使用说明	返回子Span，该子Span中仅包含原Span的后Count个元素 该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr Span<element_type, dynamic_extent> ara::core::Span< T, Extent >::last (index_type count) const;	
函数功能	获取仅含后count个元素的子Span	
参数（IN）	count	子Span所包含的原Span的元素个数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Span< element_type, dynamic_extent>	子Span
使用说明	返回子Span，该子Span中仅包含原Span的后Count个元素 该函数使用方法同std::span	
注意事项	无	

函数原型	template <std::size_t Offset, std::size_t Count = dynamic_extent> constexpr auto ara::core::Span< T, Extent >::subspan () const -> Span<element_type, SEE_BELOW >;	
函数功能	获取子Span	
模板参数	Offset	获取的子Span的开始元素在原Span偏移量

	Count	新Span所包含的元素个数，默认dynamic_extent
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Span<element_type, SEE_BELOW >	子Span
使用说明	返回Span第二个模板参数如下计算出： Count != dynamic_extent ? Count : (Extent != dynamic_extent ? Extent - Offset : dynamic_ extent) 该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr Span<element_type, dynamic_extent> ara::core::Span< T, Extent >::subspan (index_type offset, index_type count=dynamic_extent) const;	
函数功能	获取子Span	
参数（IN）	index_type offset	获取的子Span的开始元素在原Span偏移量
	index_type count=dynamic_extent	新Span所包含的元素个数，默认dynamic_extent
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Span<element_type, dynamic_extent>	子Span
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr index_type ara::core::Span< T, Extent >::size () const noexcept;	
函数功能	获取Span中的元素个数	
参数（IN）	None	-

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	index_type	该Span的元素数
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr index_type ara::core::Span< T, Extent >::size_bytes () const noexcept;	
函数功能	获取该Span覆盖的总字节数	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	index_type	该Span覆盖的总字节数
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr bool ara::core::Span< T, Extent >::empty () const noexcept;	
函数功能	查询该Span是否为空	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	若为空，返回true，否则返回false
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr reference ara::core::Span< T, Extent >::operator[] (index_type idx) const;

函数功能	获取第idx个元素的引用	
参数（IN）	index_type	想要获取的元素在Span中的索引
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	reference	返回指向该Span第idx元素的引用
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr pointer ara::core::Span< T, Extent >::data () const noexcept;	
函数功能	获取指向该Span头元素的指针	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	pointer	获取指向该Span头元素的指针
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr iterator ara::core::Span< T, Extent >::begin () const noexcept;	
函数功能	获取指向该Span头元素的迭代器	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	iterator	获取指向该Span头元素的迭代器
使用说明	该函数使用方法同std::span	

注意事项	无

函数原型	constexpr iterator ara::core::Span< T, Extent >::end () const noexcept;	
函数功能	返回指向尾元素下一个位置的迭代器	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	iterator	返回指向尾元素下一个位置的迭代器
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr const_iterator ara::core::Span< T, Extent >::cbegin () const noexcept;	
函数功能	获取指向该Span头元素的常迭代器	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	const_iterator	获取指向该Span头元素的常迭代器
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr const_iterator ara::core::Span< T, Extent >::cend () const noexcept;	
函数功能	返回指向尾元素下一个位置的常迭代器	
参数（IN）	None	-
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	const_iterator	返回指向尾元素下一个位置的常迭代器
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr reverse_iterator ara::core::Span< T, Extent >::rbegin () const noexcept;	
函数功能	返回指向Span尾元素的反向迭代器	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	reverse_iterator	返回指向Span尾元素的反向迭代器
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr reverse_iterator ara::core::Span< T, Extent >::rend () const noexcept;	
函数功能	返回指向Span首元素前一个位置的反向迭代器	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	reverse_iterator	返回指向Span首元素的前一个位置反向迭代器
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr const_reverse_iterator ara::core::Span< T, Extent >::crbegin() const noexcept;

函数功能	返回指向Span尾元素的常反向迭代器	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	const_reverse_iterator	返回指向Span尾元素的常反向迭代器
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	constexpr const_reverse_iterator ara::core::Span< T, Extent >::crend() const noexcept;	
函数功能	返回指向Span首元素前一个位置的常反向迭代器	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	const_reverse_iterator	返回指向Span首元素前一个位置的常反向迭代器
使用说明	该函数使用方法同std::span	
注意事项	无	

3.12.1 全局函数定义

函数原型	template <typename T> constexpr Span<T> MakeSpan (T *ptr, typename Span< T >::index_type count);	
函数功能	根据输入的指针和获取的元素数构建Span实例	
参数（IN）	ptr	指针
	count	从ptr开始获取的元素数
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	Span<T>	构建出的新Span<T>实例
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	template <typename T> constexpr Span<T> MakeSpan (T *firstElem, T *lastElem);	
函数功能	根据输入的范围[firstElem, lastElem)构建Span实例	
参数（IN）	firstElem	截取容器起始指针位置
	lastElem	截取容器结束指针位置
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Span<T>	构建出的新Span<T>实例
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	template <typename T, std::size_t N> constexpr Span<T, N> MakeSpan (T(&arr)[N]) noexcept;	
函数功能	根据输入的原始数组创建Span实例	
参数（IN）	T(&arr)[N]	原始数组
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Span<T, N>	构建出的新Span<T, N>实例
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	template <typename Container> constexpr Span<typename Container::value_type> MakeSpan (Container &cont);
函数功能	根据输入的容器构建Span实例

参数（IN）	Constainer &cont	容器
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Span<typename Container::value_type>	构建出的新Span实例
使用说明	该函数使用方法同std::span	
注意事项	无	

函数原型	template <typename Container> constexpr Span<typename Container::value_type const> MakeSpan(Container const &cont);	
函数功能	根据输入的const容器构建Span实例	
参数（IN）	Container const &cont	容器
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Span<typename Container::value_type const>	构建出的新Span实例
使用说明	该函数使用方法同std::span	
注意事项	无	

3.13 InstanceSpecifier

接口类定义

类名	ara::core::InstanceSpecifier
命名空间	namespace ara::core
语法	class InstanceSpecifier final {...};
头文件	#include "ara/core/instance_specifier.h"
说明	描述了AUTOSAR实例说明符的类，基本上是AUTOSAR简称路径包装器。

成员函数定义

函数原型	explicit InstanceSpecifier (StringView metaModelIdentifier);	
函数功能	构造函数	
参数（IN）	metaModelIdentifier	字符串化的元模型标识符（短名称路径）路径分隔符是“/”。基础生命周期字符串必须超过构造的InstanceSpecifier的生命周期。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
异常	CoreException	如果给定的metaModelIdentifier无效 元模型标识符/短名称路径
使用说明	合法字符包括：‘0’到‘9’，‘a’到‘z’，‘A’到‘Z’，‘/’（用于拼接路径，至少要包含一个“/”）以及下划线‘_’。	
注意事项	• 如果输入的字符串中包含非法字符，则会抛出异常 • 由于InstanceSpecifier仅保存了StringView作为成员变量，使用者应注意传入字符串的生命周期维护	

函数原型	~InstanceSpecifier () noexcept;	
函数功能	析构函数	
参数（IN）	metaModelIdentifier	字符串化的元模型标识符（短名称路径）路径分隔符是“/”。基础生命周期字符串必须超过构造的生命周期InstanceSpecifier。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	析构函数	
注意事项	无	

函数原型	static Result<InstanceSpecifier> Create (StringView metaModelIdentifier);	
函数功能	以Result<InstanceSpecifier>创建一个类实例	
参数（IN）	metaModelIdentifier	字符串化的元模型标识符（短名称路径）路径分隔符是“/”。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Result< InstanceSpecifier >	一个包含合法的InstanceSpecifier，或者ErrorCode的Result类型的返回值。
错误码	CoreErrc::kInvalidMetaModelShortname	如果metaModelIdentifier中包含违法字符或者缺失。合法字符包括：‘0’到‘9’，‘a’到‘z’，‘A’到‘Z’，‘/’（用于拼接路径）以及下划线‘_’。
	CoreErrc::kInvalidMetaModelPath	如果metaModelIdentifier不是一个合法的路径。如果不包含“/”，则表示这不是一个路径，返回此错误码。
使用说明	创建一个新的类实例	
注意事项	• 线程安全	

函数原型	bool operator== (InstanceSpecifier const &other) const noexcept;	
函数功能	==操作符函数	
参数（IN）	other	InstanceSpecifier实例，与此实例进行比较
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	如果两个实例说明符都是完全相同的模型元素则返回true，否则返回false
使用说明	比较两个实例是否相同	
注意事项	无	

函数原型	bool operator== (StringView other) const noexcept;	
函数功能	==操作符函数	
参数（IN）	other	与字符串化的实例进行比较
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	如果两个实例说明符都是完全相同的模型元素则返回true，否则返回false
使用说明	比较两个实例是否相同	
注意事项	无	

函数原型	bool operator!= (InstanceSpecifier const &other) const noexcept;	
函数功能	!=操作符函数	
参数（IN）	other	InstanceSpecifier实例，与此实例进行比较
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	如果两个实例说明符都是完全相同的模型元素则返回true，否则返回false
使用说明	比较两个实例是否相同	
注意事项	无	

函数原型	bool operator!= (StringView other) const noexcept;	
函数功能	!=操作符函数	
参数（IN）	other	与字符串化的实例进行比较
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	bool	如果两个实例说明符都是完全相同的模型元素则返回true，否则返回false
使用说明	比较两个实例是否相同	
注意事项	无	

函数原型	bool operator< (InstanceSpecifier const &other) const noexcept;	
函数功能	<操作符函数	
参数（IN）	other	InstanceSpecifier实例，与此实例进行比较
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	如果此实例小于Other返回true；反之，返回false
使用说明	<操作符，用于与其他实例说明符相比	
注意事项	无	

函数原型	StringView ToString () const noexcept;	
函数功能	获得字符串化的实例	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	StringView	字符串形式的实例说明
使用说明	实例字符串化	
注意事项	• 线程安全	

3.14 Non-Member constainer access

本章节介绍ara::core的获取容器首元素地址，查询容器内元素数量，查询元素是否为空的接口，功能类似std::data，std::size，std::empty。

头文件均为ara/core/utility.h

函数原型	template <typename Container> constexpr auto data (Container &c) -> decltype(c.data());	
函数功能	获取指向容器首元素的指针	
参数（IN）	c	容器实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	data	指向容器首元素的指针
使用说明	容器类型必须含data()函数	
注意事项	无	

函数原型	template <typename Container> constexpr auto data (Container const &c) -> decltype(c.data());	
函数功能	获取指向容器首元素的指针	
参数（IN）	c	容器实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	data	指向容器首元素的常指针
使用说明	容器类型必须含data()函数	
注意事项	无	

函数原型	template <typename T, std::size_t N> constexpr T* data (T(&array)[N]) noexcept;	
函数功能	获取指向数组的指针	
参数（IN）	array	数组实例，长度为N
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	constexpr T*	获取指向数组的指针
使用说明	无	

注意事项	无

函数原型	template <typename E> constexpr E const* data (std::initializer_list< E > il) noexcept;	
函数功能	返回指向初始化器列表il首元素的指针	
参数（IN）	il	初始化器列表
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	constexpr E const*	返回指向初始化器列表il首元素的指针
使用说明	无	
注意事项	无	

函数原型	template <typename Container> constexpr auto size (Container const &c) -> decltype(c.size());	
函数功能	获取容器中元素的个数	
参数（IN）	c	元素实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	decltype(c.size())	容器中元素的个数
使用说明	要求容器必须包含size()成员函数	
注意事项	无	

函数原型	template <typename T, std::size_t N> constexpr std::size_t size (T const (&array)[N]) noexcept;	
函数功能	获取数组元素个数	
参数（IN）	array	数组
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	std::size_t	数组中的元素个数
使用说明	无	
注意事项	无	

函数原型	template <typename Container> constexpr auto empty (Container const &c) -> decltype(c.empty());	
函数功能	查询容器是否为空	
参数（IN）	c	容器实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	decltype(c.empty())	如果容器为空，返回true，否则，返回false
使用说明	容器必须包含empty()成员函数	
注意事项	无	

函数原型	template <typename T, std::size_t N> constexpr bool empty (T const (&array)[N]) noexcept;	
函数功能	查询数组是否空	
参数（IN）	array	数组实例，长度为N
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	false
使用说明	由于数组不可能含有0个元素，所以该函数恒返回false	
注意事项	无	

函数原型	template <typename E> constexpr bool empty (std::initializer_list< E > il) noexcept;

函数功能	查询给定初始化器列表是否为空	
参数（IN）	il	初始化列表实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	如果初始化列表为空，返回true，否则，返回false
使用说明	实例字符串化	
注意事项	无	

3.15 Initialize and Shutdown

头文件均为ara/core/initialization.h

函数原型	Result<void> Initialize ();	
命名空间	ara::core	
函数功能	初始化AP平台相关变量和线程	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Result< void >	表示AP平台的初始化的结果
使用说明	所有AP API的调用、对象的创建均需放在ara::core::Initialize()接口之后，且该接口需在主函数逻辑范围内调用。	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

函数原型	Result<void> Deinitialize ();	
命名空间	ara::core	
函数功能	释放AP平台相关变量和线程	
参数（IN）	None	-
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	ara::core::Result< void >	表示AP平台资源释放结构
使用说明	在所有AP API调用结束后应调用ara::core::Deinitialize()，且确保调用该接口后不再调用AP API接口或创建相关AP对象。	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

3.16 Abnormal process termination

头文件均在：ara/core/abort.h

函数原型	void Abort (char const *text) noexcept;	
函数功能	异常退出当前进程	
参数（IN）	text	作为输出的包括日志信息的自定义的信息
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	不提供注册AbortHandler和调用AbortHandler的功能	
注意事项	无	

3.17 Optional

ara::core::Optional是包含另一个对象和其初始化状态的对象，负责管理其包含对象的生命周期。

接口类定义

类名	Optional
命名空间	namespace ara::core
语法	template <class T> class Optional { ... };
头文件	#include "ara/core/optional.h"

说明	该类型包含另一个对象和其初始化状态的对象，并负责管理其包含对象的生命周期。 此类的所有成员和支持的结构（如全局关系运算符）应遵循C++17的std::optional，但不支持constexpr构造函数、抛出bad_optional_access异常和value()接口。

自定义 type_traits

别名	说明
EnableIfCopyConstructible<T>	typename std::enable_if<std::is_copy_constructible<T>::value>::type;
EnableIfMoveConstructible<T>	typename std::enable_if<std::is_move_constructible<T>::value>::type;
EnableIfConstructible<T, Args...>	typename std::enable_if<std::is_constructible<T, Args...>::value>::type;
NotEnableIfConvertible<U, T>	typename std::enable_if<! std::is_convertible<U, T>::value>::type;
EnableIfConvertible<U, T>	typename std::enable_if<std::is_convertible<U, T>::value>::type;
EnableIfNotOptional<T>	typename std::enable_if<! is_optional<typename std::decay<T>::type>::value>::type; 其中，is_optional用来判断模板类型是否是Optional类型。
EnableIfSwappable<T>	typename std::enable_if<std::is_swappable<T>::value>::type;
EnableIfT	typename std::enable_if<E, T>::type;

别名	说明
ConvertsFromOther	EnableIfT< std::is_constructible<T, O>::value && !std::is_constructible<T, Optional<U> &>::value && !std::is_constructible<T, Optional<U> &&>::value && !std::is_constructible<T, const Optional<U> &>::value && !std::is_constructible<T, const Optional<U> &&>::value && !std::is_convertible<Optional<U> &, T>::value && !std::is_convertible<Optional<U> &&, T>::value && !std::is_convertible<const Optional<U> &, T>::value && !std::is_convertible<const Optional<U> &&, T>::value>;

别名	说明
AssignsFromOther	EnableIfT< std::is_constructible<T, O>::value && std::is_assignable<T&, O>::value && !std::is_constructible<T, Optional<U>&>::value && !std::is_constructible<T, Optional<U> &&>::value && !std::is_constructible<T, const Optional<U>&>::value && !std::is_constructible<T, const Optional<U> &&>::value && !std::is_convertible<Optional<U>&, T>::value && !std::is_convertible<Optional<U> &&, T>::value && !std::is_convertible<const Optional<U>&, T>::value && !std::is_convertible<const Optional<U> &&, T>::value && !std::is_assignable<T&, Optional<U>&>::value && !std::is_assignable<T&, Optional<U> &&>::value && !std::is_assignable<T&, const Optional<U>&>::value && !std::is_assignable<T&, const Optional<U> &&>::value>;

成员函数定义

函数原型	Optional() noexcept;	
函数功能	构造包含未初始化对象的Optional对象。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	构造函数。	
注意事项	无	

函数原型	Optional(nullopt_t) noexcept;	
函数功能	构造包含未初始化对象的Optional。	
参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	构造函数。PS：ara::core::Optional<T> opt(ara::core::nullopt); 其中T表示某一类型，ara::core::nullopt为ara::core::nullopt_t的唯一实例。	
注意事项	无	

函数原型	Optional(const Optional<T>& rhs);	
函数功能	拷贝构造。	
参数（IN）	rhs	另一个T类型的Optional对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	由Optional<T>类型的对象拷贝构造出一个Optional<T>对象。	
注意事项	无	

函数原型	template<typename T_=T, typename= EnableIfMoveConstructible<T_>> Optional(Optional<T> && rhs) noexcept(std::is_nothrow_move_constructible<T_>());	
函数功能	移动构造。当is_move_constructible_v<T>为true时，该移动构造函数才会被定义；当T具有不抛出异常的移动构造函数时，该移动构造函数会被声明为noexcept。	
参数（IN）	rhs	另一个T类型的Optional右值对象。
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	None	-
使用说明	由Optional<T>类型的右值对象构造出一个Optional<T>对象。	
注意事项	无	

函数原型	template<typename... Args, typename = EnableIfConstructible<T, Args &&...>> explicit Optional(in_place_t, Args &&... args);	
函数功能	原地构造。当is_constructible_v<T, Args...>为true时，该构造函数才会被定义。	
参数（IN）	in_place_t	一个标记类型，其指示Optional包含的类型应该原地构造，即不会发生复制操作。
参数（IN）	args	可以构造T类型的对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	示例用法：ara::core::Optional<int> opt(ara::core::in_place, 1); 其中，ara::core::inplace为ara::core::in_place_t的唯一实例。	
注意事项	无	

函数原型	template<typename U, typename... Args, typename = EnableIfConstructible<T, std::initializer_list<U>, Args &&...>> explicit Optional(in_place_t, std::initializer_list<U> il, Args &&... args);	
函数功能	原地构造。当is_constructible_v<T, initializer_list<U>&, Args&&...>为true时，该构造函数才会被定义。	
参数（IN）	in_place_t	一个标记类型，其指示Optional包含的类型应该原地构造，即不会发生复制操作。
参数（IN）	il	U类型的initializer_list，U可以构造T。
参数（IN）	args	initializer_list附加参数(可变长参数，可能为空)。

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	示例用法：ara::core::Optional<std::vector<int>> opt(ara::core::in_place, {1,2,3}); 其中，in_place_t和in_place说明同上。	
注意事项	无	

函数原型	template<typename U = T, typename = EnableIfConstructible<T, U &&>, typename = typename std::enable_if<!std::is_same<typename std::decay<U>::type, in_place_t>::value>::type, typename = typename std::enable_if<! std::is_same<Optional<T>, typename std::decay<U>::type>::value>::type> Optional(U && v);	
函数功能	移动构造。当is_constructible_v<T, U&&>为true、is_same_v<decay_t<U>, in_place_t>为false和is_same_v<optional<T>,decay_t<U>>为false时，该构造函数才会被定义。	
参数（IN）	v	U类型的右值对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-

使用说明	示例用法：
	int t = 1;
	ara::core::Optional<int> opt(std::move(t));	

注意事项	无	

函数原型	template <class U, typename = ConvertsFromOther<T, U, const U&>> Optional(const Optional<U>& rhs);
函数功能	转换拷贝构造。如果NotEnableIfConvertible<const U&, T>为true，则该构造函数可为explicit。

参数（IN）	rhs	U类型的Optional对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	template <class U, typename = ConvertsFromOther<T, U, U &&>> Optional(Optional<U>&& rhs);	
函数功能	转换移动构造。如果NotEnableIfConvertible<U&&, T>为true，则该构造函数可为explicit。	
参数（IN）	rhs	U类型的Optional右值对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	~Optional();	
函数功能	析构函数。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	None	
注意事项	无	

函数原型	Optional<T>& operator=(nullopt_t);	
函数功能	置空赋值。	
参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Optional<T>&	被赋值对象本身。
使用说明	示例：opt = ara::core::nullopt; 其中，opt为一个ara::coreOptional对象。	
注意事项	无	

函数原型	Optional<T>& operator=(const Optional<T>& rhs);	
函数功能	拷贝赋值。	
参数（IN）	rhs	另一个T类型的Optional对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Optional<T>&	被赋值对象本身。
使用说明	None	
注意事项	无	

函数原型	Optional<T>& operator=(Optional<T> && rhs) noexcept(std::is_nothrow_move_assignable<T>::value &&std::is_nothrow_move_constructible<T>::value);	
函数功能	移动赋值。当is_nothrow_move_assignable_v<T> && is_nothrow_move_constructible_v<T>为true时，该移动赋值函数会被声明为noexcept。	
参数（IN）	rhs	一个T类型的Optional右值对象。
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	Optional<T>&	被赋值对象本身。
使用说明	示例： ara::core::Optional<int> opt1; ara::core::Optional<int> opt2(ara::core::in_place, 1); opt1 = std::move(opt2); 此时opt1的值为1，opt2不包含值。	
注意事项	无	

函数原型	template<typenameU, typename= EnableIfNotOptional<U>> Optional<T>& operator=(U && rhs);	
函数功能	移动赋值。当U为非Optional<T>类型的对象时，该接口才会被定义。	
参数（IN）	rhs	U类型的可构造T类型的右值对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Optional<T>&	返回被赋值对象本身。
使用说明	示例： ara::core::Optional<int> opt1; int t = 1; opt1 = std::move(t); 此时opt1的值为1。	
注意事项	无	

函数原型	template <class U, typename = AssignsFromOther<T, U, const U&>> Optional<T>& operator=(const Optional<U>& rhs);	
函数功能	转换拷贝赋值。	
参数（IN）	rhs	U类型的Optional对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Optional<T>&	返回被赋值对象本身。
使用说明	无	

注意事项	无

函数原型	template <class U, typename = AssignsFromOther<T, U, U>> Optional<T>& operator=(Optional<U>&& rhs);	
函数功能	转换移动赋值。	
参数（IN）	rhs	U类型的Optional右值对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Optional<T>&	返回被赋值对象本身。
使用说明	无	
注意事项	无	

函数原型	template<class...Args, typename T_ = T, typename = EnableIfConstructible<T_, Args &&...>> T& emplace(Args &&... args);	
函数功能	为所包含的对象分配一个新值。当is_constructible_v<T, Args&&...>为true时，该接口才会被定义。	
参数（IN）	args	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&	被包含对象引用。
使用说明	示例： ara::core::Optional<int> opt; int i = opt.emplace(1); 则i=1; *opt=1。	
注意事项	无	

函数原型	template<classU, class...Args, typename T_ = T, typename = EnableIfConstructible<T_, std::initializer_list<U>, Args &&...>> T& emplace(std::initializer_list<U>il, Args &&...args);

函数功能	为所包含的对象分配一个新值。当is_constructible_v<T_, std::initializer_list<U>, Args &&...>为true时，该接口才会被定义。	
参数（IN）	il	U类型的initializer_list，U可以构造T。
参数（IN）	args	initializer_list附加参数(可变长参数，可能为空)。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&	被包含对象引用。
使用说明	示例： ara::core::Optional<std::vector<int>> opt; std::vector<int> v = opt.emplace({1,2,3}); 则v[1]=1; (*opt)[1]=1。	
注意事项	无	

函数原型	template<typename T_= T, typename = EnableIfMoveConstructible<T_>> void swap(Optional<T>& rhs) noexcept(std::is_nothrow_move_constructible<T>::value && noexcept(std::swap(std::declval<T&>(),std::declval<T&>())));	
函数功能	交换两个对象的值。当is_move_constructible_v<T>为true时，该接口才会被定义。	
参数（IN）	rhs	另一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	None	
注意事项	• 线程不安全	

函数原型	constexpr const T* operator->() const;
函数功能	类成员访问运算符重载。返回被包含对象的const指针，可以用来访问被包含对象的public成员。

参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	const T*	被包含对象的const指针
使用说明	示例： ara::core::Optional<std::string> opt(ara::core::in_place, "str"); std::size_t sz = opt->length(); 此时sz = 3。 注意：当opt包含的对象未初始化时，返回的为空指针，不可访问其public成员！	
注意事项	无	

函数原型	constexpr T* operator->();	
函数功能	类成员访问运算符重载。返回被包含对象的指针，可以用来访问被包含对象的public成员。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T*	被包含对象的指针。
使用说明	示例： ara::core::Optional<std::string> opt(ara::core::in_place, "str"); std::size_t sz = opt->length(); 此时sz = 3。 注意：当opt包含的对象未初始化时，返回的为空指针，不可访问其public成员！	
注意事项	无	

函数原型	constexpr const T& operator*() const&;	
函数功能	解引用运算符重载。返回被包含对象的const引用。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	const T&	被包含对象的const引用。
使用说明	示例： ara::core::Optional<std::string> opt(ara::core::in_place, "str"); *opt的值为"str"。 注意：不可对包含未初始化值的Optional使用此操作符！	
注意事项	无	

函数原型	constexpr T& operator*() &;	
函数功能	解引用运算符重载。返回被包含对象的引用。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&	被包含对象的引用。
使用说明	示例： ara::core::Optional<std::string> opt(ara::core::in_place, "str"); *opt的值为"str"。 注意：不可对包含未初始化值的Optional使用此操作符！	
注意事项	无	

函数原型	constexpr T && operator*() &&;	
函数功能	解引用运算符重载。返回被包含对象的右值。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&&	被包含对象的右值。
使用说明	注意：不可对包含未初始化值的Optional使用此操作符！	
注意事项	无	

函数原型	constexpr const T && operator*() const&&;

函数功能	解引用运算符重载。返回被包含对象的右值。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	const T&&	被包含对象的const右值。
使用说明	注意：不可对包含未初始化值的Optional使用此操作符！	
注意事项	无	

函数原型	constexpr explicit operator bool() const noexcept;	
函数功能	bool运算符重载。返回是否包含值。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	包含值为true，不包含值为false。
使用说明	示例： ara::core::Optional<int> opt if (!opt) std::cout << "opt has no value" << std::endl;	
注意事项	无	

函数原型	constexpr bool has_value() const noexcept;	
函数功能	返回是否包含值。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	包含值为true，不包含值为false。
使用说明	None	
注意事项	无	

函数原型	template<typename U, typename T_=T, typename = EnableIfCopyConstructible<T_>, typename = EnableIfConvertible<U&&, T_>> constexpr T value_or(U && v) const&;	
函数功能	访问值，如果不包含值，则返回传入的值。当is_copy_constructible_v<T> && is_convertible_v<U&&, T>为true时，该函数才会被定义	
参数（IN）	v	传入的值
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T	T类型对象
使用说明	示例： ara::core::Optional<int> opt; int i = opt.value_or(1); 由于opt为空，所以i=1。	
注意事项	无	

函数原型	template<typename U, typename T_=T, typename = EnableIfMoveConstructible<T_>, typename = EnableIfConvertible<U&&, T_>> constexpr T value_or(U && v) &&;	
函数功能	访问值，如果不包含值，则返回传入的值。当is_move_constructible_v<T> && is_convertible_v<U&&, T>为true时，该函数才会被定义	
参数（IN）	v	传入的值
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T	T类型对象
使用说明	示例： ara::core::Optional<int> opt; int i = opt.value_or(1); 由于opt为空，所以i=1。	
注意事项	无	

函数原型	void reset() noexcept;

函数功能	销毁Optional对象，使其成为未包含值状态。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	None	
注意事项	无	

函数原型	void reset() noexcept;	
函数功能	销毁Optional对象，使其成为未包含值状态。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	None	
注意事项	无	

全局函数定义

函数原型	template<typename T, typename U> constexpr bool operator==(const Optional<T> & x, const Optional<U>& y);	
函数功能	比较两个Optional对象。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	y	一个Optional<U>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x，y其中一个包含值，返回false；x，y均不包含值返回true；x，y均包含值，且值相等返回true，反之返回false。

使用说明	None
注意事项	无

函数原型	template<typename T, typename U> constexpr bool operator!=(const Optional<T> & x, const Optional<U>& y);	
函数功能	比较两个Optional对象。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	y	一个Optional<U>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x，y其中一个包含值，返回true；x，y均不包含值返回false；x，y均包含值，且值不等返回true，反之返回false。
使用说明	None	
注意事项	无	

函数原型	template<typename T, typename U> constexpr bool operator<(const Optional<T> & x, const Optional<U>& y);	
函数功能	比较两个Optional对象。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	y	一个Optional<U>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	y不包含值，返回false；y包含值，x不包含值返回true；x，y均包含值返回*x < *y。
使用说明	None	
注意事项	无	

函数原型	template<typename T, typename U> constexpr bool operator>(const Optional<T> & x, const Optional<U>& y);	
函数功能	比较两个Optional对象。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	y	一个Optional<U>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值，返回false；x包含值，y不包含值返回true；x，y均包含值返回*x > *y。
使用说明	None	
注意事项	无	

函数原型	template<typename T, typename U> constexpr bool operator<=(const Optional<T> & x, const Optional<U>& y);	
函数功能	比较两个Optional对象。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	y	一个Optional<U>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值，返回true；x包含值，y不包含值返回false；x，y均包含值返回*x <= *y。
使用说明	None	
注意事项	无	

函数原型	template<typename T, typename U> constexpr bool operator>=(const Optional<T> & x, const Optional<U>& y);
函数功能	比较两个Optional对象。

参数（IN）	x	一个Optional<T>对象。
参数（IN）	y	一个Optional<U>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	y不包含值，返回true；y包含值，x不包含值返回false；x，y均包含值返回*x >= *y。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator==(const Optional<T>& x, nullopt_t) noexcept;	
函数功能	比较Optional<T>对象和ara::core::nullopt。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x包含值返回false，反之返回true。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator==(nullopt_t, const Optional<T>& x) noexcept;
函数功能	比较ara::core::nullopt和Optional<T>对象。

参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x包含值返回false，反之返回true。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator!=(const Optional<T>& x, nullopt_t) noexcept;	
函数功能	比较Optional<T>对象和ara::core::nullopt。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x包含值返回true，反之返回false。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator!=(nullopt_t, const Optional<T>& x) noexcept;
函数功能	比较ara::core::nullopt和Optional<T>对象。

参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x包含值返回true，反之返回false。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator<(const Optional<T>& x, nullopt_t) noexcept;	
函数功能	比较Optional<T>对象和ara::core::nullopt。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	只返回false。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator<(nullopt_t, const Optional<T>& x) noexcept;
函数功能	比较ara::core::nullopt和Optional<T>对象。

参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x包含值返回true，反之返回false。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator<=(const Optional<T>& x, nullopt_t) noexcept;	
函数功能	比较Optional<T>对象和ara::core::nullopt。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回true，x包含值返回false。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator<=(nullopt_t, const Optional<T>& x) noexcept;
函数功能	比较ara::core::nullopt和Optional<T>对象。

参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x包含值返回true，反之返回false。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator>(const Optional<T>& x, nullopt_t) noexcept;	
函数功能	比较Optional<T>对象和ara::core::nullopt。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x包含值返回true，反之返回false。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator>(nullopt_t, const Optional<T>& x) noexcept;
函数功能	比较ara::core::nullopt和Optional<T>对象。

参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	只返回false。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator>=(const Optional<T>& x, nullopt_t) noexcept;	
函数功能	比较Optional<T>对象和ara::core::nullopt。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	只返回true。
使用说明	None	
注意事项	无	

函数原型	template<typename T> constexpr bool operator>=(nullopt_t, const Optional<T>& x) noexcept;
函数功能	比较ara::core::nullopt和Optional<T>对象。

参数（IN）	nullopt_t	nullopt_t为一种标识未包含对象Optional的类型，属于ara::core命名空间；具有唯一实例ara::core::nullopt。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	只返回true。
使用说明	None	
注意事项	无	

函数原型	template <typename T, typename U> constexpr bool operator==(const Optional<T>& x, const U& v);	
函数功能	比较Optional<T>对象和U类型对象。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	v	一个U类型对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回false；x包含值返回*x == v。
使用说明	None	
注意事项	无	

函数原型	template <typename T, typename U> constexpr bool operator==(const U& v, const Optional<T>& x);	
函数功能	比较U类型对象和Optional<T>对象。	
参数（IN）	v	一个U类型对象。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	bool	x不包含值返回false；x包含值返回v == *x。
使用说明	None	
注意事项	无	

函数原型	template <typename T, typename U> constexpr bool operator!=(const Optional<T>& x, const U& v);	
函数功能	比较Optional<T>对象和U类型对象。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	v	一个U类型对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回true；x包含值返回*x != v。
使用说明	None	
注意事项	无	

函数原型	template <typename T, typename U> constexpr bool operator!=(const U& v, const Optional<T>& x);	
函数功能	比较U类型对象和Optional<T>对象。	
参数（IN）	v	一个U类型对象。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回true；x包含值返回v != *x。
使用说明	None	
注意事项	无	

函数原型	template <typename T, typename U> constexpr bool operator<(const Optional<T>& x, const U& v);	
函数功能	比较Optional<T>对象和U类型对象。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	v	一个U类型对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回true；x包含值返回*x < v。
使用说明	None	
注意事项	无	

函数原型	template <typename T, typename U> constexpr bool operator<(const U& v, const Optional<T>& x);	
函数功能	比较U类型对象和Optional<T>对象。	
参数（IN）	v	一个U类型对象。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回false；x包含值返回v < *x。
使用说明	None	
注意事项	无	

函数原型	template <typename T, typename U> constexpr bool operator<=(const Optional<T>& x, const U& v);	
函数功能	比较Optional<T>对象和U类型对象。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	v	一个U类型对象。

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回true；x包含值返回*x <= v。
使用说明	None	
注意事项	无	

函数原型	template <typename T, typename U> constexpr bool operator<=(const U& v, const Optional<T>& x);	
函数功能	比较U类型对象和Optional<T>对象。	
参数（IN）	v	一个U类型对象。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回false；x包含值返回v <= *x。
使用说明	None	
注意事项	无	

函数原型	template <typename T, typename U> constexpr bool operator>(const Optional<T>& x, const U& v);	
函数功能	比较Optional<T>对象和U类型对象。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	v	一个U类型对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回false；x包含值返回*x > v。
使用说明	None	

注意事项	无

函数原型	template <typename T, typename U> constexpr bool operator>(const U& v, const Optional<T>& x);	
函数功能	比较U类型对象和Optional<T>对象。	
参数（IN）	v	一个U类型对象。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回true；x包含值返回v > *x。
使用说明	None	
注意事项	无	

函数原型	template <typename T, typename U> constexpr bool operator>=(const Optional<T>& x, const U& v);	
函数功能	比较Optional<T>对象和U类型对象。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	v	一个U类型对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回false；x包含值返回*x >= v。
使用说明	None	
注意事项	无	

函数原型	template <typename T, typename U> constexpr bool operator>=(const U& v, const Optional<T>& x);
函数功能	比较U类型对象和Optional<T>对象。

参数（IN）	v	一个U类型对象。
参数（IN）	x	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	x不包含值返回true；x包含值返回v >= *x。
使用说明	None	
注意事项	无	

函数原型	template<typename T, typename = EnableIfMoveConstructible<T>, typename = EnableIfSwappable<T>> void swap(Optional<T>& x, Optional<T>& y) noexcept(noexcept(x.swap(y)));	
函数功能	交换两个Optional<T>对象。当is_move_constructible_v<T>为true且is_swappable_v<T>为true时，该函数才会被定义；当x.swap(y)不抛出异常时，该函数才会被声明为noexcept。	
参数（IN）	x	一个Optional<T>对象。
参数（IN）	y	一个Optional<T>对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	None	
注意事项	• 线程不安全	

函数原型	template<typename T> constexpr Optional<typename std::decay<T>::type> make_optional(T && v);	
函数功能	创建一个Optional<T>对象。	
参数（IN）	v	一个T类型的右值对象。
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	Optional<typename std::decay<T>::type>	T非引用类型：一个T类型的Optional对象；T为引用类型：一个被T引用类型的Optional对象。
使用说明	示例： ara::core::Optional<int> opt = ara::core::make_optional<int>(1);	
注意事项	无	

函数原型	template<typename T, typename... Args> constexpr Optional<T> make_optional(Args &&... args);	
函数功能	创建一个Optional<T>对象。	
参数（IN）	args	一个T类型的参数包。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Optional<T>	一个T类型的Optional对象。
使用说明	示例： std::vector<int> il ={1,2,3}; ara::core::Optional<std::vector<int>> opt2 =ara::core::make_optional<std::vector<int>>(il);	
注意事项	无	

函数原型	template<typename T, typename U, typename... Args> constexpr Optional<T> make_optional(std::initializer_list<U> il, Args &&... args);	
函数功能	创建一个Optional<T>对象。	
参数（IN）	il	U类型的initializer_list，U可以构造T。
参数（IN）	args	initializer_list附加参数(可变长参数，可能为空)。
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	Optional<T>	一个T类型的Optional对象。
使用说明	示例： ara::core::Optional<std::vector<int>> opt1 = ara::core::make_optional<std::vector<int>>({1,2,3});	
注意事项	无	

Hash函数对象类定义

类名	hash
命名空间	namespace std
语法	template <class T> struct hash<ara::core::Optional<T>> { ... };
头文件	#include "ara/core/optional.h"

说明	该类型是一个函数对象类型，用来计算ara::core::Optional对象的hash值。用法如下：
	int t =2;
	ara::core::Optional<int>opt(ara::core::in_place, t);
	std::hash<ara::core::Optional<int>>()(opt); // 其结果应和std::hash<int>()(t); 结果一致。
	注：当opt为空时，该调用会返回0。

3.18 Variant

接口类定义

类名	Variant
命名空间	namespace ara::core
语法	template <typename... Types> class Variant;
头文件	#include "ara/core/variant.h"
说明	ara::core::Variant代表一种类型安全的union。 1、与union相同点：可以包含多种类型的数据，但任何时刻最多只能存放其中一种类型的数据。 2、与union不同点：Variant无须借助外力只需要通过查询自身就可辨别实际所存放数据的类型。

Variant 辅助类

名称	说明
template <typename T> ara::core::variant_size_v	获取所有一个Variant所包含类型的数量，T为一个Variant类型。
template <size_t I, typename T> ara::core::variant_alternative_t	获取一个Variant在I位置的类型，T为一个Variant类型。
ara::core::variant_npos	表示Variant处于无效状态的索引。
struct monostate;	旨在用作ara::core::Variant的空类型的替代类型。特别是，非默认可构造类型的Variant可以ara::core::monostate列为其第一个类型，使得Variant本身可默认构造。除了命名空间改为ara::core，其余接口和行为与std::monostate一致。

Variant成员函数定义

函数原型	constexpr Variant() noexcept(std::is_nothrow_default_constructible<U>::value)	
函数功能	构造一个初始化了第一个类型的Variant。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	构造函数。	
注意事项	无	

函数原型	Variant(const Variant& other);	
函数功能	拷贝构造。	
参数（IN）	other	另一个Variant对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	拷贝构造函数。	

注意事项	无

函数原型	Variant(Variant&& other) noexcept(/* see below */);	
函数功能	移动构造。当std::is_move_constructible_v对Variant中包含的所有类型为true时，该移动构造函数会被声明为noexcept。	
参数（IN）	other	另一Variant右值对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	template<typename T, typename... Args> constexpr explicit Variant(ara::core::in_place_type_t<T>, Args&&... args);	
函数功能	inplace构造。原地构造一个初始化为T类型的Variant，如果T类型在声明的Variant中出现多次或由args不能构造出T，则该构造函数不可用。	
参数（IN）	ara::core::in_place_type_t<T>	ara::core::in_place_type<T>的类型。
	args	构造T类型的值。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	示例：ara::core::Variant<std::string, std::vector<int>, float> var(ara::core::in_place_type<std::string>, "AAA");	
注意事项	无	

函数原型	template <typename T, typename U, typename... Args> constexpr explicit Variant(ara::core::in_place_type_t<T>, std::initializer_list<U> il, Args&&... args);

函数功能	inplace构造。由il原地构造一个初始化为T类型的Variant，如果T类型在声明的Variant中出现多次或由args和il不能构造出T，则该构造函数不可用。	
参数（IN）	ara::core::in_place_type_t<T>	ara::core::in_place_type<T>的类型。
	il	构造T类型的值。
	args	构造il的值。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	示例： 1、ara::core::Variant<std::string, std::vector<int>, char> var1(ara::core::in_place_type<std::vector<int>>, {1,2,3}); 2、ara::core::Variant<std::string, std::vector<int>, char> var1(ara::core::in_place_type<std::vector<int>>, 3, 3);	
注意事项	无	

函数原型	template <size_t I, typename... Args> constexpr explicit Variant(ara::core::in_place_index_t<I>, Args&&... args);	
函数功能	inplace构造。原地构造一个初始化为第I个类型的Variant，如果I不小于Variant可选类型的数量， 第I个类型数次出现不止一次或由args不能构造出第I个类型，则该构造函数不可用。	
参数（IN）	ara::core::in_place_index_t<I>	ara::core::in_place_type<I>的类型。
	args	构造第I个可选类型的值。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	示例：ara::core::Variant<std::string, std::vector<int>, float> var(ara::core::in_place_index<0>, "AAA");	
注意事项	无	

函数原型	template <size_t I, typename U, typename... Args> constexpr explicit Variant(ara::core::in_place_index_t<I>, std::initializer_list<U> il, Args&&... args);	
函数功能	inplace构造。由il原地构造一个初始化为第I个类型T类型的Variant，如果I不小于Variant中可选类型的数量或T类型在声明的Variant中出现多次或由args和il不能构造出T，则该构造函数不可用。	
参数（IN）	ara::core::in_place_index_t<I>	ara::core::in_place_index<I>的类型。
	il	构造T类型的值。
	args	构造il的值。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	示例： 1、ara::core::Variant<std::string, std::vector<int>, char> var1(ara::core::in_place_index<1>, {1,2,3}); 2、ara::core::Variant<std::string, std::vector<int>, char> var1(ara::core::in_place_index<1>, 3, 3);	
注意事项	无	

函数原型	template<typename T> constexpr Variant(T&& t) noexcept(/* see below*/);	
函数功能	转换构造函数。当T可以不抛异常地给Variant中的构造某一可选类型时，该接口保证不抛异常。	
参数（IN）	t	T类型的值。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-

使用说明	示例：ara::core::Variant<std::string, uint32_t> var5(5);// OK, choose uint32
	当有歧义时，需要显示指定入参类型，如：
	ara::core::Variant<float, uint64_t , double> var4(3ul); // OK, choose uint64	

注意事项	无	

函数原型	~Variant();	
函数功能	析构函数。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	None	
注意事项	无	

函数原型	Variant& operator=(const Variant& rhs);	
函数功能	拷贝赋值。	
参数（IN）	rhs	另一个含有相同可选类型的Variant对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Variant&	被赋值对象本身。
使用说明	None	
注意事项	无	

函数原型	Variant& operator=(Variant&& rhs) noexcept;	
函数功能	移动赋值。	
参数（IN）	rhs	另一个含有相同可选类型的Variant对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Variant&	被赋值对象本身。
使用说明	None	

注意事项	无

函数原型	template<typename T> Variant& operator=(T&& t) noexcept(/* see below */);	
函数功能	转换赋值函数。当T可以不抛异常地给Variant中的某一可选类型赋值或者构造时，该接口保证不抛异常。	
参数（IN）	t	T类型的值。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	Variant&	被赋值对象本身。
使用说明	示例：ara::core::Variant<int, std::string> v{2021}; a = 2022;	
注意事项	无	

函数原型	template <typename T, typename... Args> T& emplace(Args&&... args);	
函数功能	为所包含的对象分配一个新值。T在该Variant中出现且仅出现一次。	
参数（IN）	args	待分配给Variant的值。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&	被包含对象引用。
使用说明	示例：ara::core::Variant<int, std::string> var{0}; 此时var里的值为0, var.emplace<int>(1); 此时var所含值为1。	
注意事项	无	

函数原型	template <typename T, typename U, typename... Args> T& emplace(std::initializer_list<U> il, Args&&... args);	
函数功能	为所包含的对象分配一个新值。T在该Variant中出现且仅出现一次。	
参数（IN）	il	构造T的值。

	args	构造il的值。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&	被包含对象引用。
使用说明	示例：ara::core::Variant<std::vector<int>> var1; var1.emplace<std::vector<int>>({1,2,2}); var1.emplace<std::vector<int>>(2, 2);	
注意事项	无	

函数原型	template <size_t I, typename... Args> variant_alternative_t<I, Variant>& emplace(Args&&... args);	
函数功能	为第I个类型分配一个新值。	
参数（IN）	args	待分配给Variant的值。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	variant_alternative_t<I, Variant>&	第I个类型对象引用。
使用说明	示例：ara::core::Variant<int, std::string> var{0}; 此时var里的值为0, var.emplace<0>(1); 此时var所含值为1。	
注意事项	无	

函数原型	template <size_t I, typename... Args> variant_alternative_t<I, Variant>& emplace(Args&&... args);	
函数功能	为第I个类型分配一个新值。	
参数（IN）	il	构造T的值。
	args	构造il的值。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&	第I个类型对象引用。

使用说明	示例：ara::core::Variant<std::vector<int>> var1; var1.emplace<0>({1,2,2}); var1.emplace<0>(2, 2);
注意事项	无

函数原型	constexpr bool valueless_by_exception() const noexcept;	
函数功能	当且仅当variant包含值时返回false。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	包含值为false，不包含值为true。
使用说明	None	
注意事项	无	

函数原型	constexpr std::size_t index() const noexcept;	
函数功能	返回Variant当前持有的类型的索引。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	std::size_t	Variant当前持有的类型的索引，未持有值返回ara::core::variant_npos。
使用说明	None	
注意事项	无	

函数原型	void swap(Variant& rhs) noexcept(/* see below */);	
函数功能	交换函数。如果Variant里的每个类型都满足移动构造不抛异常，则该函数保证不抛异常	
参数（IN）	None	-

参数（INOUT）	rhs	要交换内容的Variant
参数（OUT）	None	-
返回值	None	-
使用说明	交换*this和rhs。	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

全局函数定义

函数原型	template <typename... Types> void swap(Variant<Types...>& v, Variant<Types...>& w) noexcept(/* see below */);	
函数功能	交换函数。如果Variant里的每个类型都满足移动构造不抛异常，则该函数保证不抛异常。	
参数（IN）	None	-
参数（INOUT）	lhs	要交换内容的Variant1
	rhs	要交换内容的Variant2
参数（OUT）	None	-
返回值	None	-
使用说明	该函数交换lhs和rhs。	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

函数原型	template<typename T, typename... Types> constexpr bool holds_alternative(const Variant<Types...>& var) noexcept;	
函数功能	检查var是否存储了T类型的值。如果Variant中包含了不止一个T类型，该接口不可用。	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	bool	如果存储了T类型的值，返回true，反之，返回false。
使用说明	无	
注意事项	无	

函数原型	template<size_t I, typename... Types> constexpr ara::core::variant_alternative_t<I, ara::core::Variant<Types...>>& get(ara::core::Variant<Types...>& v);	
函数功能	获取v的第I个类型的值。如果I大于等于Variant的类型数量，该接口不可用；如果Variant不含值，则抛异常。	
参数（IN）	v	Variant对象
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::variant_alternative_t<I, ara::core::Variant<Types...>>&	Variant中存储的第I个类型对象的引用。
使用说明	无	
注意事项	无	

函数原型	template<size_t I, typename... Types> constexpr ara::core::variant_alternative_t<I, ara::core::Variant<Types...>>&& get(ara::core::Variant<Types...>&& v);	
函数功能	获取v的第I个类型的值。如果I大于等于Variant的类型数量，该接口不可用；如果Variant不含值，则抛异常。	
参数（IN）	v	Variant右值对象
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::variant_alternative_t<I, ara::core::Variant<Types...>>&	Variant中存储的第I个类型对象的右值引用。
使用说明	无	

注意事项	无

函数原型	template<size_t I, typename... Types> constexpr const ara::core::variant_alternative_t<I, ara::core::Variant<Types...>>& get(const ara::core::Variant<Types...>& v);	
函数功能	获取v的第I个类型的值。如果I大于等于Variant的类型数量，该接口不可用；如果Variant不含值，则抛异常。	
参数（IN）	v	Variant对象
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	const ara::core::variant_alternative_t<I, ara::core::Variant<Types...>>&&	Variant中存储的第I个类型对象的const引用。
使用说明	无	
注意事项	无	

函数原型	template<size_t I, typename... Types> constexpr const ara::core::variant_alternative_t<I, ara::core::Variant<Types...>>&& get(const ara::core::Variant<Types...>&& v);	
函数功能	获取v的第I个类型的值。如果I大于等于Variant的类型数量，该接口不可用；如果Variant不含值，则抛异常。	
参数（IN）	v	Variant右值对象
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	const ara::core::variant_alternative_t<I, ara::core::Variant<Types...>>&&	Variant中存储的第I个类型对象的const右值引用。
使用说明	无	
注意事项	无	

函数原型	template<typename T, typename... Types > constexpr T& get(ara::core::Variant<Types...> &v);	
函数功能	获取v的T类型的值。如果Variant中T类型不止出现一次，该接口不可用；如果Variant中不包含T类型则抛出异常。	
参数（IN）	v	Variant对象
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&	Variant中存储的T类型对象的引用。
使用说明	无	
注意事项	无	

函数原型	template <typename T, typename... Types> constexpr T&& get(ara::core::Variant<Types...>&& v);	
函数功能	获取v的T类型的值。如果Variant中T类型不止出现一次，该接口不可用；如果Variant中不包含T类型则抛出异常。	
参数（IN）	v	Variant右值对象
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	T&&	Variant中存储的T类型对象的右值引用。
使用说明	无	
注意事项	无	

函数原型	template<typename T, typename... Types > constexpr const T& get(ara::core::Variant<Types...> const &v);	
函数功能	获取v的T类型的值。如果Variant中T类型不止出现一次，该接口不可用；如果Variant中不包含T类型则抛出异常。	
参数（IN）	v	Variant对象
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	const T&	Variant中存储的T类型对象的const引用。
使用说明	无	
注意事项	无	

函数原型	template<typename T, typename... Types > constexpr const T&& get(const ara::core::Variant<Types...>&& v);	
函数功能	获取v的T类型的值。如果Variant中T类型不止出现一次，该接口不可用；如果Variant中不包含T类型则抛出异常。	
参数（IN）	v	Variant右值对象
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	const T&&	Variant中存储的T类型对象的const右值引用。
使用说明	无	
注意事项	无	

函数原型	template <size_t I, typename... Types> constexpr std::add_pointer_t<variant_alternative_t<I, Variant<Types...>>> get_if(Variant<Types...>* v) noexcept;	
函数功能	获取v的第I个类型的值的指针。如果I大于等于Variant中类型数量，该接口不可用。	
参数（IN）	v	Variant指针
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	std::add_pointer_t<variant_alternative_t<I, Variant<Types...>>>	如果存在，返回Variant中存储的第I个类型对象的指针；不存在返回nullptr。
使用说明	无	
注意事项	无	

函数原型	template <size_t I, typename... Types> constexpr std::add_pointer_t<const variant_alternative_t<I, Variant<Types...>>> get_if(const Variant<Types...>* v) noexcept;	
函数功能	获取v的第I个类型的值的指针。如果I大于等于Variant中类型数量，该接口不可用。	
参数（IN）	v	Variant指针
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	std::add_pointer_t<const variant_alternative_t<I, Variant<Types...>>>	如果存在，返回Variant中存储的第I个类型const对象的指针；不存在返回nullptr。
使用说明	无	
注意事项	无	

函数原型	template <typename T, typename... Types> constexpr std::add_pointer_t<T> get_if(Variant<Types...>* v) noexcept;	
函数功能	获取v的T类型的值的指针。如果T在Variant中不止出现一次，该接口不可用。	
参数（IN）	v	Variant指针
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	std::add_pointer_t<T>	如果存在，返回Variant中存储的T类型对象的指针；不存在返回nullptr。
使用说明	无	
注意事项	无	

函数原型	template <class T, class... Types> constexpr std::add_pointer_t<const T> get_if(const Variant<Types...>* v) noexcept;

函数功能	获取v的T类型的值的指针。如果T在Variant中不止出现一次，该接口不可用。	
参数（IN）	v	Variant指针
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	std::add_pointer_t<const T>	如果存在，返回Variant中存储的T类型const对象的指针；不存在返回nullptr。
使用说明	无	
注意事项	无	

函数原型	template <typename... Types> constexpr bool operator==(const Variant<Types...>& v, const Variant<Types...>& w);	
函数功能	比较两个Variant对象。	
参数（IN）	v	一个Variant对象。
	w	一个Variant对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	若v.index() != w.index()返回false，否则： v.valueless_by_exception()返回true，否则： 返回v所存值是否等于w所存值。
使用说明	无	
注意事项	无	

函数原型	template <typename... Types> constexpr bool operator!=(const Variant<Types...>& v, const Variant<Types...>& w);	
函数功能	比较两个Variant对象。	
参数（IN）	v	一个Variant对象。

	w	一个Variant对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	若v.index() != w.index()返回true，否则： v.valueless_by_exception()返回false，否则： 返回v所存值是否不等于w所存值。
使用说明	无	
注意事项	无	

函数原型	template <typename... Types> constexpr bool operator<(const Variant<Types...>& v, const Variant<Types...>& w);	
函数功能	比较两个Variant对象。	
参数（IN）	v	一个Variant对象。
	w	一个Variant对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	若w.valueless_by_exception()返回false，否则： v.valueless_by_exception()返回true，否则： v.index() < w.index()返回true，否则： v.index() > w.index()返回false，否则： 返回v所存值是否小于w所存值。
使用说明	无	
注意事项	无	

函数原型	template <typename... Types> constexpr bool operator<=(const Variant<Types...>& v, const Variant<Types...>& w);	
函数功能	比较两个Variant对象。	
参数（IN）	v	一个Variant对象。
	w	一个Variant对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	若v.valueless_by_exception()返回true，否则： w.valueless_by_exception()返回false，否则： v.index() < w.index()返回true，否则： v.index() > w.index()返回false，否则： 返回v所存值是否小于等于w所存值。
使用说明	无	
注意事项	无	

函数原型	template <typename... Types> constexpr bool operator>(const Variant<Types...>& v, const Variant<Types...>& w);	
函数功能	比较两个Variant对象。	
参数（IN）	v	一个Variant对象。
	w	一个Variant对象。
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	bool	若v.valueless_by_exception()返回false，否则： w.valueless_by_exception()返回true，否则： v.index() > w.index()返回true，否则： v.index() < w.index()返回false，否则： 返回v所存值是否大于w所存值。
使用说明	无	
注意事项	无	

函数原型	template <typename... Types> constexpr bool operator>=(const Variant<Types...>& v, const Variant<Types...>& w);	
函数功能	比较两个Variant对象。	
参数（IN）	v	一个Variant对象。
	w	一个Variant对象。
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	若w.valueless_by_exception()返回true，否则： v.valueless_by_exception()返回false，否则： v.index() > w.index()返回true，否则： v.index() < w.index()返回false，否则： 返回v所存值是否大于等于w所存值。
使用说明	无	
注意事项	无	

4 EM

4.1 ReportExecutionState

函数原型	ara::core::Result<void> ExecutionClient::ReportExecutionState(ExecutionState state) const noexcept;	
函数功能	应用程序上报进程运行状态	
参数（IN）	State	应用程序要上报的状态，包括Running和Terminating两种
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	ara::core::Result<void>	函数执行结果： void：函数执行成功 ara::exec::ExecErrc::kGeneralError：其他错误
使用说明	应用程序调用此接口上报当前的状态。 1. 进程在完成启动后应调用ReportExecutionState接口上报ara::exec::ExecutionState::kRunning表明已成功启动。若不调用ReportExecutionState接口上报ara::exec::ExecutionState::kRunning则此进程会被关闭，若配置重试则会再次重启，若无重试，则进程启动失败。 2. 进程应在接收SIGTERM的关闭请求后，通过ReportExecutionState接口上报ara::exec::ExecutionState::kTerminating表明进程正在关闭，然后执行关闭进程的操作；若在进程关闭超时时间内进程未完成关闭，EM调用系统调用强行KILL此进程。 3. 进程在未收到关闭请求的情况下，也可在通过ReportExecutionState接口上报ara::exec::ExecutionState::kTerminating后关闭；若在进程关闭超时时间内进程未完成关闭，EM调用系统调用强行KILL此进程。 4. 当接口返回ara::exec::ExecErrc::kGeneralError时需要进行错误处理，如延时一定时间（建议1S及以上）后重新调用此接口进行重试上报状态或者其他故障处理。	
头文件	#include "ara/exec/execution_client.h"	
注意事项	● 不可重入 ● 该接口为同步调用 ● EM会控制此接口的流量值，EM每秒收到的所有APP上报状态的次数之和不得超过150次。如果超出该值，则超出的上报数据包会被丢弃，APP上报状态失败，需要自行决定是否重试上报 ● 调用该接口的最大超时时间不超过2秒，如果2秒内该接口仍未完成上报，则直接返回错误	

4.2 SetState

函数原型	static ara::core::Result<FunctionGroup::CtorToken> FunctionGroup::Preconstruct(ara::core::StringView metaModelIdentifier) noexcept;	
函数功能	预构建一个可用于生成FunctionGroup的Token	
参数（IN）	ara::core::StringView metaModelIdentifier	标识FunctionGroup名字的字符串
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	ara::core::Result<FunctionGroup::CtorToken>	函数执行结果：返回值可用于构建FunctionGroup类
使用说明	调用此接口生成一个可以用来构建FunctionGroup类的Token	
头文件	#include "ara/exec/function_group.h"	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

函数原型	ara::exec::FunctionGroup::FunctionGroup(FunctionGroup::CtorToken &&token) noexcept;	
函数功能	从FunctionGroup::CtorToken生成FunctionGroup类的实例	
参数（IN）	FunctionGroup::CtorToken &&token	FunctionGroup::Preconstruct函数返回值中的token
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	函数执行结果：本函数为FunctionGroup类的构造函数
使用说明	调用此函数从FunctionGroup::CtorToken生成FunctionGroup类的实例	
头文件	#include "ara/exec/function_group.h"	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

函数原型	bool ara::exec::FunctionGroup::operator== (FunctionGroup const &other) const noexcept;	
函数功能	提供相等比较操作符与其他FunctionGroup类的实例进行比较	
功能安全等级	不涉及	
参数（IN）	FunctionGroup const &other	FunctionGroup类的实例
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	bool	函数执行结果： true：表示两个FunctionGroup类的实例内容相同 false：表示两个FunctionGroup类的实例内容不相同
使用说明	使用==操作符对FunctionGroup类的实例进行比较	
头文件	#include "ara/exec/function_group.h"	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

函数原型	bool ara::exec::FunctionGroup::operator!= (FunctionGroup const &other) const noexcept;	
函数功能	提供不相等比较操作符与其他FunctionGroup类的实例进行比较	
功能安全等级	不涉及	
参数（IN）	FunctionGroup const &other	FunctionGroup类的实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	函数执行结果： true：表示两个FunctionGroup类的实例内容不相同 false：表示两个FunctionGroup类的实例内容相同
使用说明	使用!=操作符对FunctionGroup类的实例进行比较	
头文件	#include "ara/exec/function_group.h"	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

函数原型	static ara::core::Result<FunctionGroupState::CtorToken> FunctionGroupState::Preconstruct(FunctionGroup const &functionGroup, ara::core::StringView metaModelIdentifier) noexcept;	
函数功能	预构建一个可用于生成FunctionGroupState的Token	
参数（IN）	FunctionGroup const &functionGroup	FunctionGroup类的实例

参数（IN）	ara::core::StringView metaModelIdentifier	标识FunctionGroup目标状态名字的字符串
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Result<FunctionGroupState::CtorToken>	函数执行结果：返回值可用于构建FunctionGroupState类
使用说明	调用此接口生成一个可以用来构建FunctionGroupState类的Token	
头文件	#include "ara/exec/function_group_state.h"	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

函数原型	ara::exec::FunctionGroupState::FunctionGroupState(FunctionGroupState::CtorToken &&token) noexcept;	
函数功能	从FunctionGroupState::CtorToken生成FunctionGroupState类的实例	
参数（IN）	FunctionGroupState::CtorToken &&token	FunctionGroupState::Preconstruct函数返回值中的token
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	函数执行结果：本函数为FunctionGroupState类的构造函数
使用说明	调用此函数从FunctionGroupState::CtorToken生成FunctionGroupState类的实例	
头文件	#include "ara/exec/function_group_state.h"	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

函数原型	bool ara::exec::FunctionGroupState::operator==(FunctionGroupState const &other) const noexcept;
函数功能	提供相等比较操作符与其他FunctionGroupState类的实例进行比较
功能安全等级	不涉及

参数（IN）	FunctionGroupState const &other	FunctionGroupState类的实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	函数执行结果： true：表示两个FunctionGroupState类的实例内容相同 false：表示两个FunctionGroupState类的实例内容不相同
使用说明	使用==操作符对FunctionGroupState类的实例进行比较	
头文件	#include "ara/exec/function_group_state.h"	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为同步调用	

函数原型	bool ara::exec::FunctionGroupState::operator!=(FunctionGroupState const &other) const noexcept;	
函数功能	提供不相等比较操作符与其他FunctionGroupState类的实例进行比较	
功能安全等级	不涉及	
参数（IN）	FunctionGroupState const &other	FunctionGroupState类的实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	函数执行结果： true：表示两个FunctionGroupState类的实例内容不相同 false：表示两个FunctionGroupState类的实例内容相同
使用说明	使用!=操作符对FunctionGroupState类的实例进行比较	
头文件	#include "ara/exec/function_group_state.h"	

注意事项	● 可重入（表示线程安全概念不包括信号） ● 该接口为同步调用

函数原型	ara::core::Future<void> StateClient::SetState (FunctionGroupState const &state) const noexcept;	
函数功能	向EM发送状态切换请求	
参数（IN）	FunctionGroupState	指向FunctionGroupState的实例
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Future<void>	函数执行结果： void：函数执行成功 ara::exec::ExecErrc::kCancelled：该操作被取消 ara::exec::ExecErrc::kFailed：操作失败 ara::exec::ExecErrc::kInvalidArguments：参数错误 ara::exec::ExecErrc::kCommunicationError：通信错误 ara::exec::ExecErrc::kGeneralError：其他错误
使用说明	调用此接口请求EM设置FunctionGroup到指定目标状态，使用前需要依次调用FunctionGroup::Preconstruct()、FunctionGroup::FunctionGroup()、FunctionGroupState::Preconstruct()、FunctionGroupState::FunctionGroupState()函数来创建FunctionGroupState的实例。	
头文件	#include "ara/exec/state_client.h"	
注意事项	● 可重入（表示线程安全概念不包括信号） ● 该接口为异步调用 ● State Management连续状态切换请求时，在上一个功能组状态切换请求未完成之前，再次请求同一功能组的状态切换，此时EM会取消前一次的请求，执行此次状态切换请求。前一次的状态切换请求返回结果为被取消	

4.3 GetInitialMachineStateTransitionResult

函数原型	ara::core::Future<void> StateClient::GetInitialMachineStateTransitionResult() const noexcept;	
函数功能	获取系统启动结果	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Future<void>	函数执行结果： void：函数执行成功 ara::exec::ExecErrc::kCancelled：启动操作被取消 ara::exec::ExecErrc::kFailed：失败 ara::exec::ExecErrc::kCommunication Error：通信错误 ara::exec::ExecErrc::kGeneralError：其他错误
使用说明	当EM随系统启动后，通过调用此接口可以获取EM启动的结果	
头文件	#include "ara/exec/state_client.h"	
注意事项	• 可重入（表示线程安全概念不包括信号） • 该接口为异步调用	

4.4 ExecutionClient

函数原型	ExecutionClient::ExecutionClient(void);
函数功能	EM客户端构造函数，提供给每个EM管理的APP使用
返回值	-
使用说明	此接口提供给app使用，调用此接口构造EM客户端，后续可以调用此类上报接口。
头文件	#include "ara/exec/execution_client.h"
注意事项	• 不可重入 • 该函数为同步调用

4.5 StateClient

函数原型	StateClient::StateClient(void);
函数功能	EM客户端构造函数，提供给SM使用。
返回值	-
使用说明	EM客户端，提供给SM使用
头文件	#include "ara/exec/state_client.h"
注意事项	• 不可重入 • 该函数为同步调用

4.6 Message

函数原型	char const *Message(ara::core::ErrorDomain::CodeType errorCode) const noexcept override;	
函数功能	返回错误码相关的信息	
参数（IN）	errorCode	错误码
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	char const *	错误码相关信息
使用说明	调用此接口得到相关错误码的信息	
头文件	#include <ara/exec/exec_error_domain.h>	
注意事项	• 不可重入 • 该接口为同步接口	

4.7 ThrowAsException

函数原型	void ThrowAsException(ara::core::ErrorCode const &errorCode) const noexcept(false) override	
函数功能	根据错误码创建ExecException实例，并作为c++异常抛出	
参数（IN）	errorCode	要被throw的错误

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	传入错误码创建ExecException实例，并作为c++异常抛出	
头文件	#include <ara/exec/exec_error_domain.h>	
注意事项	• 不可重入 • 该接口为同步接口	

4.8 MakeErrorCode

函数原型	constexpr ara::core::ErrorCode MakeErrorCode (ara::exec::ExecErrc code, ara::core::ErrorDomain::SupportDataType data = 0) noexcept;	
函数功能	创建ErrorCode实例	
参数（IN）	code	错误码
参数（IN）	data	厂商定义的与错误相关的数据
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::ErrorCode	错误码对象
使用说明	传入exec错误码和厂商定义的与错误相关的数据，创建ErrorCode实例	
头文件	#include <ara/exec/exec_error_domain.h>	
注意事项	• 不可重入 • 该接口为同步接口	

4.9 GetExecErrorDomain

函数原型	constexpr ara::core::ErrorDomain const &GetExecErrorDomain() noexcept;	
函数功能	返回全局ExecErrorDomain对象的引用	
参数（IN）	None	-

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::ErrorDomain const &	全局ExecErrorDomain对象的引用
使用说明	返回全局PhmErrorDomain对象的引用	
头文件	#include <ara/exec/exec_error_domain.h>	
注意事项	• 不可重入 • 该接口为同步接口	

4.10 GetState

函数原型	ara::core::Future<void> StateClient::GetState (ara::core::String const &functionGroup, ara::core::String &state) const noexcept;	
函数功能	向EM发送获取功能组状态请求	
参数（IN）	functionGroup	对应的功能组名称
参数（INOUT）	state	用于接收对应功能组状态
参数（OUT）	None	-
返回值	ara::core::Future<void>	函数执行结果： void：函数执行成功 ara::exec::ExecErrc::kFailed：功能组状态转换失败 ara::exec::ExecErrc::kInvalidArguments：参数错误 ara::exec::ExecErrc::kCommunicationError：通信相关错误 ara::exec::ExecErrc::kBusy：功能组正在转换中 ara::exec::ExecErrc::kGeneralError：其他错误
使用说明	调用此接口向EM请求对应功能组的状态	
头文件	#include "ara/exec/state_client.h"	

注意事项	• 可重入 • 该接口为异步调用 • 支持10次/秒最大查询频率

5 LOG

5.1 数据类型

5.1.1 LogLevel

类名	ara::log::LogLevel
命名空间	namespace ara::log

语法	enum class LogLevel : uint8_t {
	kOff = 0x00,
	kFatal = 0x01,
	kError = 0x02,
	kWarn = 0x03,
	kInfo = 0x04,
	kDebug = 0x05,
	kVerbose = 0x06
	};

头文件	#include "ara/log/common.h"

说明	定义日志严重性级别： kOff：不记录日志。 kFatal：致命错误，不可恢复。 kError：影响系统功能正常运行的错误。 kWarn：若无法确保运行是否正确，则发出警告。 kInfo：提供系统运行的具体信息。 kDebug：向程序员提供详细的调试信息。 kVerbose：最为详细的系统试信息。

5.1.2 LogMode

类名	ara::log::LogMode
命名空间	namespace ara::log

语法	enum class LogMode : uint8_t {
	kRemote = 0x01,
	kFile = 0x02,
	kConsole = 0x04
	};

头文件	#include "ara/log/common.h"
说明	定义日志输出模式，用于配置日志消息的接收方式： kRemote：发送远端。 kFile：保存文件。 kConsole：控制台输出。 可以使用组合方式定义日志输出模式。

5.1.3 LogHex8

类名	ara::log::LogHex8
命名空间	namespace ara::log

语法	struct LogHex8 {
	uint8_t value;
	};

头文件	#include "ara/log/log_stream.h"
说明	表示8位十六进制值数据类型。 作为自定义类型的辅助结构，保存一个整型数据。

5.1.4 LogHex16

类名	ara::log::LogHex16
命名空间	namespace ara::log

语法	struct LogHex16 {
	uint16_t value;
	};

头文件	#include "ara/log/log_stream.h"
说明	表示16位十六进制值数据类型。

5.1.5 LogHex32

类名	ara::log::LogHex32
命名空间	namespace ara::log

语法	struct LogHex32 {
	uint32_t value;
	};

头文件	#include "ara/log/log_stream.h"
说明	表示32位十六进制值数据类型。

5.1.6 LogHex64

类名	ara::log::LogHex64
命名空间	namespace ara::log

语法	struct LogHex64 {
	uint64_t value;
	};

头文件	#include "ara/log/log_stream.h"
说明	表示64位十六进制值数据类型。

5.1.7 LogBin8

类名	ara::log::LogBin8

命名空间	namespace ara::log

语法	struct LogBin8 {
	uint8_t value;
	};

头文件	#include "ara/log/log_stream.h"
说明	表示8位二进制值数据类型。

5.1.8 LogBin16

类名	ara::log::LogBin16
命名空间	namespace ara::log

语法	struct LogBin16 {
	uint16_t value;
	};

头文件	#include "ara/log/log_stream.h"
说明	表示16位二进制值数据类型。

5.1.9 LogBin32

类名	ara::log::LogBin32
命名空间	namespace ara::log

语法	struct LogBin32 {
	uint32_t value;
	};

头文件	#include "ara/log/log_stream.h"
说明	表示32位二进制值数据类型。

5.1.10 LogBin64

类名	ara::log::LogBin64
命名空间	namespace ara::log

语法	struct LogBin64 {
	uint64_t value;
	};

头文件	#include "ara/log/log_stream.h"
说明	表示64位二进制值数据类型。

5.1.11 ClientState

类名	ara::log::ClientState
命名空间	namespace ara::log

语法	enum class ClientState : int8_t {
	kUnknown = -1,
	kNotConnected,
	kConnected
	};

头文件	#include "ara/log/common.h"
说明	表示外部客户端的连接状态： kUnknown：未知连接状态。 kNotConnected：未连接。 kConnected：已连接。

5.2 函数接口

5.2.1 Initialize

函数原型	ara::log::Initialize()	
函数功能	初始化LOG组件，通过解析配置文件（如用户配置的LogLevel、LogMode等信息），并调用InitLogging接口，完成Application的日志级别、日志模式等信息的设置。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	为空，当出现错误时返回LogErrc中定义的错误。

使用说明	● 在执行LOG其他操作前，必须初始化LOG组件。 ● Initialize和InitLogging接口差异说明： – Initialize接口通过解析日志配置文件，并调用InitLogging完成LOG组件初始化，其中，日志配置文件由用户通过AOS Suite工具指定。 – InitLogging接口仅通过接收函数入参的方式实现LOG组件初始化。 用户可根据应用场景选择初始化方式，推荐优先使用Initialize接口。
头文件	#include "ara/log/log_init.h"
注意事项	● 不可重入 ● 该接口为同步接口

5.2.2 Deinitialize

函数原型	ara::log::Deinitialize()	
函数功能	LOG组件去初始化	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	为空，当出现错误时返回LogErrc中定义的错误。
使用说明	执行去初始化操作后，不得执行任何LOG其他操作，除Initialize初始化接口。	
头文件	#include "ara/log/log_init.h"	
注意事项	● 不可重入 ● 该接口为同步接口	

5.2.3 InitLogging

函数原型	void ara::log::InitLogging(std::string appId, std::string appDescription, LogLevel appDefLogLevel, LogMode logMode, std::string directoryPath = "") noexcept;
函数功能	初始化LOG组件，通过接收用户传入的Application ID、LogLevel等信息，完成Application的日志级别等信息的设置。

参数（IN）	appId	应用ID，至少1个字符，至多4个字符
	appDescription	应用详细描述
	appDefLogLevel	默认的日志级别
	logMode	日志模式，可使用Console、File和Remote几种模式的组合。 ● Console：日志和跟踪消息输出到控制台； ● File：日志和跟踪消息输出到文件； ● Remote：日志和跟踪消息通过通信总线输出。
	directoryPath	日志文件存储路径
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	● 在执行LOG其他操作前，必须初始化LOG组件。 ● Initialize和InitLogging接口差异说明： – Initialize接口通过解析日志配置文件，并调用InitLogging完成LOG组件初始化，其中，日志配置文件由用户通过AOS Suite工具指定。 – InitLogging接口仅通过接收函数入参的方式实现LOG组件初始化。 用户可根据应用场景选择初始化方式，推荐优先使用Initialize接口。	
头文件	#include "ara/log/log_init.h"	
注意事项	● 不可重入 ● 该接口为同步接口	

5.2.4 CreateLogger

函数原型	Logger& CreateLogger(ara::core::StringView ctxId, ara::core::StringView ctxDescription, LogLevel ctxDefLogLevel = LogLevel::kWarn) noexcept;	
函数功能	创建日志Context的上下文	
参数（IN）	ctxId	日志上下文ID

	ctxDescription	日志上下文描述
	ctxDefLogLevel	默认的日志级别，若未指定，则默认为Warning
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	Logger &	内部管理的Logger对象实例的引用。对象实例的管理在Logging框架内进行。
使用说明	日志记录前，必须创建Logger对象，由Logger对象提供日志记录功能。	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.5 HexFormat (uint8)

函数原型	LogHex8 HexFormat(uint8_t value) noexcept;	
函数功能	将uint8类型的参数转换为十六进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogHex8	转换后的十六进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.6 HexFormat (int8)

函数原型	LogHex8 HexFormat(int8_t value) noexcept;	
函数功能	将int8类型的参数转换为十六进制值	
参数（IN）	value	被转换的十进制数

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogHex8	转换后的十六进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.7 HexFormat (uint16)

函数原型	LogHex16 HexFormat(uint16_t value) noexcept;	
函数功能	将uint16类型的参数转换为十六进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogHex16	转换后的十六进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.8 HexFormat (int16)

函数原型	LogHex16 HexFormat(int16_t value) noexcept;	
函数功能	将int16类型的参数转换为十六进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogHex16	转换后的十六进制数
使用说明	无	

头文件	#include "ara/log/logging.h"
注意事项	• 可重入 • 该接口为同步接口

5.2.9 HexFormat (uint32)

函数原型	LogHex32 HexFormat(uint32_t value) noexcept;	
函数功能	将uint32类型的参数转换为十六进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogHex32	转换后的十六进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.10 HexFormat (int32)

函数原型	LogHex32 HexFormat(int32_t value) noexcept;	
函数功能	将int32类型的参数转换为十六进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogHex32	转换后的十六进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.11 HexFormat (uint64)

函数原型	LogHex64 HexFormat(uint64_t value) noexcept;	
函数功能	将uint64类型的参数转换为十六进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogHex64	转换后的十六进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.12 HexFormat (int64)

函数原型	LogHex64 HexFormat(int64_t value) noexcept;	
函数功能	将int64类型的参数转换为十六进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogHex64	转换后的十六进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.13 BinFormat (uint8)

函数原型	LogBin8 BinFormat(uint8_t value) noexcept;	
函数功能	将uint8类型的参数转换为二进制值	
参数（IN）	value	被转换的十进制数

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogBin8	转换后的二进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.14 BinFormat (int8)

函数原型	LogBin8 BinFormat(int8_t value) noexcept;	
函数功能	将int8类型的参数转换为二进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogBin8	转换后的二进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.15 BinFormat (uint16)

函数原型	LogBin16 BinFormat(uint16_t value) noexcept;	
函数功能	将uint16类型的参数转换为二进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogBin16	转换后的二进制数
使用说明	无	

头文件	#include "ara/log/logging.h"
注意事项	• 可重入 • 该接口为同步接口

5.2.16 BinFormat (int16)

函数原型	LogBin16 BinFormat(int16_t value) noexcept;	
函数功能	将int16类型的参数转换为二进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogBin16	转换后的二进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.17 BinFormat (uint32)

函数原型	LogBin32 BinFormat(uint32_t value) noexcept;	
函数功能	将uint32类型的参数转换为二进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogBin32	转换后的二进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.18 BinFormat (int32)

函数原型	LogBin32 BinFormat(int32_t value) noexcept;	
函数功能	将int32类型的参数转换为二进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogBin32	转换后的二进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.19 BinFormat (uint64)

函数原型	LogBin64 BinFormat(uint64_t value) noexcept;	
函数功能	将uint64类型的参数转换为二进制值	
参数（IN）	value	被转换的十进制数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogBin64	转换后的二进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.20 BinFormat (int64)

函数原型	LogBin64 BinFormat(int64_t value) noexcept;	
函数功能	将int64类型的参数转换为二进制值	
参数（IN）	value	被转换的十进制数

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogBin64	转换后的二进制数
使用说明	无	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.21 RawBuffer

函数原型	LogRawBuffer RawBuffer(const T &value) noexcept;	
函数功能	通过buffer记录原始的二进制数据	
参数（IN）	value	被转换的raw data的数值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogRawBuffer	转换后的raw buffer
使用说明	传入数值参数，将其转换为LogRawBuffer	
头文件	#include "ara/log/logging.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.2.22 remoteClientState

函数原型	ClientState remoteClientState() noexcept;	
函数功能	从远程客户端的DLT后端获取连接状态	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ClientState	当前的客户端状态
使用说明	无	

头文件	#include "ara/log/logging.h"
注意事项	• 可重入 • 该接口为同步接口

5.3 LogStream Class

ara::log::LogStream类表示一个日志消息流，允许使用流操作符来添加数据。

5.3.1 LogStream::Flush

函数原型	void Flush() noexcept;	
函数功能	发送当前日志缓冲区并启动新的消息流。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.2 LogStream::operator<<(bool value)

函数原型	LogStream &operator <<(bool value) noexcept;	
函数功能	将给定的数据写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	

注意事项	• 可重入 • 该接口为同步接口

5.3.3 LogStream::operator<<(uint8_t value)

函数原型	LogStream &operator <<(uint8_t value) noexcept;	
函数功能	将无符号8位整型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.4 LogStream::operator<<(uint16_t value)

函数原型	LogStream &operator <<(uint16_t value) noexcept;	
函数功能	将无符号16位整型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.5 LogStream::operator<<(uint32_t value)

函数原型	LogStream &operator <<(uint32_t value) noexcept;	
函数功能	将无符号32位整型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.6 LogStream::operator<<(uint64_t value)

函数原型	LogStream &operator <<(uint64_t value) noexcept;	
函数功能	将无符号64位整型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.7 LogStream::operator<<(int8_t value)

函数原型	LogStream &operator <<(int8_t value) noexcept;	
函数功能	将有符号8位整型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.8 LogStream::operator<<(int16_t value)

函数原型	LogStream &operator <<(int16_t value) noexcept;	
函数功能	将有符号16位整型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.9 LogStream::operator<<(int32_t value)

函数原型	LogStream &operator <<(int32_t value) noexcept;	
函数功能	将有符号32位整型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	

头文件	#include "ara/log/log_stream.h"
注意事项	• 可重入 • 该接口为同步接口

5.3.10 LogStream::operator<<(int64_t value)

函数原型	LogStream &operator <<(int64_t value) noexcept;	
函数功能	将有符号64位整型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.11 LogStream::operator<<(float value)

函数原型	LogStream &operator <<(float value) noexcept;	
函数功能	将32位浮点型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.12 LogStream::operator<<(double value)

函数原型	LogStream &operator <<(double value) noexcept;	
函数功能	将64位浮点型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.13 LogStream::operator<<(const LogRawBuffer &value)

函数原型	LogStream &operator <<(const LogRawBuffer &value) noexcept;	
函数功能	将二进制数据写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.14 LogStream::operator<<(const LogHex8 &value)

函数原型	LogStream &operator <<(const LogHex8 &value) noexcept;
函数功能	将无符号整型参数以8位的十六进制写入到消息缓冲区中。

参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.15 LogStream::operator<<(const LogHex16 &value)

函数原型	LogStream &operator <<(const LogHex16 &value) noexcept;	
函数功能	将无符号整型参数以16位的十六进制形式写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.16 LogStream::operator<<(const LogHex32 &value)

函数原型	LogStream &operator <<(const LogHex32 &value) noexcept;	
函数功能	将无符号整型参数以32位的十六进制形式写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this

使用说明	无
头文件	#include "ara/log/log_stream.h"
注意事项	• 可重入 • 该接口为同步接口

5.3.17 LogStream::operator<<(const LogHex64 &value)

函数原型	LogStream &operator <<(const LogHex64 &value) noexcept;	
函数功能	将无符号整型参数以64位的十六进制形式写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.18 LogStream::operator<<(const LogBin8 &value)

函数原型	LogStream &operator <<(const LogBin8 &value) noexcept;	
函数功能	将无符号整型参数以8位的二进制形式写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.19 LogStream::operator<<(const LogBin16 &value)

函数原型	LogStream &operator <<(const LogBin16 &value) noexcept;	
函数功能	将无符号整型参数以16位的二进制形式写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.20 LogStream::operator<<(const LogBin32 &value)

函数原型	LogStream &operator <<(const LogBin32 &value) noexcept;	
函数功能	将无符号整型参数以32位的二进制形式写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.21 LogStream::operator<<(const LogBin64 &value)

函数原型	LogStream &operator <<(const LogBin64 &value) noexcept;	
函数功能	将无符号整型参数以64位的二进制形式写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.22 LogStream::operator<<(const ara::core::StringView value)

函数原型	LogStream &operator <<(const ara::core::StringView value) noexcept;	
函数功能	将ara::core::StringView类型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.23 LogStream::operator<<(const ara::core::String &value)

函数原型	LogStream &operator<<(const ara::core::String &value) noexcept;	
函数功能	将ara::core::String类型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA

参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.24 LogStream::operator<<(const std::string &value)

函数原型	LogStream &operator<<(const std::string &value) noexcept;	
函数功能	将std::string类型参数写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.25 LogStream::operator<<(const char *const value)

函数原型	LogStream &operator<<(const char * const value) noexcept;	
函数功能	将UTF8格式字符串写入到消息缓冲区中。	
参数（IN）	value	追加到消息缓冲区中的值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	

注意事项	• 可重入 • 该接口为同步接口

5.3.26 operator<<(LogStream &out, LogLevel value)

函数原型	LogStream &operator <<(LogStream &out, LogLevel value) noexcept;	
函数功能	将LogLevel枚举参数以字符串形式写入到消息缓冲区中。	
参数（IN）	out	需要添加LogLevel值的LogStream对象
	value	LogLevel枚举参数转为字符串追加到消息缓冲区中
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this
使用说明	无	
头文件	#include "ara/log/log_stream.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.3.27 operator<<(LogStream &out, const core::ErrorCode &ec)

函数原型	LogStream &operator<<(LogStream &out, const ara::core::ErrorCode &value) noexcept;	
函数功能	将ara::core::ErrorCode错误实例写入到消息缓冲区中。	
参数（IN）	out	需要添加ErrorCode内容的LogStream对象
	ec	将ErrorCode错误实例追加到消息缓冲区中
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream &	*this

使用说明	当输出到控制台时，ErrorCode应以实现定义的方式显示为一个字符串，其中包含ErrorCode:Domain().Name()的结果（即ErrorDomain 的简称），以及完整的错误代码编号。
头文件	#include "ara/log/log_stream.h"
注意事项	• 可重入 • 该接口为同步接口

5.4 Logger Class

Logger类表示一个日志记录器的上下文。 Logging框架定义了上下文含义，可以将其视为一个应用程序进程或在进程一定范围内的记录器实例。

5.4.1 Logger::LogFatal()

函数原型	LogStream LogFatal() noexcept;	
函数功能	创建一个LogStream对象。 返回的对象可通过流运算符“<<”，接收参数。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream	Fatal级别的LogStream对象
使用说明	在正常使用场景中，创建的LogStream对象的生命周期限定在一个语句内（在传递最后一个参数后，以；为结束）。如果想要延长LogStream对象的生命周期，可以为该对象设置给一个变量名。	
头文件	#include "ara/log/logger.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.4.2 Logger::LogError()

函数原型	LogStream LogError() noexcept;	
函数功能	创建一个LogStream对象。 返回的对象可通过流运算符“<<”，接收参数。	
参数（IN）	None	NA

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream	Error级别的LogStream对象
使用说明	在正常使用场景中，创建的LogStream对象的生命周期限定在一个语句内（在传递最后一个参数后，以；为结束）。如果想要延长LogStream对象的生命周期，可以为该对象设置给一个变量名。	
头文件	#include "ara/log/logger.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.4.3 Logger::LogWarn()

函数原型	LogStream LogWarn() noexcept;	
函数功能	创建一个LogStream对象。 返回的对象可通过流运算符“<<”，接收参数。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream	Warn级别的LogStream对象
使用说明	在正常使用场景中，创建的LogStream对象的生命周期限定在一个语句内（在传递最后一个参数后，以；为结束）。如果想要延长LogStream对象的生命周期，可以为该对象设置给一个变量名。	
头文件	#include "ara/log/logger.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.4.4 Logger::LogInfo()

函数原型	LogStream LogInfo() noexcept;	
函数功能	创建一个LogStream对象。 返回的对象可通过流运算符“<<”，接收参数。	
参数（IN）	None	NA

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream	Info级别的LogStream对象
使用说明	在正常使用场景中，创建的LogStream对象的生命周期限定在一个语句内（在传递最后一个参数后，以；为结束）。如果想要延长LogStream对象的生命周期，可以为该对象设置给一个变量名。	
头文件	#include "ara/log/logger.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.4.5 Logger::LogDebug()

函数原型	LogStream LogDebug() noexcept;	
函数功能	创建一个LogStream对象。 返回的对象可通过流运算符“<<”，接收参数。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream	Debug级别的LogStream对象
使用说明	在正常使用场景中，创建的LogStream对象的生命周期限定在一个语句内（在传递最后一个参数后，以；为结束）。如果想要延长LogStream对象的生命周期，可以为该对象设置给一个变量名。	
头文件	#include "ara/log/logger.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.4.6 Logger::LogVerbose()

函数原型	LogStream LogVerbose() noexcept;	
函数功能	创建一个LogStream对象。 返回的对象可通过流运算符“<<”，接收参数。	
参数（IN）	None	NA

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	LogStream	Verbose级别的LogStream对象
使用说明	在正常使用场景中，创建的LogStream对象的生命周期限定在一个语句内（在传递最后一个参数后，以；为结束）。如果想要延长LogStream对象的生命周期，可以为该对象设置给一个变量名。	
头文件	#include "ara/log/logger.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.4.7 Logger::IsEnabled(LogLevel logLevel)

函数原型	bool IsEnabled(LogLevel logLevel) const noexcept;	
函数功能	核查当前配置的日志上报级别。 应用程序在执行密集日志的程序前，可能希望能够检查记录器logger实际配置的日志上报级别。	
参数（IN）	logLevel	待检查的日志级别
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	bool	如果待检查的日志级别满足配置的上报要求，则为真。
使用说明	无	
头文件	#include "ara/log/logger.h"	
注意事项	• 可重入 • 该接口为同步接口	

5.4.8 Logger::Unregister()

函数原型	void Unregister() noexcept;	
函数功能	去注册日志上下文。去注册后，不得再使用该logger继续记录日志。	
参数（IN）	None	NA

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
头文件	#include "ara/log/logger.h"	
注意事项	• 不可重入 • 该接口为同步接口	

5.4.9 Logger::GetId()

函数原型	const ara::core::String& GetId() const noexcept;	
函数功能	获取日志上下文的ID。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::String	日志上下文的ID
使用说明	无	
头文件	#include "ara/log/logger.h"	
注意事项	• 可重入 • 该接口为同步接口	

6 PHM

6.1 接口列表

表 6-1 接口列表

接口	简介	头文件
6.2 SupervisedEntity(ara::core::InstanceSpecifier const &instance)	类SupervisiedEntity的构造函数	ara/phm/supervised_entity.h
6.3 SupervisedEntity(const ara::core::InstanceSpecifier &instance, std::string const &processName, CommunicationMode const &communicationMode)	类SupervisiedEntity的构造函数	ara/phm/supervised_entity.h
6.4 ReportCheckpoint	类SupervisiedEntity实例上报Checkpoint	ara/phm/supervised_entity.h
6.5 GetLocalSupervisionStatus	类SupervisiedEntity实例获取其当前的Local Supervision Status	ara/phm/supervised_entity.h
6.6 GetGlobalSupervisionStatus	类SupervisiedEntity实例获取其当前的Glocal Supervision Status	ara/phm/supervised_entity.h
6.7 HealthChannel(ara::core::InstanceSpecifier const &instance)	类HealthChannel的构造函数	ara/phm/health_channel.h

接口	简介	头文件
6.8 HealthChannel(const ara::core::InstanceSpecifier &instance, std::string const &processName, CommunicationMode const &communicationMode)	类HealthChannel的构造函数	ara/phm/health_channel.h
6.9 ReportHealthStatus	类HealthChannel实例上报HealthStatus	ara/phm/health_channel.h
6.10 SetConcurrentNumber	设置处理上报信息的线程数量	ara/phm/phm_state_manager.h
6.11 GetConcurrentNumber	返回上报信息的线程数量	ara/phm/phm_state_manager.h
6.12 RegisterRecoveryCallback	注册第三方应用的回调函数	ara/phm/phm_state_manager.h
6.13 Init	初始化phm的监控服务	ara/phm/phm_state_manager.h
6.14 DeInit	取消初始化phm的监控服务	ara/phm/phm_state_manager.h
6.15 QueryRuleResults	查询故障发生的rule相关的status信息	ara/phm/phm_state_manager.h
6.16 ProcessNotifier	类ProcessNotifier构造函数	ara/phm/process_notifier.h
6.17 ProcessChanged	通知PHM进程状态发生了变化，供EM使用	ara/phm/process_notifier.h
6.18 RegisterProcessStatesCallback	注册获取进程状态回调函数，在没有EM的情况下，用于PHM服务端获取进程状态集合	ara/phm/phm_state_manager.h
6.19 SetProcessState	PHM服务端进程设置进程状态	ara/phm/phm_state_manager.h
6.20 PhmClientCfg::SetUser	设置shm文件所属用户	ara/phm/phm_client_cfg.h
6.21 PhmClientCfg::GetUser	返回shm文件所属用户	ara/phm/phm_client_cfg.h
6.22 PhmErrorDomain::Name	返回PhmErrorDomain的名称常量	ara/phm/phm_error_domain.h

接口	简介	头文件
6.23 PhmErrorDomain::Message	返回错误码相关的信息	ara/phm/phm_error_domain.h
6.24 PhmErrorDomain::ThrowAsException	根据错误码创建PhmException实例，并作为c++异常抛出	ara/phm/phm_error_domain.h
6.25 PhmException::PhmException	构造一个包含错误码的PhmException实例	ara/phm/phm_error_domain.h
6.26 GetPhmErrorDomain	返回全局PhmErrorDomain对象的引用	ara/phm/phm_error_domain.h
6.27 MakeErrorCode	创建ErrorCode实例	ara/phm/phm_error_domain.h

6.2 SupervisedEntity(ara::core::InstanceSpecifier const &instance)

函数原型	ara::phm::SupervisedEntity::SupervisedEntity(ara::core::InstanceSpecifier const &instance);	
函数功能	类SupervisiedEntity的构造函数	
参数（IN）	instance	SupervisiedEntity对应的实例说明符
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	应用程序调用该构造函数创建SupervisiedEntity对象，传入配置生成的InstanceSpecifier。	
头文件	#include <ara/phm/supervised_entity.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.3 SupervisedEntity(const ara::core::InstanceSpecifier &instance, std::string const &processName, CommunicationMode const &communicationMode)

函数原型	ara::phm::SupervisedEntity::SupervisedEntity(const ara::core::InstanceSpecifier &instance, std::string const &processName, CommunicationMode const &communicationMode);	
函数功能	类SupervisiedEntity的构造函数	
参数（IN）	instance	SupervisiedEntity对应的实例说明符
参数（IN）	processName	SupervisiedEntity对应的进程名称
参数（IN）	CommunicationMode	客户端使用的通信方式
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-

使用说明	应用程序调用该构造函数创建SupervisiedEntity对象，传入配置生成的InstanceSpecifier，进程名称和通信方式。
	支持设置客户端的底层通信为SHM&UDP，communicationMode如果传入为空集合，cm侧会校验失败
	using CommunicationMode = std::set<rtf::com::TransportMode>;
	enum class TransportMode : uint8_t {
	TCP = 0x00U,
	UDP = 0x01U,
	SHM = 0x02U,
	DSHM = 0X03U,
	ICC = 0x04U
	};
	除SHM&UDP外，其他TransportMode当前暂不支持。	

头文件	#include <ara/phm/supervised_entity.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.4 ReportCheckpoint

函数原型	ara::core::Result<void> SupervisedEntity::ReportCheckpoint (Checkpoint checkpointId);	
函数功能	类SupervisiedEntity实例上报Checkpoint	
参数（IN）	checkpointId	checkpoint标识符
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Result<void>	返回结果为空或者包含一个实现的具体错误码
使用说明	应用程序调用ReportCheckpoint接口，上报一个checkpoint的实例。 1. 当返回值Result的HasValue()为false时，需要进行错误处理，如延时一定时间（建议1S及以上）后重新调用此接口，进行重试上报状态或者其他故障处理。	
头文件	#include <ara/phm/supervised_entity.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.5 GetLocalSupervisionStatus

函数原型	ara::core::Result<LocalSupervisionStatus> SupervisedEntity::GetLocalSupervisionStatus () const;	
函数功能	类SupervisiedEntity实例获取其当前的Local Supervision Status	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Result<LocalSupervisionStatus>	返回Local Supervision Status

使用说明	SupervisiedEntity实例调用GetLocalSupervisionStatus接口，获取其所属的Local Supervision Status。
	LocalSupervisionStatus为枚举类：
	enum class LocalSupervisionStatus : uint32_t {
	K_DEACTIVATED = 1U, // Supervsion未激活
	K_OK = 2U, // Supervsion激活并且没有错误
	K_FAILED = 3U, // 在容忍限度内出错被检测
	K_EXPIRED = 4U // 在容忍限度外出错被检测
	};

头文件	#include <ara/phm/supervised_entity.h>
注意事项	• 不可重入 • 该接口为同步接口

6.6 GetGlobalSupervisionStatus

函数原型	ara::core::Result<GlobalSupervisionStatus> SupervisedEntity::GetGlobalSupervisionStatus () const;	
函数功能	类SupervisiedEntity实例获取其当前的Glocal Supervision Status	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Result<GlobalSupervisionStatus>	返回Glocal Supervision Status

使用说明	SupervisiedEntity实例调用GetGlobalSupervisionStatus 接口，获取其所属的Glocal Supervision Status。
	Glocal Supervision Status为枚举类：
	enum class GlobalSupervisionStatus : uint32_t {
	K_DEACTIVATED = 1U, // Supervsion未激活
	K_OK = 2U, // 所有的LocalSupervision为kOK或者kDeactivated
	K_FAILED = 3U, // 至少有一个LocalSupervision为kFailed,并且没有kExpired的
	K_EXPIRED = 4U, // 至少有一个LocalSupervision处于kExpired状态，但自达到kExpired以来的监督周期数未超过容忍
	K_STOPPED = 5U // 至少有一个LocalSupervision处于kExpired状态，但自达到kExpired以来的监督周期数超过容忍
	};	

头文件	#include <ara/phm/supervised_entity.h>	

注意事项	• 不可重入 • 该接口为同步接口

6.7 HealthChannel(ara::core::InstanceSpecifier const &instance)

函数原型	ara::phm::HealthChannel::HealthChannel(ara::core::InstanceSpecifier const &instance);	
函数功能	类HealthChannel的构造函数	
参数（IN）	instance	HealthChannel对应的实例说明符
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	应用程序调用该构造函数创建HealthChannel对象，传入配置生成的InstanceSpecifier。	
头文件	#include <ara/phm/health_channel.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.8 HealthChannel(const ara::core::InstanceSpecifier &instance, std::string const &processName, CommunicationMode const &communicationMode)

函数原型	ara::phm::HealthChannel::HealthChannel(const ara::core::InstanceSpecifier &instance, std::string const &processName, CommunicationMode const &communicationMode);	
函数功能	类HealthChannel的构造函数	
参数（IN）	instance	SupervisiedEntity对应的实例说明符
参数（IN）	processName	SupervisiedEntity对应的进程名称
参数（IN）	CommunicationMode	客户端使用的通信方式

参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-

使用说明	应用程序调用该构造函数创建HealthChannel对象，传入配置生成的InstanceSpecifier，进程名称和通信方式。
	支持设置客户端的底层通信为SHM&UDP，communicationMode如果传入为空集合，cm侧会校验失败
	using CommunicationMode = std::set<rtf::com::TransportMode>;
	enum class TransportMode : uint8_t {
	TCP = 0x00U,
	UDP = 0x01U,
	SHM = 0x02U,
	DSHM = 0X03U,
	ICC = 0x04U
	};
	除SHM&UDP外，其他TransportMode当前暂不支持。	

头文件	#include <ara/phm/health_channel.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.9 ReportHealthStatus

函数原型	ara::core::Result<void> HealthChannel::ReportHealthStatus (HealthStatus healthStatusId);	
函数功能	类HealthChannel实例上报HealthStatus	
参数（IN）	healthStatusId	HealthStatus标识符
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::Result<void>	返回结果为空或者包含一个实现的具体错误码
使用说明	应用程序调用ReportHealthStatus接口，上报一个HealthStatus的实例。 1. 当返回值Result的HasValue()为false时，需要进行错误处理，如延时一定时间（建议1S及以上）后重新调用此接口，进行重试上报状态或者其他故障处理。	

头文件	#include <ara/phm/health_channel.h>
注意事项	• 不可重入 • 该接口为同步接口

6.10 SetConcurrentNumber

函数原型	bool PhmStateManager::SetConcurrentNumber(uint32_t number);	
函数功能	设置处理上报信息的线程数量	
参数（IN）	number	线程数量
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	bool	返回结果，设置线程数是否成功
使用说明	调用SetConcurrentNumber来设置线程数量，传入的线程数应为1-64范围内的正整数。	
头文件	#include <ara/phm/phm_state_manager.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.11 GetConcurrentNumber

函数原型	bool PhmStateManager::GetConcurrentNumber();	
函数功能	返回上报信息的线程数量	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	uint32_t	线程数量
使用说明	调用GetConcurrentNumber返回线程数量	
头文件	#include <ara/phm/phm_state_manager.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.12 RegisterRecoveryCallback

函数原型	void PhmStateManager::RegisterRecoveryCallback(const std::function<void(std::string, std::string, std::string, std::string, std::string)> &cb);	
函数功能	注册第三方应用的回调函数	
参数（IN）	const std::function<void(std::string, std::string, std::string, std::string, std::string)> &cb	函数引用cb, 其包含5个参数，依次为 ruleName、actionName（AppPhmActionItem的名称）、portName、methodName、processName
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	void	-
使用说明	应用程序调用RegisterRecoveryCallback，注册回调函数，接受recovery action的相关配置信息，由回调函数来通过指定的process去执行相关的恢复动作。	
头文件	#include <ara/phm/phm_state_manager.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.13 Init

函数原型	bool PhmStateManager::Init(const std::string &user = "", const communicateMode& mode = {rtf::com::TransportMode::UDP, rtf::com::TransportMode::SHM});	
函数功能	初始化phm的监控服务	
参数（IN）	const std::string &user = ""	shm文件属主，默认值为""
	const communicateMode& mode = {rtf::com::TransportMode::UDP, rtf::com::TransportMode::SHM}	通信方式，默认为UDP+SHM
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	bool	返回结果，表示当前服务是否初始化成功状态；成功初始化后，再次初始化返回true
使用说明	应用程序调用Init接口，初始化phm的监控服务；shm通信下user参数用于设置shm文件属主。 ● 调用该接口前，需要： 1. 设置正确的json配置文件环境变量 2. 设置处理上报信息的线程数量 ● 该接口会初始化以下内容： 1. 读取配置文件，若出错则初始化失败 2. 调用EM的GetProcessStates接口，初始化当前运行进程列表 3. 初始化线程池、定时器 4. 初始化服务端，提供通信功能，若出错则初始化失败 ● shm通信下，同topic的客户端和服务端，shm文件只会创建一次，文件所属用户为首个进程设置的用户。客户端和服务端可通信的前提是都有权限访问shm文件，如果首个进程创建的shm文件属主为root，对于root进程与非root进程通信的场景，需要调用此接口设置shm文件为对应的非root用户。 ● communicateMode只可以配置为UDP+SHM或者UDP+SHM+ICC，否则报错，初始化失败。	
头文件	#include <ara/phm/phm_state_manager.h>	
注意事项	● 不可重入 ● 该接口为同步接口	

6.14 DeInit

函数原型	void PhmStateManager::DeInit();	
函数功能	取消初始化phm的监控服务	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	void	-

使用说明	调用Init接口完成监控后，应用程序需要调用DeInit接口，取消初始化phm的监控服务。 该接口会执行以下操作： 1. 释放线程池、定时器 2. 停止服务端，取消通信功能 3. 清理数据
头文件	#include <ara/phm/phm_state_manager.h>
注意事项	• 不可重入 • 该接口为同步接口

6.15 QueryRuleResults

函数原型	std::map<std::string, bool> PhmStateManager::QueryRuleResults (const std::string &ruleName);	
函数功能	查询故障发生的rule相关的status信息	
参数（IN）	const std::string &ruleName	PhmRule的名称
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	std::map<std::string, bool>	<PhmRule对应的LogicalExpression关联的status, status的状态值>
使用说明	调用此接口传入当前出现故障的rule，查询得到该rule相关的状态信息std::map<std::string, bool>；其中string为状态名称，bool为状态值。	
头文件	#include <ara/phm/phm_state_manager.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.16 ProcessNotifier

函数原型	ProcessNotifier::ProcessNotifier();	
函数功能	实例化ProcessNotifier对象，供EM使用	
参数（INOUT）	None	-

参数（OUT）	None	-
返回值	None	-
使用说明	MDC：仅**限**Foundation内部使用，不开放给客户使用，产品接口文档中需要删除， EM调用此接口实例化ProcessNotifier对象；仅限EM使用。	
头文件	#include <ara/phm/process_notifier.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.17 ProcessChanged

函数原型	void ProcessNotifier::ProcessChanged(std::string const &processName, std::string const &processState);	
函数功能	通知PHM进程状态发生了变化，供EM使用	
参数（IN）	const std::string &processName	进程名称
	const std::string &processState	进程状态
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	仅限Foundation内部使用，不开放给客户使用，产品接口文档中需要删除， EM调用此接口通知进程状态变化，传入进程标识和相应的状态； 进程状态有以下几种字符串表示：RUNNING、TERMINATED、ABORTED 其中，RUNNING或ABORTED表示：进程运行或者启动失败，需要监控。 TERMINATED表示：进程正常退出，不再需要监控。 仅EM组件使用	
头文件	#include <ara/phm/process_notifier.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.18 RegisterProcessStatesCallback

函数原型	void PhmStateManager::RegisterProcessStatesCallback(const std::function<std::vector<std::pair<std::string, std::string>>(void)> &cb)	
函数功能	注册获取进程状态回调函数，在没有EM的情况下，用于PHM服务端获取进程状态集合	
参数（IN）	const std::function<std::vector<std::pair<std::string, std::string>>(void)> &cb	获取进程状态集合的回调函数
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	调用此接口注册返回进程状态的回调函数，PHM服务端调用此回调函数，得到返回的当前运行进程集合；进程状态有以下几种字符串表示： IDLE; STARTING; RUNNING; TERMINATING; TERMINATED; ABORTED 仅限没有EM时使用	
头文件	#include <ara/phm/phm_state_manager.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.19 SetProcessState

函数原型	void PhmStateManager:: SetProcessState (const std::string &processName, const std::string &processState);	
函数功能	PHM服务端进程设置进程状态	
参数（IN）	const std::string &processName	进程名称
	const std::string &processState	进程状态
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-

使用说明	PHM服务端进程调用此接口设置进程状态 进程状态有以下几种字符串表示：RUNNING、ABORTED、TERMINATED RUNNING或ABORTED表示：进程运行或者启动失败，需要监控。 TERMINATED表示：进程正常退出，不再需要监控。 仅限没有EM时使用
头文件	#include <ara/phm/phm_state_manager.h>
注意事项	• 不可重入 • 该接口为同步接口

6.20 PhmClientCfg::SetUser

函数原型	void PhmClientCfg::SetUser(std::string const &user);	
函数功能	设置shm文件所属用户	
参数（IN）	const std::string &user	用户名称
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	• 同topic的客户端和服务端，shm文件只会创建一次，文件所属用户为首个进程设置的用户。客户端和服务端可通信的前提是都有权限访问shm文件，如果首个进程创建的shm文件属主为root，对于root进程与非root进程通信的场景，需要调用此接口设置shm文件为对应的非root用户。 • EM拉起进程的场景下，EM会使用此接口创建与uid参数相同用户的shm文件，同topic的其他EM拉起的进程无需再调用此接口。	
头文件	#include <ara/phm/phm_client_cfg.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.21 PhmClientCfg::GetUser

函数原型	std::string PhmClientCfg::GetUser();
函数功能	返回设置的shm文件所属用户

参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	std::string	用户名称
使用说明	调用此接口返回shm文件所属用户	
头文件	#include <ara/phm/phm_client_cfg.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.22 PhmErrorDomain::Name

函数原型	char const *Name() const noexcept override;	
函数功能	返回PhmErrorDomain的名称常量	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	char const *	error domain的名称
使用说明	调用此接口error domain的名称：Phm	
头文件	#include <ara/phm/phm_error_domain.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.23 PhmErrorDomain::Message

函数原型	char const *Message(ara::core::ErrorDomain::CodeType errorCode) const noexcept override;	
函数功能	返回错误码相关的信息	
参数（IN）	errorCode	错误码
参数（INOUT）	None	-
参数（OUT）	None	-

返回值	char const *	错误码相关信息
使用说明	调用此接口得到相关错误码的信息	
头文件	#include <ara/phm/phm_error_domain.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.24 PhmErrorDomain::ThrowAsException

函数原型	void ThrowAsException(ara::core::ErrorCode const &errorCode) const noexcept(false) override	
函数功能	根据错误码创建PhmException实例，并作为c++异常抛出	
参数（IN）	errorCode	要被throw的错误
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	传入错误码创建PhmException实例，并作为c++异常抛出	
头文件	#include <ara/phm/phm_error_domain.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.25 PhmException::PhmException

函数原型	explicit PhmException(ara::core::ErrorCode && err) noexcept;	
函数功能	构造一个包含错误码的PhmException实例	
参数（IN）	err	错误码
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	None	-
使用说明	传入错误码，构造PhmException实例	
头文件	#include <ara/phm/phm_error_domain.h>	

注意事项	• 不可重入 • 该接口为同步接口

6.26 GetPhmErrorDomain

函数原型	constexpr ara::core::ErrorDomain const &GetPhmErrorDomain() noexcept;	
函数功能	返回全局PhmErrorDomain对象的引用	
参数（IN）	None	-
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::ErrorDomain const &	全局PhmErrorDomain对象的引用
使用说明	返回全局PhmErrorDomain对象的引用	
头文件	#include <ara/phm/phm_error_domain.h>	
注意事项	• 不可重入 • 该接口为同步接口	

6.27 MakeErrorCode

函数原型	constexpr ara::core::ErrorCode MakeErrorCode (ara::phm::PhmErrc code, ara::core::ErrorDomain::SupportDataType data = 0) noexcept;	
函数功能	创建ErrorCode实例	
参数（IN）	code	错误码
参数（IN）	data	厂商定义的与错误相关的数据
参数（INOUT）	None	-
参数（OUT）	None	-
返回值	ara::core::ErrorCode	错误码对象

使用说明	传入phm错误码和厂商定义的与错误相关的数据，创建ErrorCode实例
	enum class PhmErrc : ara::core::ErrorDomain::CodeType {
	kGeneralError = 1U,
	kInvalidArguments = 2U,
	kCommunicationError = 3U,
	};

头文件	#include <ara/phm/phm_error_domain.h>
注意事项	• 不可重入 • 该接口为同步接口

7 PER

调用PER接口之前需要调用ara::core::Initialize接口完成PER初始化；去初始化PER需要调用ara::core::Deinitialize接口。具体接口描述见**3.15 Initialize and Shutdown**。

KVS数据库相关操作接口位于头文件："ara/per/key_value_storage.h"。

7.1 OpenKeyValueStorage

函数原型	ara::per::OpenKeyValueStorage(ara::core::InstanceSpecifier const &kvs)	
函数功能	打开KVS数据库	
参数（IN）	kvs	kvs数据库索引名称
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result< SharedHandle< KeyValueStorage >>	SharedHandle，当出现错误时返回PerErrc中定义的错误。
使用说明	返回的智能指针在无引用自动析构时关闭数据库。当运行数据库不存在时，将拷贝初始数据库作为运行数据库。如果初始数据库不存在则打开数据库操作失败。	
注意事项	• 可重入 • 该接口为同步接口	

7.2 ResetKeyValueStorage

函数原型	ara::per::ResetKeyValueStorage(ara::core::InstanceSpecifier const &kvs)	
函数功能	将KeyValueStorage实例重置为初始状态。	
参数（IN）	kvs	kvs数据库索引名称
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	为空，当出现错误时返回PerErrc中定义的错误。
使用说明	此接口允许将该数据库存储的数据重置为初始数据库的状态。当该数据库已经打开时，接口调用失败返回kResourceBusyError。调用此接口时需要保证对应的初始数据库存在，如果初始数据库不存在将返回错误。	
注意事项	• 可重入 • 该接口为同步接口	

7.3 GetCurrentKeyValueStorageSize

函数原型	ara::per::GetCurrentKeyValueStorageSize(ara::core::InstanceSpecifier const &kvs)	
函数功能	返回KVS数据库当前占用的空间（以字节为单位）。	
参数（IN）	kvs	kvs数据库索引名称
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<uint64_t>	数据库占用的字节数，当出现错误时返回PerErrc中定义的错误
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

7.4 KeyValueStorage class

KVS数据库支持存储的数据类型为：

基本数据类型有bool、double、float、int8_t、int16_t、int32_t、int64_t、uint8_t、uint16_t、uint32_t、uint64_t；

复杂数据类型有：String、Vector、Array（C++中定义方式，不支持C语言风格数组）、Map、Struct（需使用代码生成工具生成序列化头文件）；

7.4.1 KeyValueStorage::GetAllKeys

函数原型	ara::per::KeyValueStorage::GetAllKeys()	
函数功能	返回KeyValueStorage所有当前可用键的列表。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<ara::core::Vector<ara::core::String>>	KeyValueStorage所有当前可用键的列表，当出现错误时返回PerErrc中定义的错误
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

7.4.2 KeyValueStorage::HasKey

函数原型	ara::per::KeyValueStorage::HasKey(ara::core::StringView key)	
函数功能	检查KVS数据库中是否存在该key	
参数（IN）	key	需要检查的key
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<bool>	存在或者不存在，当出现错误时返回PerErrc中定义的错误
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

7.4.3 KeyValueStorage::GetValue

函数原型	ara::per::KeyValueStorage::GetValue(ara::core::StringView key)	
函数功能	返回数据库中该key对应的数据	
参数（IN）	key	需要获取数据的key
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<T>	索引到的数据或者当出现错误时返回PerErrc中定义的错误
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

7.4.4 KeyValueStorage::SetValue

函数原型	ara::per::KeyValueStorage::SetValue(ara::core::StringView key, const T &value)	
函数功能	将key对应的数据存储在KVS数据库缓存中。如果值已经存在，则将覆盖它，与存储的数据类型无关。	
参数（IN）	key	需要存储数据的key
参数（IN）	value	需要存储的数据
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	返回空，当出现错误时返回PerErrc中定义的错误
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

7.4.5 KeyValueStorage::RemoveKey

函数原型	ara::per::KeyValueStorage::RemoveKey(ara::core::StringView key)	
函数功能	从KVS数据库缓存中移除该key及对应的数据	
参数（IN）	key	需要移除数据的key
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	返回空，当出现错误时返回PerErrc中定义的错误
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

7.4.6 KeyValueStorage::RemoveAllKeys

函数原型	ara::per::KeyValueStorage::RemoveAllKeys()	
函数功能	移除KVS数据库缓存中所有的key及其对应的数据	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	返回空，当出现错误时返回PerErrc中定义的错误
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

7.4.7 KeyValueStorage::SyncToStorage

函数原型	ara::per::KeyValueStorage::SyncToStorage()	
函数功能	将数据库缓存的数据写入磁盘文件中	
参数（IN）	None	NA

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	返回空，当出现错误时返回PerErrc中定义的错误
使用说明	当数据库未被修改过，此接口将直接返回成功。	
注意事项	• 可重入 • 该接口为同步接口	

7.4.8 KeyValueStorage::DiscardPendingChanges

函数原型	ara::per::KeyValueStorage::DiscardPendingChanges()	
函数功能	删除自上次调用SyncToStorage()以来或自使用OpenKeyValueStorage()打开KeyValueStorage以来对KeyValueStorage的所有挂起更改。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	返回空，当出现错误时返回PerErrc中定义的错误
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8 DM

8.1 注意事项

1. 以下所有API接口调用前应调用ara::core::Initialize接口，初始化诊断资源，且main函数结束前应调用ara::core::Deinitialize接口回收诊断资源。
2. 事件在系统内是唯一的，DM应仅支持从一个单一来源。这意味着只有一个应用程序可以报告特定事件并且事件上报接口不能重入。

8.2 共用数据类型

8.2.1 MetaInfo

接口类定义

类名	ara::diag::MetaInfo
命名空间	namespace ara::diag
语法	class MetaInfo final {...};
头文件	#include "ara/diag/meta_info.h"
说明	ara::diag::MetaInfo用于获取传输层信息。

8.2.1.1 MetaInfo 默认构造函数

函数原型	MetaInfo（ ）= delete	
函数功能	禁止使用MetaInfo默认构造函数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.1.2 MetaInfo 移动构造函数

函数原型	MetaInfo(MetaInfo && other) noexcept = default	
函数功能	移动构造函数	
参数（IN）	other	MetaInfo实例
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.1.3 MetaInfo 拷贝构造函数

函数原型	MetaInfo(const MetaInfo & other) = delete	
函数功能	禁用拷贝构造函数	
参数（IN）	other	MetaInfo实例
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA

使用说明	无
注意事项	• 可重入 • 该接口为同步接口

8.2.1.4 MetaInfo 移动赋值函数

函数原型	MetaInfo &operator = (MetaInfo && other) = default	
函数功能	移动赋值函数	
参数（IN）	other	MetaInfo实例
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.1.5 MetaInfo 拷贝赋值函数

函数原型	MetaInfo &operator = (const MetaInfo & other) = delete	
函数功能	禁用拷贝赋值函数	
参数（IN）	other	MetaInfo实例
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.1.6 MetaInfo::GetValue

函数原型	ara::core::Optional<ara::core::StringView> GetValue(ara::core::StringView key) const

函数功能	获取指定key的value内容	
参数（IN）	key	需要获取内容的key值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Optional<ara::core::StringView>	若MetaInfo中存在该指定key值，则返回值中可获取对应value值。 若MetaInfo中不存在该指定key值，则返回值中不可获取对应value值。
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.1.7 MetaInfo::GetContext

函数原型	Context GetContext() const	
函数功能	获取MetaInfo的内容类型	
参数（IN）	key	需要获取内容的key值
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Optional<ara::core::StringView>	若MetaInfo中存在该指定key值，则返回值中可获取对应value值。 若MetaInfo中不存在该指定key值，则返回值中不可获取对应value值。
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.1.8 MetaInfo::Context

类型名	Context
命名空间	ara::diag::MetaInfo

语法	enum class Context : std::uint32_t {
	kDiagnosticCommunication, //DCM服务请求
	kFaultMemory, //读取冻结帧请求
	kDoIP//读取VIN请求
	};

头文件	#include "ara/diag/meta_info.h"
说明	MetaInfo内容类型

8.2.1.9 MetaInfo::~MetaInfo

函数原型	~MetaInfo()	
函数功能	MetaInfo析构函数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.2 ReentrancyType

类型名	ReentrancyType
命名空间	ara::diag::ReentrancyType

语法	enum class ReentrancyType: std::uint8_t {
	kFully = 0x00, //回调函数内代码可重入
	kNot = 0x01 //回调函数内代码不可重入
	};

头文件	#include "ara/diag/reentrancy.h"
说明	可重入类型

8.2.3 CancellationHandler

接口类定义

类名	ara::diag::CancellationHandler
命名空间	namespace ara::diag
语法	class CancellationHandler final {...};
头文件	#include "ara/diag/cancellation_handfler.h"
说明	CancellationHandler包含表示对应服务处理是否已被取消的状态变量，用户可使用该接口类查询服务处理状态或注册服务取消通知。

8.2.3.1 CancellationHandler 默认构造函数

函数原型	CancellationHandler（ ）= delete	
函数功能	禁止使用CancellationHandler默认构造函数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.3.2 CancellationHandler 移动构造函数

函数原型	CancellationHandler（CancellationHandler && other）	
函数功能	移动构造函数	
参数（IN）	other	CancellationHandler实例
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA

使用说明	无
注意事项	• 可重入 • 该接口为同步接口

8.2.3.3 CancellationHandler 拷贝构造函数

函数原型	CancellationHandler（CancellationHandler & other）= delete	
函数功能	禁用拷贝构造函数	
参数（IN）	other	CancellationHandler实例
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.3.4 CancellationHandler 移动赋值函数

函数原型	CancellationHandler &operator = (CancellationHandler && other)	
函数功能	移动赋值函数	
参数（IN）	other	CancellationHandler实例
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.3.5 CancellationHandler 拷贝赋值函数

函数原型	CancellationHandler &operator = (CancellationHandler & other) = delete	
函数功能	禁用拷贝赋值函数	
参数（IN）	other	CancellationHandler实例
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.3.6 CancellationHandler::IsCanceled

函数原型	bool IsCanceled() const	
函数功能	查询服务是否被取消处理	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	bool	true：服务处理已被取消 false：服务处理未被取消
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.2.3.7 CancellationHandler::SetNotifier

函数原型	void SetNotifier(std::function<void()> notifier)	
函数功能	注册通知回调	
参数（IN）	other	CancellationHandler实例
参数（INOUT）	None	NA

参数（OUT）	None	NA
返回值	None	NA
使用说明	若服务被取消，通过该函数注册的函数，将被调用	
注意事项	• 可重入 • 该接口为同步接口	

8.3 生成诊断 API 接口

8.3.1 Typed Routine

类名	ShortnameOfRIPortInterface（ARXML定义的DiagnosticRoutineInterface的ShortName）
命名空间	ARXML中定义的命名空间 Namespace1OfPortInterface::Namespace2OfPortInterface
语法	class ShortnameOfRIPortInterface {...};
头文件	#include "ara/diag/<ShortNameOfDiagnosticRoutineInterface>_routine.h"
说明	该接口类为用户定义的DiagnosticRoutineInterface生成的RoutineControl服务接口类，继承该接口类，在子类中重写Start、Stop、RequestResults分别完成RoutineControl服务的启动、停止、请求运行结果的处理。

8.3.1.1 Routine::StartOutput type

类型名	StartOutput
命名空间	Routine类内 Namespace1OfPortInterface::Namespace2OfPortInterface::ShortnameOfRIPortInterface
语法	struct StartOutput{...};
头文件	#include "ara/diag/<ShortNameOfDiagnosticRoutineInterface>_routine.h"
说明	该结构体为Routine::Start接口返回值，StartOutput内部成员为DiagnosticRoutineInterface中定义的Start的argument的顺序排列，数据类型为argument数据类型， 成员名为argument的ShortName。

8.3.1.2 Routine::StopOutput type

类型名	StopOutput
命名空间	Routine类内 Namespace1OfPortInterface::Namespace2OfPortInterface::ShortnameOfRIPortInterface
语法	struct StopOutput{...};
头文件	#include "ara/diag/<ShortNameOfDiagnosticRoutineInterface>_routine.h"
说明	该结构体为Routine::Stop接口返回值，StopOutput内部成员为DiagnosticRoutineInterface中定义的Stop的argument的顺序排列，数据类型为argument数据类型， 成员名为argument的ShortName。

8.3.1.3 Routine::RequestResultsOutput type

类型名	RequestResultsOutput
命名空间	Routine类内 Namespace1OfPortInterface::Namespace2OfPortInterface::ShortnameOfRIPortInterface
语法	struct RequestResultsOutput{...};
头文件	#include "ara/diag/<ShortNameOfDiagnosticRoutineInterface>_routine.h"
说明	该结构体为Routine::Stop接口返回值，RequestResultsOutput内部成员为DiagnosticRoutineInterface中定义的RequestResult的argument的顺序排列，数据类型为argument数据类型， 成员名为argument的ShortName。

8.3.1.4 Routine constructor

函数原型	explicit ShortnameOfRIPortInterface(const ara::core::InstanceSpecifier &specifier, ReentrancyType reentrancyType)	
函数功能	RoutineControl服务接口构造函数	
参数（IN）	specifier	DiagnosticRoutineInterface的InstanceSpecifier
	reentrancyType	指定该DiagnosticRoutineInterface内接口的可重入性

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.3.1.5 Routine destructor

函数原型	virtual ~ShortnameOfRIPortInterface() noexcept = default	
函数功能	RoutineControl服务接口析构函数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.3.1.6 Routine::Offer

函数原型	ara::core::Result<void> Offer()	
函数功能	提供对应DID的DataIdentifier服务。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA

返回值	ara::core::Result<void>	提供服务返回值，若提供服务成功，ara::core::Result<void>含正确接口。 若提供服务失败ara::core::Result<void>含错误码，错误码如下： DiagOfferErrc::kAlreadyOfferd 服务已提供 DiagOfferErrc::kConfiguration Mismatch 服务配置错误
使用说明	该接口调用成功后，DM收到对应的RoutineControl请求后，将调用该Routine子类的Start、Stop、RequestResults接口	
注意事项	• 可重入 • 该接口为同步接口	

8.3.1.7 Routine::StopOffer

函数原型	void StopOffer()	
函数功能	停止提供RoutineControl服务。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	该接口调用后，DM收到对应的RoutineControl请求后，将不会触发Routine子类的Start、Stop、RequestResults接口。	
注意事项	• 可重入 • 该接口为同步接口	

8.3.1.8 Routine::Start

函数原型	virtual ara::core::Future<StartOutput> Start(Namespace1OfTypeOfArgumentDataPrototype::Type1OfArgumentDataPrototype Shortname1OfArgumentDataPrototype, Namespace2OfTypeOfArgumentDataPrototype::Type2OfArgumentDataPrototype Shortname2OfArgumentDataPrototype,..., MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0;	
函数功能	RoutineControl服务Start请求处理接口	
参数（IN）	cancellationHandler	coversation取消通知接口（当前不支持）
参数（INOUT）	metaInfo	诊断消息元信息，可用于获取该UDS请求对应conversation（当前不支持）
参数（OUT）	None	NA
返回值	ara::core::Future<StartOutput>	返回Routine启动结果，若启动成功，返回用户定义的StartOutput，若启动失败返回DiagUdsNrcErrorDomain内各错误码。
使用说明	生成接口类中该接口为纯虚接口，用户需要在子类中重写该Start函数，DM收到RoutineControl服务的Start请求后，将调用子类重写的Start接口。	
注意事项	可重入性：由Routine constructor第二个参数Reentrancy设置 异步	

8.3.1.9 Routine::Stop

函数原型	virtual ara::core::Future<StopOutput> Stop(Namespace1OfTypeOfArgumentDataPrototype::Type1OfArgumentDataPrototype Shortname1OfArgumentDataPrototype, Namespace2OfTypeOfArgumentDataPrototype::Type2OfArgumentDataPrototype Shortname2OfArgumentDataPrototype,..., MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0;
函数功能	RoutineControl服务Stop请求处理接口

参数（IN）	cancellationHandler	coversation取消通知接口（当前不支持）
参数（INOUT）	metaInfo	诊断消息元信息，可用于获取该UDS请求对应conversation（当前不支持）
参数（OUT）	None	NA
返回值	ara::core::Future<StopOutput>	返回Routine停止结果，若停止成功，返回用户定义的StopOutput，若停止失败返回DiagUdsNrcErrorDomain内各错误码。
使用说明	生成接口类中该接口为纯虚接口，用户需要在子类中重写该Stop函数，DM收到RoutineControl服务的Stop请求后，将调用子类重写的Stop接口。	
注意事项	可重入性：由Routine constructor第二个参数Reentrancy设置 异步	

8.3.1.10 Routine::RequestResults

函数原型	virtual ara::core::Future<RequestResultsOutput> RequestResults(Namespace1OfTypeOfArgumentDataPrototype::Type1OfArgumentDataPrototype Shortname1OfArgumentDataPrototype, Namespace2OfTypeOfArgumentDataPrototype::Type2OfArgumentDataPrototype Shortname2OfArgumentDataPrototype,..., MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0;	
函数功能	RoutineControl服务Stop请求处理接口	
参数（IN）	cancellationHandler	coversation取消通知接口（当前不支持）
参数（INOUT）	metaInfo	诊断消息元信息，可用于获取该UDS请求对应conversation（当前不支持）
参数（OUT）	None	NA
返回值	ara::core::Future<RequestResultsOutput>	返回Routine运行结果，若运行成功，返回用户定义的RequestResultsOutput，若失败返回DiagUdsNrcErrorDomain内各错误码。

使用说明	生成接口类中该接口为纯虚接口，用户需要在子类中重写该RequestResults函数，DM收到RoutineControl服务的RequestResults请求后，将调用子类重写的RequestResults接口。
注意事项	可重入性：由Routine constructor第二个参数Reentrancy设置 异步

8.3.2 Typed DataIdentifier

类名	ShortnameOfDIPortInterface（ARXML定义的DiagnosticDataIdentifierInterface的ShortName）
命名空间	ARXML中定义的命名空间Namespace1OfPortInterface::Namespace2OfPortInterface
语法	class ShortnameOfDIPortInterface {...};
头文件	#include "ara/diag/<ShortNameOfDiagnosticDataIdentifierInterface>_data_identifier.h"
说明	该接口类为用户定义的DiagnosticDataIdentifierInterface生成的DataIdentifier服务接口类，继承该接口类，在子类中重写Write、Read分别完成指定DID的WriteDataByIdentifier、ReadDataByIdentifier服务的处理。

8.3.2.1 DataIdentifier::Output type

类型名	Output
命名空间	DataIdentifier类内Namespace1OfPortInterface::Namespace2OfPortInterface::ShortnameOfDIPortInterface
语法	struct Output {...};
头文件	#include "ara/diag/<ShortNameOfDataIdentifier>_data_identifier.h"
说明	该结构体为DataIdentifier::Read接口返回值，Output内部成员为DataIdentifierInterface中定义的Read的argument的顺序排列，数据类型为argument数据类型， 成员名为argument的ShortName。

8.3.2.2 DataIdentifier constructor

函数原型	ShortnameOfDIPortInterface(const ara::core::InstanceSpecifier &specifier, DataIdentifierReentrancyType reentrancyType)	
函数功能	DataIdentifier服务接口构造函数	
参数（IN）	specifier	DiagnosticDataIdentifierInterface的InstanceSpecifier
	reentrancyType	指定该DiagnosticDataIdentifierInterface内write、read接口的可重入性
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	注意每个Diagnostic Server的每个DID仅能构造一个服务实例	
注意事项	• 可重入 • 该接口为同步接口	

8.3.2.3 DataIdentifier destructor

函数原型	virtual ~ShortnameOfDIPortInterface() noexcept = default	
函数功能	DataIdentifier服务接口析构函数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.3.2.4 DataIdentifier::Offer

函数原型	ara::core::Result<void> Offer()
函数功能	提供对应DID的DataIdentifier服务。

参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	提供服务返回值，若提供服务成功，ara::core::Result<void>含正确接口。 若提供服务失败ara::core::Result<void>含错误码，错误码如下： DiagOfferErrc::kAlreadyOfferd 服务已提供 DiagOfferErrc::kConfiguration Mismatch 服务配置错误
使用说明	该接口调用成功后，DM收到对应的WriteDataByIdentifier、ReadDataByIdnetifier请求后，将调用该DataIdentifier子类的Write、Read接口。	
注意事项	• 可重入 • 该接口为同步接口	

8.3.2.5 DataIdentifier::StopOffer

函数原型	void StopOffer()	
函数功能	停止提供对应DID的DataIdentifier服务。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	该接口调用后，DM收到对应的WriteDataByIdentifier、ReadDataByIdnetifier请求后，将不会触发DataIdentifierInterface子类的write、read接口	
注意事项	• 可重入 • 该接口为同步接口	

8.3.2.6 DataIdentifier::Read

函数原型	virtual ara::core::Future<Output> Read(MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0;	
函数功能	ReadDataByIdentifier请求处理接口	
参数（IN）	cancellationHandler	coversation取消通知接口（当前不支持）
参数（INOUT）	metaInfo	诊断消息元信息，可用于获取该UDS请求对应conversation（当前不支持）
参数（OUT）	None	NA
返回值	ara::core::Future<Output>	返回Read DID结果，若读取成功，返回DID数据内容，若读取失败返回DiagUdsNrcErrorDomain内各错误码。
使用说明	生成接口类中该接口为纯虚接口，用户需要在子类中重写该Read函数，DM收到ReadDataByIdentifier请求后，将调用子类重写的Read接口。	
注意事项	可重入性：由DataIdentifier constructor第二个参数Reentrancy设置 异步	

8.3.2.7 DataIdentifier::Write

函数原型	virtual ara::core::Future<void> Write(Namespace1OfTypeOfArgumentDataPrototype::Type1OfArgumentDataPrototype Shortname1OfArgumentDataPrototype, Namespace2OfTypeOfArgumentDataPrototype::Type2OfArgumentDataPrototype Shortname2OfArgumentDataPrototype, ..., MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0;	
函数功能	WriteDataByIdentifier请求处理接口	
参数（IN）	cancellationHandler	coversation取消通知接口（当前不支持）
	Shortname1OfArgumentDataPrototype, Shortname2OfArgumentDataPrototype, ...	DataIdentifier的各Diagnostic Data Element数值

参数（INOUT）	metaInfo	诊断消息元信息，可用于获取该UDS请求对应conversation（当前不支持）
参数（OUT）	None	NA
返回值	ara::core::Future<void>	返回Read DID结果，若读取成功，ara::core::Future<void>返回空内容Result，若读取失败返回ara::core::Future<void>返回DiagUdsNrcErrorDomain内各错误码。
使用说明	生成接口类中该接口为纯虚接口，用户需要在子类中重写该Write函数，DM收到WriteDataByIdentifier请求后，将调用子类重写的Write接口，另入参Diagnostic Data Element数据类型为DataIdentifierInterface中定义的Write的argument的顺序排列，参数名为argument ShortName。	
注意事项	可重入性：由DataIdentifier constructor第二个参数Reentrancy设置 异步	

8.3.3 Typed DataElement

类名	ShortnameOfDEPortInterface（ARXML定义的DiagnosticDataElementInterface的ShortName）
命名空间	ARXML中定义的命名空间Namespace1OfPortInterface::Namespace2OfPortInterface
语法	class ShortnameOfDEPortInterface {...};
头文件	#include "ara/diag/<ShortNameOfDiagnosticDataElementInterface>_data_element.h"
说明	该接口类为用户定义的DiagnosticDataElementInterface生成的DataElement服务接口类，继承该接口类，在子类中重写Read分别完成指定Diagnostic data element读取请求处理。

8.3.3.1 DataElement::OperationOutput type

类型名	OperationOutput
命名空间	DataElement类内Namespace1OfPortInterface::Namespace2OfPortInterface::ShortnameOfDEPortInterface
语法	struct OperationOutput {...};

头文件	#include "ara/diag/<ShortNameOfDiagnosticDataElementInterface>_data_element.h"
说明	该结构体为DataElement::Read接口返回值，OperationOutput 内部成员为DataElementInterface中定义的Read的argument，数据类型为argument数据类型， 成员名为argument的ShortName。

8.3.3.2 DataElement constructor

函数原型	ShortnameOfDEPortInterface(const ara::core::InstanceSpecifier &specifier, ReentrancyType reentrancyType)	
函数功能	构造函数	
参数（IN）	specifier	DiagnosticDataElementInterface的InstanceSpecifier
	reentrancyType	指定该DiagnosticDataElementInterface内read接口的可重入性
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	注意每个Diagnostic Server的每个DiagnosticElement仅能构造一个服务实例。	
注意事项	• 可重入 • 该接口为同步接口	

8.3.3.3 DataElement destructor

函数原型	virtual ~ShortnameOfDEPortInterface() noexcept = default	
函数功能	DataElement服务析构函数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	

注意事项	• 可重入 • 该接口为同步接口

8.3.3.4 DataElement::Offer

函数原型	ara::core::Result<void> Offer()	
函数功能	提供对应DataElement服务。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	提供服务返回值，若提供服务成功，ara::core::Result<void>含正确接口。 若提供服务失败ara::core::Result<void>含错误码，错误码如下： DiagOfferErrc::kAlreadyOfferd 服务已提供 DiagOfferErrc::kConfiguration Mismatch 服务配置错误
使用说明	该接口调用成功后，DM需要读取DiagnosticDataElement数据进行Condition校验时，将调用该DataElement子类的Read接口。	
注意事项	• 可重入 • 该接口为同步接口	

8.3.3.5 DataElement::StopOffer

函数原型	void StopOffer()	
函数功能	停止提供对应DataElement服务。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	该接口调用后，DM需要读取DiagnosticDataElement数据进行Condition校验时，将不会调用该DataElement子类的Read接口。	

注意事项	• 可重入 • 该接口为同步接口

8.3.3.6 DataElement::Read

函数原型	virtual ara::core::Future<OperationOutput> Read(MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0;	
函数功能	DiagnosticDataElement读取请求处理接口	
参数（IN）	cancellationHandler	coversation取消通知接口（ReadElement功能中，该接口无效）
参数（INOUT）	metaInfo	诊断消息元信息（ReadElement功能中，该信息无效）
参数（OUT）	None	NA
返回值	ara::core::Future<OperationOutput>	返回Read Element结果，若读取成功，返回Diagnostic Data Element数据内容，若读取失败返回DiagUdsNrcErrorDomain内各错误码。
使用说明	生成接口类中该接口为纯虚接口，用户需要在子类中重写该Read函数，DM收到DiagnosticDataElement读取请求后，将调用子类重写的Read接口。	
注意事项	可重入性：由DataElement constructor第二个参数Reentrancy设置 异步	

8.4 诊断 API 接口

8.4.1 Monitor

接口类定义

类名	ara::diag::Monitor
命名空间	namespace ara::diag
语法	class Monitor final {...};
头文件	#include "ara/diag/monitor.h"
说明	类实现对诊断Monitor接口的操作。

8.4.1.1 Monitor::CounterBased

类名	ara::diag::Monitor::CounterBased
命名空间	namespace ara::diag::Monitor

语法	struct CounterBased {
	std::int16_t failedThreshold; //符合失败的阈值
	std::int16_t passedThreshold; //符合成功的阈值
	std::uint16_t failedStepSize; //每个预失败报告的步长
	std::uint16_t passedStepSize; //每个预成功报告的步长
	std::int16_t failedJumpValue; //跳转值失败
	std::int16_t passedJumpValue; //跳转值成功
	bool useJumpToFailed; //是否支持跳转到失败
	bool useJumpToPassed; //是否支持跳转到成功
	};

头文件	#include "ara/diag/monitor.h"
说明	表示基于计数器的去抖的参数。

8.4.1.2 Monitor::TimeBased

类名	ara::diag::Monitor::TimeBased
命名空间	namespace ara::diag::Monitor

语法	struct TimeBased {
	std::uint32_t passedMs; //直到成功的时间（毫秒）
	std::uint32_t failedMs; //直到失败的时间（毫秒）
	};

头文件	#include "ara/diag/monitor.h"
说明	表示基于定时器的去抖的参数。

8.4.1.3 InitMonitorReason

类名	ara::diag::InitMonitorReason
命名空间	namespace ara::diag

语法	enum class InitMonitorReason {
	kClear = 0, /* 事件已清除，所有内部值和状态都将重置 */
	kRestart = 1, /* 事件的操作周期已（重新）启动 */
	kReenabled = 2, /* 启用条件或重新启用DTC设置 */
	};

头文件	#include "ara/diag/monitor.h"
说明	表示报告给应用程序的状态信息可以重新初始化监视器的原因。

8.4.1.4 MonitorAction

类名	ara::diag::MonitorAction
命名空间	namespace ara::diag

语法	enum class MonitorAction : uint8_t {
	kPassed = 0x00, // 监视器报告合格的测试结果通过。
	kFailed = 0x01, // 监视器报告合格的测试结果失败。
	kPrepassed = 0x02, // 监视器报告不合格的测试结果预通过。
	kPrefailed = 0x03, // 监视器报告不合格的测试结果预失败。
	kFdcThresholdReached = 0x04, // 监视器触发扩展数据记录和冻结帧的存储（如果触发条件连接到此阈值）。
	kResetTestFailed = 0x05, // 重置测试失败位，没有任何其他副作用，如就绪。
	kFreezeDebouncing = 0x06, // 冻结内部去抖计数器/计时器
	kResetDebouncing = 0x07, // 重置内部去抖计数器/计时器。
	};

头文件	#include "ara/diag/monitor.h"
说明	表示应用程序报告的与错误监控相关的状态信息。

8.4.1.5 Monitor::Monitor

函数原型	Monitor(const ara::core::InstanceSpecifier &specifier, std::function< void(InitMonitorReason)> initMonitor, std::function< std::sint8_t()> getFaultDetectionCounter)	
函数功能	Monitor类构造函数	
参数（IN）	specifier	Monitor实例名
	initMonitor	提供给用户注册的Monitor初始化函数指针

	getFaultDetectionCounter	提供给用户注册的获取故障检测计数器的初始化函数指针
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	Monitor(const ara::core::InstanceSpecifier &specifier, std::function< void(InitMonitorReason)> initMonitor, CounterBased debouncing)	
函数功能	Monitor类具有基于计数器的去抖算法的构造函数。	
参数（IN）	specifier	Monitor实例名
	initMonitor	提供给用户注册的Monitor初始化函数指针
	debouncing	将基于计数器去抖算法设置给Monitor
返回值	None	-
使用说明	无	
注意事项	无	

函数原型	Monitor(const ara::core::InstanceSpecifier &specifier, std::function< void(InitMonitorReason)> initMonitor, TimeBased debouncing)	
函数功能	Monitor类具有基于定时器的去抖算法的构造函数。	
参数（IN）	specifier	Monitor实例名
	initMonitor	提供给用户注册的Monitor初始化函数指针
	debouncing	将基于定时器去抖算法设置给Monitor
返回值	None	-
使用说明	无	
注意事项	无	

8.4.1.6 Monitor::ReportMonitorAction

函数原型	ara::core::Result ReportMonitorAction (MonitorAction action)	
函数功能	Monitor上报相关状态信息给诊断进程的函数。	
参数（IN）	action	包含诊断监视器或命令最后一个（未）合格的测试结果，以控制去抖或强制预存储。
返回值	ara::core::Result<void>	结果为无效或错误。
使用说明	无	
注意事项	无	

8.4.2 Event

类名	ara::diag::Event
命名空间	namespace ara::diag
语法	class Event {...};
头文件	#include "ara/diag/event.h"
说明	类实现对诊断事件接口的操作。

8.4.2.1 DTCFormatType

类名	ara::diag::DTCFormatType
命名空间	namespace ara::diag

语法	enum class DTCFormatType : uint8_t {
	kDTCFormatOBD = 0, /* SAE_J2012-DA_DTCFormat_00，如ISO 15031-6规范中定义 */
	kDTCFormatUDS = 1, /* ISO_14229-1_DTCFormat， 如ISO 14229-1 规范中定义 */
	kDTCFormatJ1939 = 2/* SAE_J1939-73_DTCFormat ，如SAE J1939-73规范中定义 */
	};

头文件	#include "ara/diag/event.h"
说明	表示根据ISO 14229-1的DTC格式的类型。

8.4.2.2 EventStatusBit

类名	ara::diag::EventStatusBit
命名空间	namespace ara::diag

语法	enum class EventStatusBit : uint8_t {
	kTestFailed = 0, // /< bit 0: 测试失败
	kTestFailedThisOperationCycle = 1, // /< bit 1: 本操作周期测试失败
	kTestNotCompletedThisOperationCycle = 6, // /< bit 6: 本操作周期测试未完成
	};

头文件	#include "ara/diag/event.h"
说明	表示单个事件状态位。

8.4.2.3 Event::EventStatusByte

类名	ara::diag::Event::EventStatusByte
命名空间	namespace ara::diag::Event
语法	using EventStatusByte = std::uint8_t;
头文件	#include "ara/diag/event.h"
说明	当前事件状态字节，位编码。

8.4.2.4 Event::DebouncingState

类名	ara::diag::Event::DebouncingState
命名空间	namespace ara::diag::Event

语法	enum class DebouncingState : uint8_t {
	kNeutral = 0x00, // （对应于FDC = 0）
	kTemporarilyDefective = 0x01, // / 暂时缺陷（对应于0 < FDC < 127）
	kFinallyDefective = 0x02, // /< 最终缺陷（对应于FDC = 127）
	kTemporarilyHealed = 0x04, // /< 暂时愈合（对应于FDC = -128 < FDC < 0）
	kFinallyHealed = 0x08 // 最终愈合（对应于FDC = -128）
	};

头文件	#include "ara/diag/event.h"
说明	事件的去抖状态。

8.4.2.5 Event::Event

函数原型	explicit Event (const ara::core::InstanceSpecifier &specifier);	
函数功能	Event类对象的构造函数。	
参数（IN）	specifier	实例表示。
返回值	无	-
使用说明	无	
注意事项	无	

8.4.2.6 Event::~Event

函数原型	~Event () noexcept=default;	
函数功能	Event类对象的析构函数。	
参数（IN）	无	-
返回值	无	-
使用说明	无	
注意事项	无	

8.4.2.7 Event::GetEventStatus

函数原型	ara::core::Result GetEventStatus ();	
函数功能	返回当前诊断事件状态。	
参数（IN）	无	-
返回值	ara::core::Result< EventStatusByte >	当前诊断事件状态
使用说明	无	
注意事项	无	

8.4.2.8 Event::SetEventStatusChangedNotifier

函数原型	ara::core::Result SetEventStatusChangedNotifier (std::function< void(EventStatusByte)> notifier);

函数功能	注册一个通知器函数，如果诊断事件发生更改，该函数将被调用。	
参数（IN）	std::function< void(EventStatusByte)> notifier)	注册诊断事件发生更改的函数钩子
返回值	ara::core::Result<void>	-
使用说明	无	
注意事项	无	

8.4.2.9 Event::GetDTCNumber

函数原型	ara::core::Result GetDTCNumber (DTCFormatType dtcFormat);	
函数功能	返回与此事件实例相关的DTC-ID。	
参数（IN）	dtcFormat	定义返回值的DTC格式。
返回值	ara::core::Result< std::uint32_t >	相应DTCFormatType中的DTC编号。
使用说明	无	
注意事项	无	

8.4.2.10 Event::GetDebouncingStatus

函数原型	ara::core::Result GetDebouncingStatus ();	
函数功能	返回当前去抖状态。	
参数（IN）	无	-
返回值	ara::core::Result< DebouncingState >	返回此事件的当前去抖状态。
使用说明	无	
注意事项	无	

8.4.2.11 Event::GetTestComplete

函数原型	ara::core::Result GetTestComplete ();	
函数功能	如果事件已成熟到测试完成，则获取状态(对应于FDC = -128或FDC =127)。	
参数（IN）	无	-

返回值	ara::core::Result< bool >	返回此的当前测试完成状态事件。“true”，如果FDC = -128或FDC = 127；“false”所有其他情况。
使用说明	无	
注意事项	无	

8.4.2.12 Event::GetFaultDetectionCounter

函数原型	ara::core::Result GetFaultDetectionCounter ();	
函数功能	返回此事件的故障检测计数器的当前值。	
参数（IN）	无	-
返回值	ara::core::Result< std::int8_t >	故障检测计数器的当前值
使用说明	无	
注意事项	无	

8.4.3 DTCInformation

8.4.3.1 ControlDtcStatusType

类型名	ControlDtcStatusType
命名空间	ara::diag

语法	enum class ControlDtcStatusType: uint8_t {
	kDTCSettingOn = 0x00, //启动DTC状态更新
	kDTCSettingOff = 0x01 //停止DTC状态更新
	};

头文件	#include "ara/diag/dtc_information.h"
说明	DTC更新状态

8.4.3.2 UdsDtcStatusBitType

类型名	UdsDtcStatusBitType
命名空间	ara::diag

语法	enum class UdsDtcStatusBitType: uint8_t {
	kTestFailed = 0x01, //bit 0: TestFailed
	kTestFailedThisOperationCycle = 0x02 //bit 1: TestFailedThisOperationCycle
	kPendingDTC = 0x04 //bit 2: PendingDTC
	kConfirmedDTC = 0x08, //bit 3: ConfirmedDTC
	kTestNotCompletedSinceLastClear = 0x10, //bit4: TestNotCompletedSinceLastClear
	kTestFailedSinceLastClear = 0x20, //bit 5: TestFailedSinceLastClear
	kTestNotCompletedThisOperationCycle = 0x40, //bit 6: TestNotCompletedThisOperationCycle
	kWarningIndicatorRequested = 0x80 //bit 7: WarningIndicatorRequested
	};

头文件	#include "ara/diag/dtc_information.h"
说明	UDS DTC状态位

8.4.3.3 DTCInformation::UdsDtcStatusByteType

类型名	UdsDtcStatusByteType
命名空间	ara::diag
语法	using UdsDtcStatusByteType = std::uint8_t
头文件	#include "ara/diag/dtc_information.h"
说明	UDS DTC状态

8.4.3.4 DTCInformation::SnapshotDataIdentifierType

类型名	SnapshotDataIdentifierType
命名空间	ara::diag

语法	struct SnapshotRecordUpdatedType {
	uint16_t dataIdentifier;
	ara::core::vector<uint8_t> data;
	}

头文件	#include "ara/diag/dtc_information.h"
说明	冻结帧对应的DID数据

8.4.3.5 DTCInformation::SnapshotDataRecordType

类型名	SnapshotDataRecordType
命名空间	ara::diag

语法	struct SnapshotRecordUpdatedType {
	uint8_t recordNumber;
	ara::core::vector<SnapshotDataIdentifierType> snapshotData;
	}

头文件	#include "ara/diag/dtc_information.h"
说明	冻结帧数据

8.4.3.6 DTCInformation::SnapshotRecordUpdatedType

类型名	SnapshotRecordUpdatedType
命名空间	ara::diag

语法	struct SnapshotRecordUpdatedType {
	uint32_t dtcUdsValue;
	SnapshotDataRecordType snapshotRecord;
	}

头文件	#include "ara/diag/dtc_information.h"
说明	冻结帧更新数据结构

8.4.3.7 DTCInformation::DTCInformation

函数原型	explicit DTCInformation(const ara::core::InstanceSpecifier& specifier)	
函数功能	DTCInformation接口构造函数	
参数（IN）	specifier	DiagnosticDTCInformationInterface的InstanceSpecifier
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.8 DTCInformation::~DTCInformation

函数原型	~DTCInformation() noexcept = default	
函数功能	DTCInformation接口析构函数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.9 DTCInformation::GetCurrentStatus

函数原型	ara::core::Result<UdsDtcStatusByteType> GetCurrentStatus(std::uint32_t dtc)	
函数功能	获取指定DTC的状态位信息	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.10 DTCInformation::SetDTCStatusChangedNotifier

函数原型	ara::core::Result<void> SetDTCStatusChangedNotifier(std::function<void(std::uint32_t dtc, UdsDtcStatusByteType udsStatusByteOld, UdsDtcStatusByteType udsStatusByteNew)> notifier)
函数功能	设置DTC状态位更新通知

参数（IN）	notifier	状态位更新通知回调
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	设置结果
使用说明	通过本函数设置DTC状态位更新通知后，DTC状态位变化，notifier函数将会被调用，参数分别为变化DTC，变化前DTC状态位，变化后DTC状态位	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.11 DTCInformation::SetSnapshotRecordUpdatedNotifier

函数原型	ara::core::Result<void> SetSnapshotRecordUpdatedNotifier(std::function<void(SnapshotRecordUpdatedType)> notifier)	
函数功能	设置冻结帧状态更新通知回调	
参数（IN）	notifier	冻结帧更新通知回调
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	设置结果
使用说明	通过本函数设置冻结帧更新通知回调后，冻结帧更新，notifier函数将会被调用	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.12 DTCInformation::GetNumberOfStoredEntries

函数原型	ara::core::Result<std::uint32_t> GetNumberOfStoredEntries()	
函数功能	获取event memory中当前存储的event entry条数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA

返回值	ara::core::Result<std::uint32_t>	当前存储的event entry条数
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.13 DTCInformation::SetNumberOfStoredEntriesNotifier

函数原型	ara::core::Result<void> SetNumberOfStoredEntriesNotifier(std::function<void(std::uint32_t)> notifier)	
函数功能	设置event memory中当前存储的event entry条数更新通知	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	设置结果
使用说明	通过本函数设置更新通知回调后，event memory中当前存储的event entry条数更新，notifier函数将会被调用	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.14 DTCInformation::Clear

函数原型	ara::core::Result<void> Clear(std::uint32_t DTCGroup)	
函数功能	清空DTC/DTCGroup相关存储内容	
参数（IN）	DTCGroup	带清空DTC/DTCGroup
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	清空结果
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.15 DTCInformation::GetControlDTCStatus

函数原型	ara::core::Result<ControlDtcStatusType> GetControlDTCStatus()	
函数功能	获取DTC状态更新控制信息	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	包含DTC状态控制信息的结果	
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.16 DTCInformation::SetControlDtcStatusNotifier

函数原型	ara::core::Result<void> SetControlDtcStatusNotifier(std::function<void(ControlDtcStatusType)> notifier)	
函数功能	设置DTC状态更新控制通知回调	
参数（IN）	notifier	状态更新回调函数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	设置结果
使用说明	通过本接口设置通知函数回调，DTC更新状态控制位更新将会触发该回调，入参为当前DTC更新控制状态	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.17 DTCInformation::EnableControlDtc

函数原型	ara::core::Result<void> EnableControlDtc()	
函数功能	使能DTC状态更新	
参数（IN）	None	NA
参数（INOUT）	None	NA

参数（OUT）	None	NA
返回值	ara::core::Result<void>	使能DTC状态更新结果
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.18 DTCInformation::GetEventMemoryOverflow

函数原型	ara::core::Result<bool> GetEventMemoryOverflow()	
函数功能	查询EventMemory是否溢出	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<bool>	查询结果：返回error表明查询失败； 返回true，Event memory溢出， 返回false，Event memory未溢出
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.3.19 DTCInformation::SetEventMemoryOverflowNotifier

函数原型	ara::core::Result<void> SetEventMemoryOverflowNotifier(std::function<void(bool)> notifier)	
函数功能	设置EventMemory溢出通知回调	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	使能DTC状态更新结果
使用说明	通过本接口设置通知函数回调，EventMemory溢出，该回调将会被调用	

注意事项	• 可重入 • 该接口为同步接口

8.4.4 Conversation

类名	ara::diag::Conversation
命名空间	namespace ara::diag
语法	class Conversation {...};
头文件	#include "ara/diag/conversation.h"
说明	Conversation信息获取接口。

8.4.4.1 uds_transport::UdsTransportProtocolMgr::GlobalChannelIdentifier

类型名	GlobalChannelIdentifier
命名空间	ara::diag::uds_transport::UdsTransportProtocolMgr
语法	using GlobalChannelIdentifier = std::tuple<UdsTransportProtocolHandler ID, ChannelID>;
头文件	#include "ara/diag/uds_transport/protocol_mgr.h"
说明	UdsTransport通道索引

8.4.4.2 uds_transport::UdsTransportProtocolHandlerID

类型名	UdsTransportProtocolHandlerID
命名空间	ara::diag::uds_transport
语法	using UdsTransportProtocolHandlerID = uint8_t;
头文件	#include "ara/diag/uds_transport/protocol_types.h"
说明	UdsTransportProtocolHandler索引

8.4.4.3 uds_transport::ChannelID

类型名	ChannelID
命名空间	ara::diag::uds_transport

语法	using ChannelID = uint32_t
头文件	#include "ara/diag/uds_transport/protocol_types.h"
说明	通道ID

8.4.4.4 ara::diag::uds_transport::UdsMessage::Address

类型名	Address
命名空间	ara::diag::uds_transport::UdsMessage
语法	using Address = uint16_t;
头文件	#include "ara/diag/uds_transport/uds_message.h"
说明	UDS消息源地址、目标地址数据类型定义

8.4.4.5 diag::ActivityStatusType

类型名	ActivityStatusType
命名空间	ara::diag

语法	enum class ActivityStatusType {
	kActive = 0x00, //活跃状态，即正在处理诊断请求或正处于non-default session状态
	kInactive = 0x01 //非活跃状态
	};

头文件	#include "ara/diag/conversation.h"
说明	conversation活动状态类型定义

8.4.4.6 diag::SessionControlType

类型名	SessionControlType
命名空间	ara::diag

语法	enum class SessionControlType: std::uint8_t {
	kDefaultSession = 0x01,
	kProgrammingSession = 0x02,
	kExtendedDiagnosticSession = 0x03,
	kSafetySystemDiagnosticSession = 0x04
	};

头文件	#include "ara/diag/conversation.h"
说明	diagnostic session类型定义

8.4.4.7 diag::SecurityLevelType

类型名	SecurityLevelType
命名空间	ara::diag

语法	enum class SecurityLevelType: std::uint8_t {
	kLocked = 0x00,
	configuration_dependent
	};

头文件	#include "ara/diag/conversation.h"
说明	security level类型定义

8.4.4.8 Conversation::ConversationIdentifierType

类型名	ConversationIdentifierType
命名空间	ara::diag::Conversation

语法	struct ConversationIdentifierType {
	ara::diag::uds_transport::UdsMessage::Address saAddress;
	ara::core::String diagServerName;
	ara::diag::uds_transport::UdsTransportProtocolMgr::GlobalChannelIdentifier globalIdentifier;
	};

头文件	#include "ara/diag/conversation.h"
说明	Conversation索引类型定义

8.4.4.9 Conversation::GetConversation

函数原型	static ara::core::Result<std::reference_wrapper<ara::diag::Conversation >> GetConversation(MetaInfo &metaInfo);	
函数功能	基于MetaInfo获取Conversation	
参数（IN）	metaInfo	含传输层信息的MetaInfo

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<std::reference_wrapper<ara::diag::Conversation>>	返回获取到的Conversation或错误码
使用说明	仅Context为kDiagnosticCommunication的MetaInfo可获取Conversation	
注意事项	• 可重入 • 该接口为同步接口 • 由于R1911/R2011规范该接口返回值类型为ara::core::Result<ara::diag::Conversation&>，存在语法问题（ara::core::Result不支持引用类型）。修改该接口返回值类型为ara::core::Result<std::reference_wrapper<ara::diag::Conversation>>	

8.4.4.10 Conversation::GetCurrentActiveConversations

函数原型	static ara::core::Vector<std::reference_wrapper<ara::diag::Conversation>> GetCurrentActiveConversations()	
函数功能	获取所有处于活跃状态的conversation	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Vector<std::reference_wrapper<ara::diag::Conversation>>	所有处于活跃状态的Conversation
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口 • 由于R1911/R2011规范该接口返回值类型为ara::core::Vector<ara::diag::Conversation&>，存在语法问题（ara::core::Vector不支持引用类型）。修改该接口返回值类型为ara::core::Vector<std::reference_wrapper<ara::diag::Conversation>>	

8.4.4.11 Conversation::GetConversationIdentifier

函数原型	ara::core::Result<ConversationIdentifierType> GetConversationIdentifier()	
函数功能	获取ConversationIdentifier信息	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<ConversationIdentifierType>	返回ConversationIdentifier信息
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.4.12 Conversation::GetActivityStatus

函数原型	ara::core::Result<ActivityStatusType> GetActivityStatus()	
函数功能	获取活跃状态信息	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<ActivityStatusType>	活跃状态信息或错误码
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.4.13 Conversation::SetActivityNotifier

函数原型	ara::core::Result<void> SetActivityNotifier(std::function<void(ActivityStatusType)> notifier)	
函数功能	设置活跃状态变化通知函数	
参数（IN）	notifier	活跃状态变化通知函数

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	设置成功返回void，设置失败返回错误码
使用说明	通过本接口设置通知函数，对应Conversation活跃状态变化，该通知函数将被调用	
注意事项	• 可重入 • 该接口为同步接口	

8.4.4.14 Conversation::GetDiagnosticSessionShortName

函数原型	ara::core::Result<ara::core::StringView> GetDiagnosticSessionShortName(SessionControlType session)	
函数功能	获取指定DiagnosticSession的ARXML中的ShortName	
参数（IN）	session	Diagnostic Session
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<ara::core::StringView>	成功返回session对应的DiagnosticSession的ShortName，失败返回错误码
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.4.15 Conversation::GetDiagnosticSession

函数原型	ara::core::Result<SessionControlType> GetDiagnosticSession()	
函数功能	获取当前diagnostic session	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<SessionControlType>	成功返回当前diagnostic session，失败返回错误码

使用说明	无
注意事项	• 可重入 • 该接口为同步接口

8.4.4.16 Conversation::SetDiagnosticSessionNotifier

函数原型	ara::core::Result<void> SetDiagnosticSessionNotifier(std::function<void(SessionControlType)> notifier)	
函数功能	设置diagnostic session变化通知函数	
参数（IN）	notifier	diagnostic session变化通知函数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	设置成功返回void，设置失败返回错误码
使用说明	通过本接口设置通知函数，对应Conversation diagnostic session变化，该通知函数将被调用	
注意事项	• 可重入 • 该接口为同步接口	

8.4.4.17 Conversation::GetDiagnosticSecurityLevelShortName

函数原型	ara::core::Result<ara::core::StringView> GetDiagnosticSecurityLevelShortName(SecurityLevelType securityLevel)	
函数功能	获取当前Diagnostic security level的ShortName	
参数（IN）	securityLevel	该入参暂无使用用途
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<ara::core::StringView>	成功返回当前Diagnostic security level对应的ShortName，失败返回错误码

使用说明	由于SecurityLevelType为枚举类型，不同securitylevel均为configuration_dependent，无法具体确定SecurityLevelName，因此本接口返回为该conversation当前securityLevel的ShortName，忽略入参securityLevel。若当前securityLevel为kLocked,返回错误
注意事项	• 可重入 • 该接口为同步接口

8.4.4.18 Conversation::GetDiagnosticSecurityLevel

函数原型	ara::core::Result<SecurityLevelType> GetDiagnosticSecurityLevel()	
函数功能	获取当前diagnostic security level	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<SecurityLevelType>	成功返回当前diagnostic security level，失败返回错误码
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.4.19 Conversation::SetSecurityLevelNotifier

函数原型	ara::core::Result<void> SetSecurityLevelNotifier(std::function<void(SecurityLevelType)> notifier)	
函数功能	设置security level变化通知函数	
参数（IN）	notifier	diagnostic security level变化通知函数
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	设置成功返回void，设置失败返回错误码

使用说明	通过本接口设置通知函数，对应Conversation diagnostic security level变化，该通知函数将被调用
注意事项	• 可重入 • 该接口为同步接口

8.4.4.20 Conversation::ResetToDefaultSession

函数原型	ara::core::Result<void> ResetToDefaultSession()	
函数功能	重置当前converstion的diagnostic session为default session	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	重置成功，返回void，否则，返回错误
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.5 Condition

8.4.5.1 ConditionType

类名	ara::diag::ConditionType
命名空间	namespace ara::diag

语法	enum class ConditionType : uint8_t {
	kConditionFalse = 0x00,
	kConditionTrue = 0x01
	};

头文件	#include "ara/diag/condition.h"
说明	表示Condition的状态。

8.4.5.2 Condition::Condition

函数原型	explicit Condition (const ara::core::InstanceSpecifier &specifier);	
函数功能	Condition 类构造函数	
参数（IN）	specifier	Condition 实例名
返回值	None	-
使用说明	无	
注意事项	无	

8.4.5.3 Condition::~Condition

函数原型	~Condition () noexcept=default;	
函数功能	Condition类析构函数	
返回值	None	-
使用说明	无	
注意事项	无	

8.4.5.4 Condition::GetCondition

函数原型	ara::core::Result<ConditionType> GetCondition ();	
函数功能	获取Condition状态的接口	
返回值	ara::core::Result<ConditionType>	返回获取的Condition状态。
使用说明	无	
注意事项	无	

8.4.5.5 Condition::SetCondition

函数原型	ara::core::Result<void> SetCondition (ConditionType condition);	
函数功能	设置Condition的接口	
参数（IN）	condition	当前的condition状态。
返回值	ara::core::Result<void>	-
使用说明	无	
注意事项	无	

8.4.6 OperationCycle

8.4.6.1 OperationCycleType

类名	ara::diag::OperationCycleType
命名空间	namespace ara::diag

语法	enum class OperationCycleType : uint8_t {
	kOperationCycleStart = 0x00, // 启动、重启操作周期。
	kOperationCycleEnd = 0x01, // 结束操作周期。
	};

头文件	#include "ara/diag/operation_cycle.h"
说明	表示操作周期的状态信息。

8.4.6.2 OperationCycle::OperationCycle

函数原型	explicit OperationCycle (const ara::core::InstanceSpecifier &specifier);	
函数功能	OperationCycle 类构造函数	
参数（IN）	specifier	OperationCycle实例名
返回值	None	-
使用说明	无	
注意事项	无	

8.4.6.3 OperationCycle::~OperationCycle

函数原型	~OperationCycle() noexcept=default;	
函数功能	OperationCycle类析构函数	
返回值	None	-
使用说明	无	
注意事项	无	

8.4.6.4 OperationCycle::GetOperationCycle

函数原型	ara::core::Result<OperationCycleType> GetOperationCycle();	
函数功能	获取OperationCycle状态的接口	
返回值	ara::core::Result<OperationCycleType>	返回获取的OperationCycle状态。
使用说明	无	
注意事项	无	

8.4.6.5 OperationCycle::SetNotifier

函数原型	ara::core::Result<void> SetNotifier(std::function<void(OperationCycleType)> notifier);	
函数功能	注册OperationCycle状态变化通知函数的接口	
参数（IN）	notifier	注册OperationCycle变化通知函数。
返回值	ara::core::Result<void>	-
使用说明	注册通知器函数，如果操作周期更改，则会调用该函数。	
注意事项	无	

8.4.6.6 OperationCycle::SetOperationCycle

函数原型	ara::core::Result<void> SetOperationCycle(OperationCycleType operationCycle);	
函数功能	设置OperationCycle的接口	
参数（IN）	operationCycle	当前的OperationCycle状态。
返回值	ara::core::Result<void>	-
使用说明	无	
注意事项	无	

8.4.7 ServiceValidation

类名	ServiceValidation
命名空间	ara::diag

语法	class ServiceValidation {...};
头文件	#include "ara/diag/service_validation.h"
说明	制造商或供应商服务校验接口，用户需要继承该接口类，并重写实现其中的Validation和Confirmation接口，以完成服务校验功能

8.4.7.1 diag::ConfirmationStatusType

类名	ara::diag::ConfirmationStatusType
命名空间	namespace ara::diag

语法	enum class ConfirmationStatusType {
	kResPosOk = 0x00, //正响应成功发送
	kResPosNotOk = 0x01, //正响应未被成功发送
	kResNegOk = 0x02, //负响应成功发送
	kResNegNotOk = 0x03, //负响应未被成功发送
	kResPosSuppressed = 0x04, //正响应被抑制发送
	kResNegSuppressed = 0x05, //负响应被抑制发送
	kCanceled = 0x06, //对应conversation被取消
	kNoProcessingNoResponse = 0x07 //拒绝处理和应答该诊断请求
	};

头文件	#include "ara/diag/service_validation.h"
说明	服务处理状态信息。

8.4.7.2 ServiceValidation::ServiceValidation

函数原型	explicit ServiceValidation(const ara::core::InstanceSpecifier &specifier)	
函数功能	ServiceValidation服务构造函数	
参数（IN）	specifier	DiagnosticServiceValidationInterface的InstanceSpecifier
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	

注意事项	• 可重入 • 该接口为同步接口

8.4.7.3 ServiceValidation::~ServiceValidation

函数原型	virtual ~ServiceValidation() noexcept	
函数功能	ServiceValidation服务析构函数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.7.4 ServiceValidation::Validation

函数原型	virtual ara::core::Future<void> Validate(const ara::core::Span<std::uint8_t> &requestData, MetaInfo &metaInfo)=0	
函数功能	服务校验接口	
参数（IN）	requestData	诊断请求报文信息（包含SID字段）
	metaInfo	诊断请求MetaInfo
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Future<void>	校验结果，返回空或DiagnosticUdsErrorDomain内各错误码之一
使用说明	若用户ARXML中配置了制造商或供应商校验相关内容，用户需要以子类形式实现该Validation接口完成校验功能	
注意事项	• 可重入 • 该接口为同步接口	

8.4.7.5 ServiceValidation::Confirmation

函数原型	virtual ara::core::Future<void> Confirmation(ConfirmationStatusType status, MetaInfo &metaInfo)	
函数功能	服务处理完成确认接口	
参数（IN）	requestData	服务处理结果
	metaInfo	诊断请求MetaInfo
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Future<void>	校验结果，返回空或错误码
使用说明	若用户ARXML中配置了制造商或供应商校验相关内容，用户需要以子类形式实现该Confirmation接口完成服务处理确认功能。	
注意事项	• 可重入 • 该接口为同步接口	

8.4.7.6 ServiceValidation::Offer

函数原型	ara::core::Result<void> Offer()	
函数功能	提供ServiceValidation服务	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	提供服务返回值，若提供服务成功，ara::core::Result<void>含空值。 若提供服务失败ara::core::Result<void>含错误码，错误码如下： DiagOfferErrc::kAlreadyOfferd 服务已提供 DiagOfferErrc::kConfiguration Mismatch 服务配置错误
使用说明	该接口调用成功后，DM需要进行请求校验或消息处理确认时，将调用该ServiceValidation子类的Validation、Confirmation接口	

注意事项	• 可重入 • 该接口为同步接口

8.4.7.7 ServiceValidation::StopOffer

函数原型	void StopOffer()	
函数功能	停止提供ServiceValidation服务。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	该接口调用成功后，DM需要进行请求校验或消息处理确认时，将不会调用该ServiceValidation子类的Validation、Confirmation接口	
注意事项	• 可重入 • 该接口为同步接口	

8.4.8 SecurityAccess

8.4.8.1 SecurityAccess::KeyCompareResultType

类名	ara::diag::SecurityAccess
命名空间	namespace ara::diag

语法	enumclass KeyCompareResultType {
	kKeyValid = 0x00, // Key is valid
	kKeyInvalid = 0x01, // Key is invalid.
	};

头文件	#include "ara/diag/security_access.h"
说明	表示Key比较的状态。

8.4.8.2 SecurityAccess::SecurityAccess

函数原型	SecurityAccess(const ara::core::InstanceSpecifier &specifier, ReentrancyType reentrancyType)

函数功能	SecurityAccess类构造函数	
参数（IN）	specifier	DiagnosticSecurityAccessInterface的实例名
	reentrancyType	指定接口是可完全调用还是不可重入调用
返回值	None	-
使用说明	无	
注意事项	无	

8.4.8.3 SecurityAccess::~SecurityAccess

函数原型	~SecurityAccess()	
函数功能	SecurityAccess类析构函数	
返回值	None	-
使用说明	无	
注意事项	无	

8.4.8.4 SecurityAccess::GetSeed

函数原型	virtual ara::core::Future<ara::core::Span<std::uint8_t> > GetSeed(ara::core::Span<std::uint8_t> securityAccessDataRecord, MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0;	
函数功能	SecurityAccess获取种子接口	
参数（IN）	securityAccessDataRecord	Security Access的数据。
	metaInfo	请求的MetaInfo。
	cancellationHandler	设置当前会话是否被取消。
返回值	ara::core::Future<ara::core::Span<std::uint8_t> >	返回获取的种子。
使用说明	无	
注意事项	无	

8.4.8.5 SecurityAccess::CompareKey

函数原型	virtual ara::core::Future<KeyCompareResultType> CompareKey(ara::core::Span<std::uint8_t> key, MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0;	
函数功能	SecurityAccess比较密钥的接口	
参数（IN）	key	Security Access的数据。
	metaInfo	请求的MetaInfo。
	cancellationHandler	设置当前会话是否被取消。
返回值	ara::core::Future<KeyCompare ResultType>	返回获取的种子。
使用说明	当诊断请求完成时，将调用此方法，以通知结果。	
注意事项	无	

8.4.8.6 SecurityAccess::Offer

函数原型	ara::core::Result<void> Offer();	
函数功能	提供SecurityAccess服务的接口。	
返回值	ara::core::Result<void>	-
使用说明	此接口将使DM能够将请求消息转发到此处理程序。	
注意事项	无	

8.4.8.7 SecurityAccess::StopOffer

函数原型	void StopOffer ();	
函数功能	停止SecurityAccess服务的接口。	
返回值	无	-
使用说明	此接口将禁止转发来自DM的请求消息。	
注意事项	无	

8.4.9 CommunicationControl

类名	CommunicationControl
命名空间	ara::diag

语法	class CommunicationControl{...};
头文件	#include "ara/diag/communication_control.h"
说明	CommunicationControl服务接口，用户需要集成该接口类，并重写实现其中的CommCtrlRequest接口。

8.4.9.1 CommunicationControl::ComCtrlRequestParamsType

类名	ara::diag::CommunicationControl
命名空间	namespace ara::diag

语法	struct ComCtrlRequestParamsType {
	uint8_t controlType;
	uint8_t communicationType;
	uint16_t nodeIdentificationNumber;
	};

头文件	#include "ara/diag/communication_control.h"
说明	表示UDS 0x28 communicationControl服务请求中的各参数。

8.4.9.2 CommunicationControl::CommunicationControl

函数原型	explicit CommunicationControl(const ara::core::InstanceSpecifier &specifier, ReentrancyType reentrancyType)	
函数功能	CommunicationControl服务构造函数	
参数（IN）	specifier	CommunicationControlInterface的InstanceSpecifier
	reentrancyType	指定该CommunicationControlInterface内CommCtrlRequest接口的可重入性
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.9.3 CommunicationControl::~CommunicationControl

函数原型	virtual ~CommunicationControl() noexcept	
函数功能	CommunicationControl服务析构函数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.9.4 CommunicationControl::CommCtrlRequest

函数原型	virtual ara::core::Future<void> CommCtrlRequest (ComCtrlRequestParamsType controlType, MetaInfo &metaInfo, CancellationHandler cancellationHandler)=0;	
函数功能	请求服务下载接口	
参数（IN）	controlType	UDS请求消息中各参数组成的结构体
	cancellationHandler	coversation取消通知接口
参数（INOUT）	metaInfo	诊断消息元信息，可用于获取该UDS请求对应conversation
参数（OUT）	None	NA
返回值	ara::core::Future<void>	校验结果，返回请求下载服务正响应或DiagnosticUdsErrorDomain内各错误码之一
使用说明	生成接口类中该接口为纯虚接口，用户需要在子类中重写该CommCtrlRequest函数，DM收到CommunicationControl请求后，将调用子类重写的CommCtrlRequest接口。	
注意事项	• 可重入 • 该接口为同步接口	

8.4.9.5 CommunicationControl::Offer

函数原型	ara::core::Result<void> Offer()	
函数功能	提供CommunicationControl服务	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	提供服务返回值，若提供服务成功，ara::core::Result<void>含空值。 若提供服务失败ara::core::Result<void>含错误码，错误码如下： DiagOfferErrc::kAlreadyOfferd 服务已提供 DiagOfferErrc::kConfiguration Mismatch 服务配置错误
使用说明	该接口调用成功后，DM收到CommunicationControl请求，将调用该CommunicationControl子类的CommCtrlRequest接口。	
注意事项	• 可重入 • 该接口为同步接口	

8.4.9.6 CommunicationControl::StopOffer

函数原型	void StopOffer()	
函数功能	停止提供CommunicationControl服务。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	该接口调用后，DM收到CommunicationControl请求，不会调用该CommunicationControl子类的CommCtrlRequest接口。	
注意事项	• 可重入 • 该接口为同步接口	

8.4.10 DownloadService

类名	DownloadService
命名空间	ara::diag
语法	class DownloadService {...};
头文件	#include "ara/diag/download.h"
说明	下载服务接口，用户需要继承该接口类，并重写实现其中的RequestDownload、DownloadData、RequestDownloadExit接口，以完成下载服务功能。

8.4.10.1 DownloadService::OperationOutput

类名	ara::diag::DownloadService::OperationOutput
命名空间	namespace ara::diag

语法	struct OperationOutput {
	ara::core::Vector<std::uint8_t> responseData;
	};

头文件	#include "ara/diag/download.h"
说明	表示Key比较的状态。

8.4.10.2 DownloadService::DownloadService

函数原型	explicit DownloadService(const ara::core::InstanceSpecifier &specifier, ReentrancyType reentrancyType)	
函数功能	DownloadService	
参数（IN）	specifier	DownloadServiceInterface的InstanceSpecifier
	reentrancyType	指定该DownloadServiceInterface内RequestDownload、DownloadData、RequestDownloadExit接口的可重入性
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA

使用说明	注意每个Diagnostic Server的仅能构造一个DownloadService服务实例。
注意事项	• 可重入 • 该接口为同步接口

8.4.10.3 DownloadService::~DownloadService

函数原型	virtual ~DownloadService() noexcept	
函数功能	DownloadService服务析构函数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.10.4 DownloadService::RequestDownload

函数原型	virtual ara::core::Future<OperationOutput>RequestDownload(std::uint8_t dataFormatIdentifier, std::uint8_t addressAndLengthFormatIdentifier, ara::core::Span<std::uint8_t> memoryAddressAndSize, MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0	
函数功能	请求服务下载接口	
参数（IN）	dataFormatIdentifier	UDS dataFormat Identifier字段
	addressAndLengthFormatIdentifier	UDS addressAndLengthFormatIdentifier字段
	memoryAddressAndSize	UDS memoryAddressAndSize字段
	cancellationHandler	coversation取消通知接口
参数（INOUT）	metaInfo	诊断消息元信息，可用于获取该UDS请求对应conversation

参数（OUT）	None	NA
返回值	ara::core::Future<OperationOutput>	返回请求下载服务正响应或DiagnosticUdsErrorDomain内各错误码之一
使用说明	生成接口类中该接口为纯虚接口，用户需要在子类中重写该RequestDownload函数，DM收到RequestDownload请求后，将调用子类重写的RequestDownload接口。	
注意事项	• 可重入 • 该接口为同步接口 • 由于RequestDownloal服务的response需要包含maxNumberOfBlockLength，与R2011规范（规范返回值ara::core::Future<void>）不同，该接口返回值为ara::core::Future<OperationOutput>，用于填写response响应的maxNumberOfBlockLength参数，另根据Adaptive autosar，Diagnostic Server instance和External service processor的地址和数据长度信息均存储于uint64数据类型中，因此，maxNumberOfBlockLength<=8，若用户返回字节数maxNumberOfBlockLength>8，返回错误码kUploadDownloadNotAccepted。	

8.4.10.5 DownloadService::DownloadData

函数原型	virtual ara::core::FutureDownloadData(ara::core::Span<std::uint8_t> transferRequestParameterRecord, MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0	
函数功能	服务校验接口	
参数（IN）	requestData	诊断请求报文信息（包含SID字段）
	metaInfo	诊断请求MetaInfo
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Future<void>	校验结果，返回空或DiagnosticUdsErrorDomain内各错误码之一
使用说明	若用户ARXML中配置了制造商或供应商校验相关内容，用户需要以子类形式实现该Validation接口完成校验功能。	
注意事项	• 可重入 • 该接口为同步接口	

8.4.10.6 DownloadService::RequestDownloadExit

函数原型	virtual ara::core::FutureRequestDownloadExit(ara::core::Span<std::uint8_t> transferRequestParameterRecord, MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0	
函数功能	服务校验接口	
参数（IN）	requestData	诊断请求报文信息（包含SID字段）
	metaInfo	诊断请求MetaInfo
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Future<void>	校验结果，返回空或DiagnosticUdsErrorDomain内各错误码之一
使用说明	若用户ARXML中配置了制造商或供应商校验相关内容，用户需要以子类形式实现该Validation接口完成校验功能。	
注意事项	• 可重入 • 该接口为同步接口	

8.4.10.7 DownloadService::Offer

函数原型	ara::core::Result<void> Offer()	
函数功能	提供DownloadService服务	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	ara::core::Result<void>	提供服务返回值，若提供服务成功，ara::core::Result<void>含空值。 若提供服务失败ara::core::Result<void>含错误码，错误码如下： DiagOfferErrc::kAlreadyOfferd 服务已提供 DiagOfferErrc::kConfigurationMismatch 服务配置错误

使用说明	该接口调用成功后，DM收到RequestDownload、TransferData（下载请求后传输下载数据）、RequestTransferExit（请求下载退出），将调用该DownloadService子类的RequestDownload、DownloadData、RequestDownloadExit接口。
注意事项	• 可重入 • 该接口为同步接口

8.4.10.8 DownloadService::StopOffer

函数原型	void StopOffer()	
函数功能	停止提供DownloadService服务。	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	该接口调用后，DM收到RequestDownload、TransferData（下载请求后传输下载数据）、RequestTransferExit（请求下载退出），不会调用该DownloadService子类的RequestDownload、DownloadData、RequestDownloadExit接口。	
注意事项	• 可重入 • 该接口为同步接口	

8.4.11 ECUResetRequest

8.4.11.1 EcuResetRequest::LastResetType

类名	ara::diag::SecurityAccess
命名空间	namespace ara::diag

语法	enumclass LastResetType : std::uint32_t {
	kRegular = 0, // 定期停机。
	kUnexpected = 1, // 意外复位。
	kSoftReset = 2, // 诊断软复位。
	kHardReset = 3, // 诊断硬复位。
	kKeyOffOnReset = 4, // 诊断按键关闭复位。
	kCustomReset = 5, // 诊断自定义复位。
	};

头文件	#include "ara/diag/ecu_reset_request.h"
说明	请求复位的类型。

8.4.11.2 EcuResetRequest::ResetRequestType

类名	ara::diag::SecurityAccess
命名空间	namespace ara::diag

语法	enum class ResetRequestType : std::uint32_t {
	kSoftReset = 0, // 软复位。
	kHardReset = 1, // 硬复位。
	kKeyOffOnReset = 2, // 按键关闭复位。
	kCustomReset = 3, // 自定义复位。
	};

头文件	#include "ara/diag/ecu_reset_request.h"
说明	请求复位的类型。

8.4.11.3 EcuResetRequest::StopOffer

函数原型	void StopOffer ();	
函数功能	停止EcuResetRequest服务的接口。	
返回值	无	-
使用说明	此接口将禁止转发来自DM的请求消息。	
注意事项	无	

8.4.11.4 EcuResetRequest::Offer

函数原型	ara::core::Result<void> Offer();

函数功能	提供EcuResetRequest服务的接口。	
返回值	ara::core::Result<void>	-
使用说明	此接口将使DM能够将请求消息转发到此处理程序。	
注意事项	无	

8.4.11.5 EcuResetRequest::GetLastResetCause

函数原型	virtual ara::core::Result<LastResetType> GetLastResetCause() = 0;	
函数功能	获取上次复位的类型	
返回值	ara::core::Result<LastResetType>	上次机器复位的类型。
使用说明	无	
注意事项	无	

8.4.11.6 EcuResetRequest::RequestReset

函数原型	virtual ara::core::Future<void> RequestReset(ResetRequestType resetType, ara::core::Optional<std::uint8_t> id, const MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0;	
函数功能	请求ECU复位接口	
参数（IN）	resetType	请求复位的类型。
	id	自定义重置类型的ID。将只进行评估当resetType为“自定义”时。
	metaInfo	请求的MetaInfo。
	cancellationHandler	设置当前会话是否被取消。
返回值	ara::core::Future<void>	-
使用说明	任何EcuReset子功能都可以调用，EnableRapidShutdown、DisableRapidShutdown除外。 状态管理需要仔细评估是否请求重新启动部分或整个计算机。一旦接受复位请求，状态管理必须依赖 ExecuteReset()触发的结果。	
注意事项	无	

8.4.11.7 EcuResetRequest::ExecuteReset

函数原型	virtual void ExecuteReset(MetaInfo metaInfo) = 0;	
函数功能	执行ECU复位接口	
参数（IN）	metaInfo	请求的MetaInfo。
返回值	无	-
使用说明	状态管理必须执行请求的重置。DM向测试仪发送响应消息后调用。	
注意事项	无	

8.4.11.8 EcuResetRequest::EnableRapidShutdown

函数原型	virtual ara::core::Future<void> EnableRapidShutdown(bool enable, const MetaInfo &metaInfo, CancellationHandler cancellationHandler) = 0;	
函数功能	使能、去使能快速关闭接口	
参数（IN）	enable	当enable设置为true时，快速关闭将启用，将enable设置为false将禁用快速关闭。
	metaInfo	请求的MetaInfo。
	cancellationHandler	设置当前会话是否被取消。
返回值	ara::core::Future<void>	-
使用说明	无	
注意事项	无	

8.4.11.9 EcuResetRequest::~EcuResetRequest

函数原型	virtual ~EcuResetRequest() noexcept = default;	
函数功能	EcuResetRequest类析构函数	
返回值	None	-
使用说明	无	
注意事项	无	

8.4.11.10 EcuResetRequest::EcuResetRequest

函数原型	explicit EcuResetRequest(const ara::core::InstanceSpecifier &specifier);	
函数功能	EcuResetRequest类构造函数	
参数（IN）	specifier	将此实例与manifest中的PortPrototype链接的InstanceSpecifier
返回值	None	-
使用说明	无	
注意事项	无	

8.4.12 DiagnosticManagerConfig

类名	DiagnosticManagerConfig
命名空间	rtf::diag
语法	class DiagnosticManagerConfig{...};
头文件	#include "rtf/diag/diagnostic_manager.h"
说明	DM进程相关配置接口类

8.4.12.1 DiagnosticManagerConfig::Create

函数原型	static std::unique_ptr<DiagnosticManagerConfig> Create(const std::string &configFileDir, const std::string &perSpecifier)	
函数功能	创建DM进程配置	
参数（IN）	configFileDir	DM进程配置文件目录
	perSpecifier	DM进程持久化功能InstanceSpecifier
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	std::unique_ptr<DiagnosticManagerConfig>	DM进程配置
使用说明	指定DM进程相关配置参数，生成DM进程配置	

注意事项	• 可重入 • 该接口为同步接口

8.4.12.2 DiagnosticManagerConfig::GetConfigDir

函数原型	std::string GetConfigFileDir() const	
函数功能	获取DM进程配置文件目录	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	std::string	DM进程配置文件目录
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.12.3 DiagnosticManagerConfig::GetPerSpecifier

函数原型	std::string GetPerSpecifier() const	
函数功能	获取PER持久化数据库的名称	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	std::string	PER持久化数据库的名称
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.12.4 DiagnosticManagerConfig::~DiagnosticManager

函数原型	~DiagnosticManagerConfig()
函数功能	DiagnosticManagerConfig析构函数

参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.13 DiagnosticManager

类名	DiagnosticManager
命名空间	rtf::diag
语法	class DiagnosticManager{...};
头文件	#include "rtf/diag/diagnostic_manager.h"
说明	诊断功能管理接口类

8.4.13.1 DiagnosticManager::Create

函数原型	static std::shared_ptr<DiagnosticManager> Create(std::unique_ptr<DiagnosticManagerConfig> config)	
函数功能	创建诊断功能管理模块	
参数（IN）	config	诊断功能管理相关配置
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	std::shared_ptr<DiagnosticManager>	诊断功能管理模块
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

8.4.13.2 DiagnosticManager::Initialize

函数原型	bool Initialize()	
函数功能	初始化诊断功能管理	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	bool	ture: 初始化成功； false：初始化失败
使用说明	初始化过程将会读取DM进程配置，初始化DCM、DEM模块	
注意事项	• 可重入 • 该接口为同步接口	

8.4.13.3 DiagnosticManager::Start

函数原型	void Start()	
函数功能	启动诊断管理	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	调用该接口后，DM进程开始接收UDS请求，与应用发送的事件上报、DCM信息（如Conversation信息）、DEM信息（DTC信息）查询请求等	
注意事项	• 可重入 • 该接口为同步接口	

8.4.13.4 DiagnosticManager::Stop

函数原型	void Stop()	
函数功能	停止诊断管理	
参数（IN）	None	NA

参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	调用该接口后，DM进程将停止接收UDS请求，与应用发送的事件上报、DCM信息（如Conversation信息）、DEM信息（DTC信息）查询请求等	
注意事项	• 可重入 • 该接口为同步接口	

8.4.13.5 DiagnosticManager::~DiagnosticManager

函数原型	~DiagnosticManager()	
函数功能	DiagnosticManager析构函数	
参数（IN）	None	NA
参数（INOUT）	None	NA
参数（OUT）	None	NA
返回值	None	NA
使用说明	无	
注意事项	• 可重入 • 该接口为同步接口	

9 Tsync

9.1 数据类型

9.1.1 TimeBaseType

类名	ara::tsync::TimeBaseType
命名空间	ara::tsync

语法	enum class TimeBaseType : std::uint8_t {
	kSynchMasterTBType = 0U,
	kSynchSlaveTBType = 1U,
	kOffsetMasterTBType = 2U,
	kOffsetSlaveTBType = 3U,
	kPureLocalTBType = 4U,
	};

头文件	#include "ara/tsync/time_base_type.h"
说明	kSynchMasterTBType：同步主时基 kSynchSlaveTBType：同步从时基 kOffsetMasterTBType：偏移主时基 kOffsetSlaveTBType：偏移从时基 kPureLocalTBType：本地时基

说明

当前仅支持kSynchMasterTBType与kSynchSlaveTBType。

9.1.2 StatusFlag

类名	ara::tsync::StatusFlag
命名空间	ara::tsync

语法	enum class StatusFlag: std::uint8_t {
	kTimeOut= 0U,
	kSynchronized,
	kSynchToGateway,
	kTimeLeapFuture,
	kTimeLeapPast,
	kHasDLS,
	kDLSActive,
	};

头文件	#include "ara/tsync/time_base_status.h"
说明	描述了时基的状态。 kTimeOut：时基在一定时间内没有同步 kSynchronized：时基至少同步一次 kSynchToGateway：时基与网关同步 kTimeLeapFuture：已进行大于某个阈值的调整，且时间是向未来时间同步 kTimeLeapPast：已进行大于某个阈值的调整，且时间是向过去时间同步 kHasDLS：支持夏令时 kDLSActive： 夏令时已启用

说明

当前仅支持kTimeOut、kSynchronized、kTimeLeapFuture及kTimeLeapPast四种时基状态。

9.1.3 Identities

时基标识用于在本地区分同一时钟类型的多种表现形式。

类名	ara::tsync::SynchMasterIdentity
命名空间	ara::tsync

语法	enum class SynchMasterIdentity: std::uint8_t {
	k0 = 0,
	k1= 1
	};

头文件	#include "ara/tsync/time_base_type.h"
说明	SynchMasterIdentity 用于在本地区分 SynchMasterTB 的多种表现形式。

类名	ara::tsync::SynchSlaveIdentity
命名空间	ara::tsync

语法	enum class SynchSlaveIdentity: std::uint8_t {
	k0 = 0,
	k1= 1
	};

头文件	#include "ara/tsync/time_base_type.h"
说明	SynchSlaveIdentity用于在本地区分 SynchSlaveTB的多种表现形式。

9.1.4 NotificationType

类名	ara::tsync::NotificationType
命名空间	ara::tsync

语法	enum class NotificationType: std::uint8_t {
	SYNCHRONIZED = 0,
	SYNCHRONIZATION_TIMEOUT,
	TIME_LEAP_FUTURE,
	TIME_LEAP_PAST
	};

头文件	#include "ara/tsync/synch_slave_tb.h"
说明	时基当前的状态。

9.2 SynchMasterTB

说明

当前主时基接口交付相较于AP规范有裁剪，具体交付接口以此文档为准。

接口类定义

类名	ara::tsync::SynchMasterTB
命名空间	namespace ara::tsync
语法	template <SynchMasterIdentity ID, typename ReferenceClock = std::chrono::steady_clock> class SynchMasterTB {...};
头文件	#include "ara/tsync/synch_master_tb.h"
说明	SynchMasterTB 可用于通过分配给外部时钟的时钟资源提供时钟信息。为了能够使用任何类型的 SynchMasterTB，需要通过使用适当的 InstanceSpecifier 调用 FindResource() 方法来配置它们。 SynchMasterIdentity ID： 用于局部区分SynchMasterTB的不同表现形式。 typename ReferenceClock = std::chrono::steady_clock： 内部用于测量时间进度的时钟。默认为std::chrono::steady_clock。

成员类型定义

类型名	ara::tsync::SynchMasterTB::duration
范围	struct ara::tsync::SynchMasterTB
语法	using ara::tsync::SynchMasterTB< ID, ReferenceClock >::duration = std::chrono::nanoseconds;
头文件	#include "ara/tsync/synch_master_tb.h"
说明	成员类型别名，用于表示此时钟使用的默认持续时间单位。

类型名	ara::tsync::SynchMasterTB::period
范围	struct ara::tsync::SynchMasterTB
语法	using ara::tsync::SynchMasterTB< ID, ReferenceClock >::period = duration::period;
头文件	#include "ara/tsync/synch_master_tb.h"

说明	成员类型别名，用于表示此时钟使用的默认持续时间周期。

类型名	ara::tsync::SynchMasterTB::referenceClock
范围	struct ara::tsync::SynchMasterTB
语法	using ara::tsync::SynchMasterTB< ID, ReferenceClock >::referenceClock = ReferenceClock;
头文件	#include "ara/tsync/synch_master_tb.h"
说明	存储 ReferenceClock 类型的成员类型别名。

类型名	ara::tsync::SynchMasterTB::rep
范围	struct ara::tsync::SynchMasterTB
语法	using ara::tsync::SynchMasterTB< ID, ReferenceClock >::rep = duration::rep;
头文件	#include "ara/tsync/synch_master_tb.h"
说明	成员类型别名，用于表示此时钟使用的默认持续时间类型。

类型名	ara::tsync::SynchMasterTB::time_point
范围	struct ara::tsync::SynchMasterTB
语法	using ara::tsync::SynchMasterTB< ID, ReferenceClock >::time_point = std::chrono::time_point<SynchMasterTB, duration>;
头文件	#include "ara/tsync/synch_master_tb.h"
说明	成员类型别名，表示此时钟使用的 time_point 类型。

成员函数定义

函数原型	static bool ara::tsync::SynchMasterTB< ID, ReferenceClock >::FindResource (ara::core::StringView instanceSpecifier);	
函数功能	配置SynchMasterTB的函数。	
参数（IN）	instanceSpecifier	允许查询 SynchMasterTB 配置的 ID。 输入对应的 InstanceId。

参数（INOUT）	None	-
返回值	bool	如果可以成功获取属于实例说明符的资源，则为 true，否则为 false。
使用说明	• 在调用其它函数前，必须调用此函数查找对应的时基源。 • 由于ara::core::InstanceSpecifier类型必须带“/”字符，而非具体ID，与规范“支持不同时基ID”冲突，因此此处使用ara::core::StringView。	
注意事项	线程安全，当前仅支持instance ID为“0”和“1”。	

函数原型	static TimeBaseStatus<SynchMasterTB> ara::tsync::SynchMasterTB< ID, ReferenceClock >::GetTimeBaseStatus ();	
函数功能	获取时钟当前状态快照的方法。这包括状态标志、时钟配置和实际时间值。	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	TimeBaseStatus<SynchMasterTB>	包含所有时钟的特定TimeBaseStatus相关的时钟信息。
使用说明	无	
注意事项	线程安全	

函数原型	static constexpr TimeBaseType ara::tsync::SynchMasterTB< ID, ReferenceClock >::GetType ();	
函数功能	获取时基类型。	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	TimeBaseType	返回TimeBaseType::kSynchMaster。
使用说明	无	
注意事项	线程安全	

函数原型	template <typename Duration = duration> static std::chrono::time_point<SynchMasterTB, Duration> ara::tsync::SynchMasterTB< ID, ReferenceClock >::now ();	
函数功能	获取当前时间点。	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	std::chrono::time_point<SynchMasterTB, Duration>	当前时间点。
使用说明	无	
注意事项	线程安全	

函数原型	template <typename Duration = duration> static void ara::tsync::SynchMasterTB< ID, ReferenceClock >::SetTime(std::chrono::time_point< SynchMasterTB, Duration > timePoint);	
函数功能	设置时间，此函数会触发同步消息的传输。	
参数（IN）	timePoint	设置的时间
参数（INOUT）	None	-
返回值	None	-
使用说明	模板参数Duration： 作为参数传递的时间点的持续时间类型。	
注意事项	线程安全，参数与系统启动是初始化的时间差不能超过14*10^13（ns）	

函数原型	template <typename Duration = duration> static void ara::tsync::SynchMasterTB< ID, ReferenceClock >::UpdateTime (std::chrono::time_point< SynchMasterTB, Duration > timePoint);	
函数功能	更新时间。此函数不会触发同步消息的传输。	
参数（IN）	timePoint	设置的时间
参数（INOUT）	None	-
返回值	None	-

使用说明	模板参数Duration：作为参数传递的时间点的持续时间类型。
注意事项	线程安全，参数与系统启动是初始化的时间差不能超过14*10^13（ns）

函数原型	template <typename Duration = duration> static void ara::tsync::SynchMasterTB< ID, ReferenceClock >::StopFindResource();	
函数功能	与时基断开联系。	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	None	-
使用说明	与时间源断开联系。进程退出前，调用此函数释放资源。	
注意事项	线程安全	

9.3 SynchSlaveTB

说明

当前从时基接口交付相较于AP规范有裁剪，且注册回调功能基于实际业务开发，通过监听同步状态和时间跳跃状态触发回调，具体交付接口以此文档为准。

接口类定义

类名	ara::tsync::SynchSlaveTB
命名空间	namespace ara::tsync
语法	template <SynchSlaveIdentityID, typename ReferenceClock = std::chrono::steady_clock> class SynchSlaveTB {...};
头文件	#include "ara/tsync/synch_slave_tb.h"
说明	SynchSlaveTB可用于通过分配给外部时钟的时钟资源提供时钟信息。为了能够使用任何类型的 SynchSlaveTB，需要通过使用适当的 InstanceSpecifier 调用 FindResource() 方法来配置它们。 SynchSlaveIdentity ID： 用于局部区分SynchSlaveTB的不同表现形式。 typename ReferenceClock = std::chrono::steady_clock： 内部用于测量时间进度的时钟。默认为std::chrono::steady_clock。

成员类型定义

类型名	ara::tsync::SynchSlaveTB::duration
范围	struct ara::tsync::SynchSlaveTB
语法	using ara::tsync::SynchSlaveTB< ID, ReferenceClock >::duration = std::chrono::nanoseconds;
头文件	#include "ara/tsync/synch_slave_tb.h"
说明	成员类型别名，用于表示此时钟使用的默认持续时间单位。

类型名	ara::tsync::SynchSlaveTB::period
范围	struct ara::tsync::SynchSlaveTB
语法	using ara::tsync::SynchSlaveTB< ID, ReferenceClock >::period = duration::period;
头文件	#include "ara/tsync/synch_slave_tb.h"
说明	成员类型别名，用于表示此时钟使用的默认持续时间周期。

类型名	ara::tsync::SynchSlaveTB::referenceClock
范围	struct ara::tsync::SynchSlaveTB
语法	using ara::tsync::SynchSlaveTB< ID, ReferenceClock >::referenceClock = ReferenceClock;
头文件	#include "ara/tsync/synch_slave_tb.h"
说明	存储 ReferenceClock 类型的成员类型别名。

类型名	ara::tsync::SynchSlaveTB::rep
范围	struct ara::tsync::SynchSlaveTB
语法	using ara::tsync::SynchSlaveTB< ID, ReferenceClock >::rep = duration::rep;
头文件	#include "ara/tsync/synch_slave_tb.h"
说明	成员类型别名，用于表示此时钟使用的默认持续时间类型。

类型名	ara::tsync::SynchSlaveTB::time_point

范围	struct ara::tsync::SynchSlaveTB
语法	using ara::tsync::SynchSlaveTB< ID, ReferenceClock >::time_point = std::chrono::time_point<SynchMasterTB, duration>;
头文件	#include "ara/tsync/synch_slave_tb.h"
说明	成员类型别名，表示此时钟使用的 time_point 类型。

成员函数定义

函数原型	static bool ara::tsync::SynchSlaveTB< ID, ReferenceClock >::FindResource (ara::core::StringView instanceSpecifier);	
函数功能	配置SynchSlaveTB的函数。	
参数（IN）	instanceSpecifier	允许查询 SynchSlaveTB配置的ID。 输入对应的InstanceId。
参数（INOUT）	None	-
返回值	bool	如果可以成功获取属于实例说明符的资源，则为 true，否则为 false。
使用说明	• 在调用其它函数前，必须调用此函数查找对应的时基源。 • 由于ara::core::InstanceSpecifier类型必须带“/”字符，而非具体ID，与规范“支持不同时基ID”冲突，因此此处使用ara::core::StringView。	
注意事项	线程安全，当前仅支持instance ID为“0”和“1”。	

函数原型	static TimeBaseStatus<SynchSlaveTB> ara::tsync::SynchSlaveTB< ID, ReferenceClock >::GetTimeBaseStatus ();	
函数功能	获取时钟当前状态快照的方法。这包括状态标志、时钟配置和实际时间值。	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	TimeBaseStatus<SynchSlaveTB>	包含所有时钟的特定TimeBaseStatus相关的时钟信息。
使用说明	无	
注意事项	线程安全	

函数原型	static constexpr TimeBaseType ara::tsync::SynchSlaveTB< ID, ReferenceClock >::GetType ();	
函数功能	获取时基类型。	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	TimeBaseType	返回TimeBaseType::kSynchSlave。
使用说明	无	
注意事项	线程安全	

函数原型	template <typename Duration = duration> static std::chrono::time_point<SynchSlaveTB, Duration> ara::tsync::SynchSlaveTB< ID, ReferenceClock >::now ();	
函数功能	获取当前时间点。	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	std::chrono::time_point<SynchSlaveTB, Duration>	当前时间点。
使用说明	无	
注意事项	线程安全	

函数原型	static double ara::tsync::SynchSlaveTB< ID, ReferenceClock >::GetRateDeviation ();	
函数功能	获取速率偏差。	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	double	当前速率偏差。
使用说明	无	
注意事项	线程安全	

函数原型	static void ara::tsync::SynchSlaveTB< ID, ReferenceClock >::RegisterTimeBaseNotification (const std::function<void(const NotificationType type)>& callBackFunc);	
函数功能	注册收到异常同步状态的回调函数	
参数（IN）	callBackFunc	获取异常同步状态的回调函数
参数（INOUT）	None	-
返回值	None	-
使用说明	无	
注意事项	线程安全	

函数原型	template <typename Duration = duration> static void ara::tsync::SynchSlaveTB< ID, ReferenceClock >::StopFindResource();	
函数功能	与时基断开联系	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	None	-
使用说明	与时间源断开联系。进程退出前，调用此函数释放资源。	
注意事项	线程安全	

9.4 TimeBaseStatus

接口类定义

类名	ara::tsync::TimeBaseStatus
命名空间	namespace ara::tsync
语法	template <typename TB, typename STB = TB> class TimeBaseStatus {...};
头文件	#include "ara/tsync/time_base_status.h"
说明	包含时钟所有相关信息。

成员函数定义

函数原型	ara::tsync::TimeBaseStatus< TB, STB >::TimeBaseStatus (const ara::core::Vector< StatusFlag > &statusFlags, std::uint8_t updateCounter);	
函数功能	构造函数	
参数（IN）	statusFlags	时基当前的状态标志位集
	updateCounter	时基当前的更新次数
参数（INOUT）	None	-
返回值	None	-
使用说明	无	
注意事项	线程安全	

函数原型	TB::time_point ara::tsync::TimeBaseStatus< TB, STB >::GetCreationTime() const;	
函数功能	获得创建此实例的时间。	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	TB::time_point	创建此实例的时间。
使用说明	无	
注意事项	线程安全	

函数原型	TimeBaseStatus<STB, STB> ara::tsync::TimeBaseStatus< TB, STB >::GetSynchStatus () const;	
函数功能	获取此实例的副本。	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	TimeBaseStatus< STB, STB >	此实例的副本。
使用说明	无	
注意事项	线程安全	

函数原型	std::uint8_t ara::tsync::TimeBaseStatus< TB, STB >::GetUpdateCounter() const;	
函数功能	获取时基的更新次数。	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	std::uint8_t	创建此实例的时基的更新次数。
使用说明	无	
注意事项	线程安全	

函数原型	bool ara::tsync::TimeBaseStatus< TB, STB >::IsStatusFlagActive (StatusFlag flag) const;	
函数功能	判断指定的状态标志位是否存在	
参数（IN）	flag	被判断的状态标志位
参数（INOUT）	None	-
返回值	bool	true: 此标志位存在； 反之，false。
使用说明	无	
注意事项	线程安全	

函数原型	bool ara::tsync::TimeBaseStatus< TB, STB >::operator== (const TimeBaseStatus &other) const;	
函数功能	==操作函数	
参数（IN）	other	被比较的另一个实例
参数（INOUT）	None	-
返回值	bool	true: 两个实例相同；false: 两个实例不同
使用说明	无	
注意事项	线程安全，包含对状态标记数组、创建时间及更新计数的比较。	

函数原型	ara::tsync::TimeBaseStatus< TB, STB >::operator typename TB::time_point () const;	
函数功能	转换操作函数	
参数（IN）	None	-
参数（INOUT）	None	-
返回值	TB::time_point	创建此实例的时间。
使用说明	转换运算符以启用将 TimeBaseStatuse 隐式转换为其基础 time_points 以提高可用性。	
注意事项	线程安全	

