1. **(2016判断)既然IDL编译器能够根据CORBA组件接口的定义自动生成接口的实现,即能够自动生成客户桩类供客户程序使用,那么,CORBA组件就没有必要由开发人员实现了.**

回答:IDL编译器的确能根据组件接口定义生成客户桩类(stub),但是由于IDL本身并不能编写可执行语句,所以stub类对接口的实现无法完成任何业务逻辑,而业务逻辑部分的实现由服务器应用程序上下文中的Servant给出,这部分必须由程序员使用具体的开发语言进行编写.

1. **(2016简答)IDL编译器的输入是什么?主要输出结果是什么?每个文件中的代码的作用是什么?**

回答:输入idl文件,其中使用idl语言定义了一些接口以及相关的其他定义,根据所使用的IDL编译器的不同,输出是6-7个文件,其中是使用具体编程语言实现的接口,以生成java实现为例,假设我们定义了一个名为InterfaceName的接口,将会生成以下文件:

1. \_IntefaceNameStub.java

CORBA组件在客户端侧的桩代码,用于将客户端侧的请求编码并向服务器发出.但是客户端代码通常并不直接使用桩代码.

1. IntefaceName.java

操作接口的派生接口,客户端通常直接操作此接口的方法来实现对远程CORBA组件的调用.

1. IntefaceNameHelper.java

辅助工具类,其中定义了许多使用功能和支持功能的静态方法.这些方法提供的功能有:

--从Any对象提取或向Any对象插入对象（extract和insert方法）；

--从输入／输出流读写对象（read和write方法）；

--获取对象的库标识和类型码（id和type方法）；

--绑定对象与类型转换操作（bind和narrow方法）等等。

1. IntefaceNameHolder.java

为传递对象提供支持的类,具体来讲:

--IDL有三种参数传递方式：in、out和inout。

--in类型的参数以及返回结果与Java的传参方式与返回方式完全相同。

--out和inout两种类型的参数允许参数具有返回结果的能力，无法直接映射到Java语言(或者其他不支持通过参数传递结果的编程语言)的参数传递机制，这时Holder类为传递out和inout参数提供了一个托架（holder）。

\*In表明实际参数从客户程序传向对象实现

\*Out表明数据从实现对象传递给客户程序，并且对象实现无需从客户程序获取参数的初始值

\*Inout表明数据从客户程序传给对象实现，然后经对象实现加工后再返回给客户程序

1. IntefaceNameOperations.java

定义了程序员使用idl语言描述的接口中的所有常量和方法

1. IntefaceNamePOA.java

定义了CORBA对象的服务端框架代码,功能有:

-解包in类型的参数并将参数传递给对象实现。

-打包返回值与所有out类型的参数。

--打包（marshal）：指将特定程序设计语言描述的数据类型转换为CORBA的IIOP流格式。

--解包（unmarshal）：从IIOP流格式转换为依赖于具体程序设计语言的数据结构。

\*编写对象实现的最简单途径是继承这些POA类，即把它们作为对象实现的基类。

1. IntefaceNmaePOATie.java(有些版本的IDL编译器可能并不提供这个文件)

用于创建并部署应用程序的类,

1. **(2015判断)IDL编译器能根据IDL描述的接口自动生成特定程序语言stub类,stub类是一个实现了给借口的具体来(Concrete Class).既然IDL编译器具备根据接口自动生成改接口的实现的能力那么,利用面向对象方法实现某个类时,只需要利用IDL把这个类拟完成的全部成员函数以一个接口的形式定义出来即可,根本不需要实现这些成员函数,IDL编译器会自动实现.**

回答:生成特定语言的stub类中不包含具体接口的实现,而是生成一系列的桩代码,提供了为用户有效创建对象并发送请求的机制.而接口所定义的方法和函数是在Server方的实现对象Servant中实现的.另一方面,接口方法的实现根本无法预测,IDL编译器不可能只根据函数原型猜测出函数的具体业务逻辑,让IDL提供接口的具体实现是不现实的.

1. **(2015简答)概述利用IDL编译器构造一个CORBA组件的一般过程.**
2. **(2014简答)根据你对CORBA组建技术中的IDL编译器的理解,利用RUP方法中的需求模型描述形式,给出IDL编译器的需求模型,说明及要求包括:把整个ID编译器看做黑盒,不考虑其内部构造,术语表中的术语不少于6个,每个术语给出清晰的解释,所有的用例的参与者总和不能少于2个,用例的个数不能少于2个.**
3. **(2012简答) 简述IDL编译器自动生成客户桩stub代码的基本原理(自动生成stub的设计思路)**
4. **(2011简答)若把IDL编译器看成一个软件系统,请根据你对相关知识的理解,利用术语表以及UML用例图模型,概要描述该系统的需求允许但不是必须使用补充性说明文档.**
5. **(2011简答)从实现角度概述IDL编译器自动生成客户桩(IDL stub)以及服务器框架(IDL Skeleton)的基本原理**
6. **(2011简答)结合CORBA组件中的ORB和EJB组件技术中的EJB容器,简述你对”中间件”的理解.**
7. **(2010判断正误)CORBA组件中的ORB是一种中间件.**

回答:正确,ORB是一种通信中间件

1. **(2009判断正误)在CORBA组件技术中,IDL编译器针对某接口自动生成的客户桩类(stub类)是对该接口的一种实现.**

回答:正确,stub是对接口的一种伪实现(我猜测是没有实现真正的业务逻辑,但是的确写了一个具体类并实现了那个接口,这实际上就是实现,只不过无法提供有价值的业务能力)接口的完整实现由CORBA服务器完成.

1. **(2009简答)IDL编译器的作用是什么?**

回答:对IDL接口定义进行编译,在客户端生成客户桩,在服务器端生成Skeleton骨架.

1. **(2008简答)结合ORB和BOA,简述你对”伪对象”的理解**
2. **(2008简答)采用何种技术,使得CORBA对象的实现与程序设计语言无关?**

回答:CORBA采用IDL语言描述客户机和服务器程序之间交互操作用到的数据类型和对象接口。因为IDL只描述接口，不描述实现，它是一个纯说明性语言，因此IDL无法编写可执行的语句，也无法解说对象的状态，*IDL*定义把焦点集中在对象接口、其他接口所支持的操作和操作时可能引发的异常上。*IDL*与编程语言无关，这是*CORBA*支持异构系统和独立开发应用程序集成的关键。通过IDL编译器,使用者生成任意一种语言实现的stub和skeleton,这使得CORBA对象可以使用任意语言实现,并与skeleton和stub实现交互.

1. **(2008简答)简述你对客户桩stub和服务器框架skeleton的理解.**

回答:stub和skeleton都是由IDL编译器根据IDL文件生成的,他们是由某种编程语言实现的,具体是什么编程语言取决于所用的IDL编译器和相应设置.其中,stub是程序激发请求时所需要的一套例程,客户侧代码通过stub编码远程命令并发送到服务器,服务器返回的结果由stub解码为调用结果返回给客户侧代码,skeleton是存在与服务器端的,他负责根据stub传送过来的调用命令来调用服务器上下文中的远程对象方法,并把调用结果编码并传送给stub.总体来说,stub和skeleton就是客户侧代码和服务器端远程对象之间的桥梁,客户端程序可以把stub当成IDL接口的一种实现,但实际上真正的实现存在于远端CORBA对象中.

1. **(2007简答)”CORBA对象需要实现IDL描述的接口定义,自动生成的客户桩stub也是对该接口的实现”你如何理解这个论述?**

回答:IDL编译器根据IDL文件描述的接口信息生成由特定编程语言构成的stub和skeleton,客户端通过stub向服务器发出请求.IDL编译器把IDL文件中的接口描述转换成了具体编程语言的实现,所以说stub是对IDL所定义的接口的实现,但是,该实现并没有具体的业务逻辑能力,只是提供了通过ORB向服务端发送请求的操作,业务逻辑必须由程序员在服务端使用某种具体的编程语言来实现,在CORBA组件技术中,表现为服务器应用程序上下文中的Servant,这实际上就是实现IDL描述的接口定义的程序对象,所以说,为了完成业务逻辑,CORBA对象需要时间IDL描述的接口定义.

1. **(2007简答)简述实现CORBA组件的一般步骤**
   1. 建立客户机
      1. 基于功能和界面等设计编写代码中的非CORBA部分
      2. 选择通信样式(同步/延迟同步/异步)和激发类型(桩类型/动态类型/两者都有)
      3. 定义IDL接口
      4. 如果仅支持桩类型激发,生成客户桩并连入客户机;如果仅支持动态激发,将IDL代码装入接口仓库;如果两者都支持,就进行以上两项操作.
      5. 编写激发请求代码并获取激发所需信息,引用,参数.
      6. 编写成功请求的程序代码
      7. 编写异常和错误处理代码
      8. 如果客户机同时也是服务器,参照建立服务器的相关操作
   2. 建立服务器
      1. 决定实现所支持的激活策略
      2. 生成服务器框架
      3. 把框架和服务器应用程序及对象适配器连接起来
      4. 编写服务器初始化代码
      5. 编写代码通知BOA实现已经被激活
      6. 创建消息分发循环
      7. 编写代码,支持框架中的实现,上下文,异常处理
      8. 编写代码冻结实现
      9. 编写关闭服务器的代码
      10. 如果服务器同时也是客户机,参照建立客户机的相关操作