CORBA介绍

潘爱民

北京大学计算机科学技术研究所

http://www.icst.pku.edu.cn/compcourse

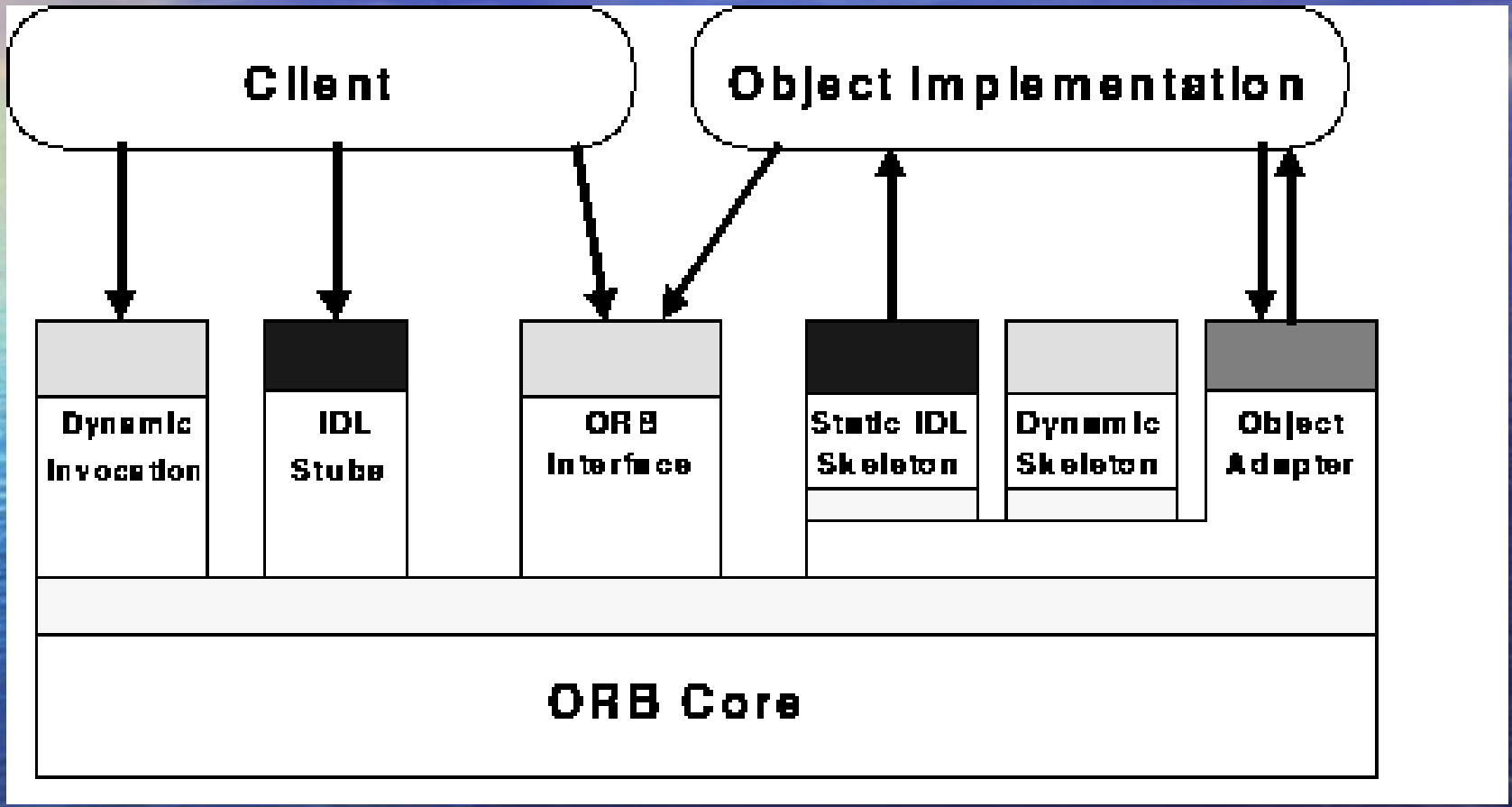
内容

- Overview
- Spec overview
- C++ Mapping
- Develop CORBA application with C++
- CORBA Services
- ORB Products

CORBA Overview

- ORB(Object Request Broker)
- Interface Definition Language (IDL)
- CORBA Communications Model: IIOP
- CORBA Object Model: IOR
- CORBA Clients and Servers
  - client stubs and server skeletons
- CORBAservices and CORBAfacilities

CORBA结构


Client
Object Implementation
Dynamic Invocation
IDL Stubs
ORB Interface
Static IDL Skeleton
Dynamic Skeleton
Object Adapter
ORB Core

CORBA Specification overview

- OMG IDL
- Interface:
  - ORB Interface、DII/DSI、Interface Repository
- POA
- Interoperability
  - GIOP、IIOP、IOR
- Mapping
  - COM、Automation
  - C mapping、C++ mapping、SmallTalk
- Service Specification(单独的文档)
  - naming service、event service、transaction service、security service、trade service、……

OMG IDL

- 类C风格的语言
- 定义接口: interface
- 基本数据类型
  - void、boolean、char、wchar、octet、(unsigned) short、(unsigned) long、(unsigned) long long、float、double、long double、string、any
- 常数、常量表达式
- 构造数据类型
  - typedef、enum、struct、union、数组、sequence

OMG IDL(续)

- Interface，例如：

```
interface Account {
    void deposit( in unsigned long amount );
    void withdraw( in unsigned long amount );
    long balance();
};
```

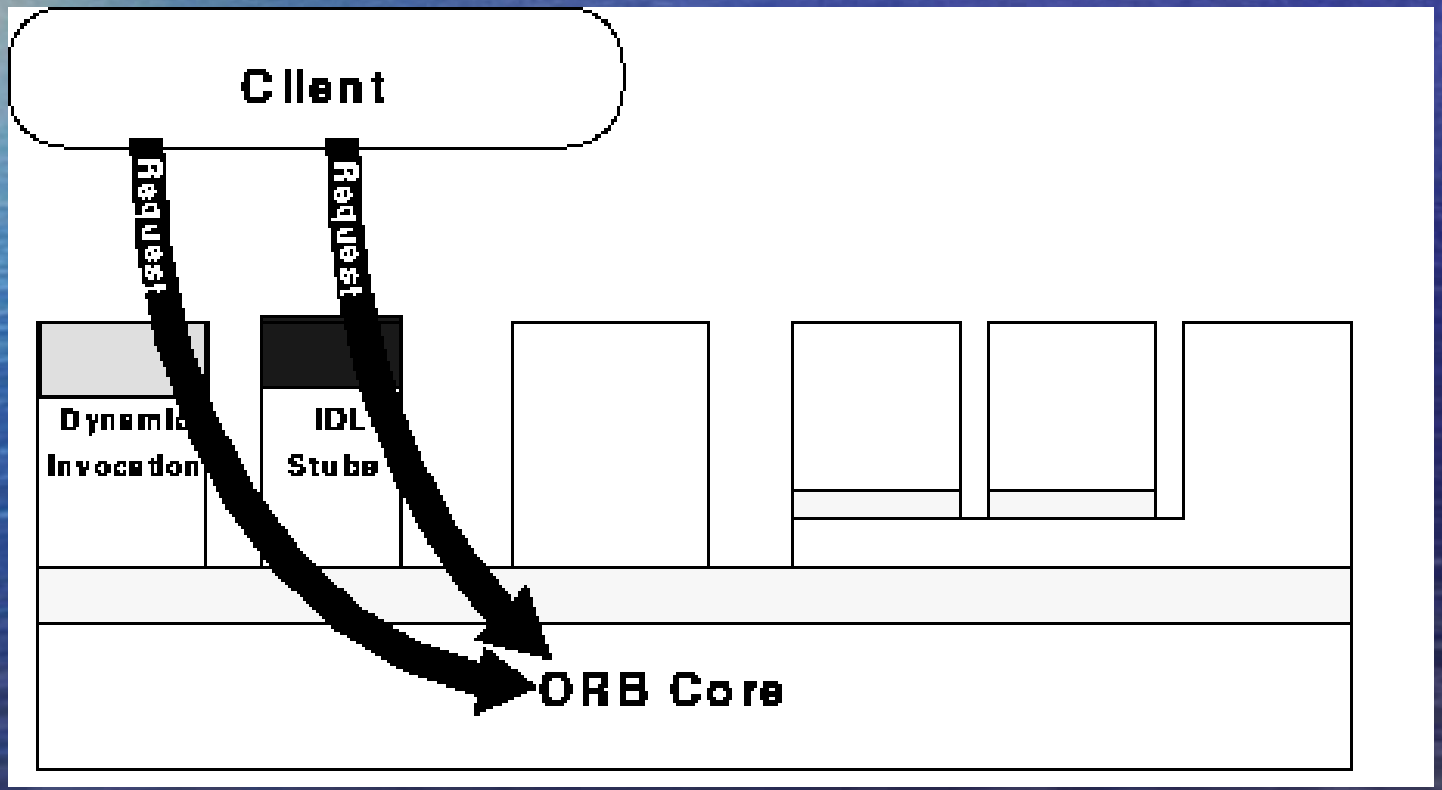
- 接口方法中参数的方向属性
- 异常：用户自定义异常和系统异常
- oneway操作
- module、include
- 允许多重继承，与实现(implementation)无关

C++ mapping

- 基本数据类型与C++的对应关系
  - 特殊标识符的处理
  - module ——〉 namespace
  - IDL类型——〉CORBA::中的数据类型
  - 字符串处理不用new、delete，而用专门的函数
- 枚举类型
- 复合类型
  - 除了类型本身外，另生成一个_var类型(类似auto_ptr)
- 接口类型
  - _ptr类型、_var类型

客户端C++ mapping

- 客户通过stub调用对象的方法
  - 客户如何得到对象引用
  - 客户如何调用对象方法



客户端C++ mapping(续一)

- IDL编译器为每个接口生成两个对象引用类型，例如对于Account接口，生成：
  - Account、Account_ptr、Account_var

```
class Account;
typedef Account *Account_ptr;
typedef ObjVar<Account> Account_var;

class Account :
 virtual public CORBA::Object
{
public:
    ……
   static Account_ptr _narrow(
    CORBA::Object_ptr obj );
  static Account_ptr _narrow(
    CORBA::AbstractBase_ptr obj );
    void deposit( CORBA::ULong amount );
    void withdraw( CORBA::ULong amount
    CORBA::Long balance();

  protected:
    Account() {};
  private:
    Account( const Account& );
    void operator=( const Account& );
};
```

客户端C++ mapping(续二)

- Account中的方法由IDL编译器提供实现，如

```
void Account::deposit( CORBA::ULong amount )
{
    CORBA::StaticAny _amount( CORBA::_stc_ulong, &amount );
    CORBA::StaticRequest __req( this, "deposit" );
    __req.add_in_arg( &_amount );
    __req.invoke();
}
CORBA::Long Account::balance()
{
    CORBA::Long _res;
    CORBA::StaticAny __res( CORBA::_stc_long, &_res );
    CORBA::StaticRequest __req( this, "balance" );
    __req.set_result( &__res );
    __req.invoke();
  return _res;
}
```

客户端C++ mapping(续三)

- 客户通过ORB接口获得初始的对象引用
  - 首先初始化ORB并获得ORB接口
  - 然后利用字符串形式的引用获得内部的对象引用

```
int main( int argc, char *argv[] )
{
  // ORB initialization
  CORBA::ORB_var orb = CORBA::ORB_init( argc, argv, "local-orb" );

  CORBA::Object_var obj = orb->string_to_object( argv[1] );
  Account_var client = Account::_narrow( obj );

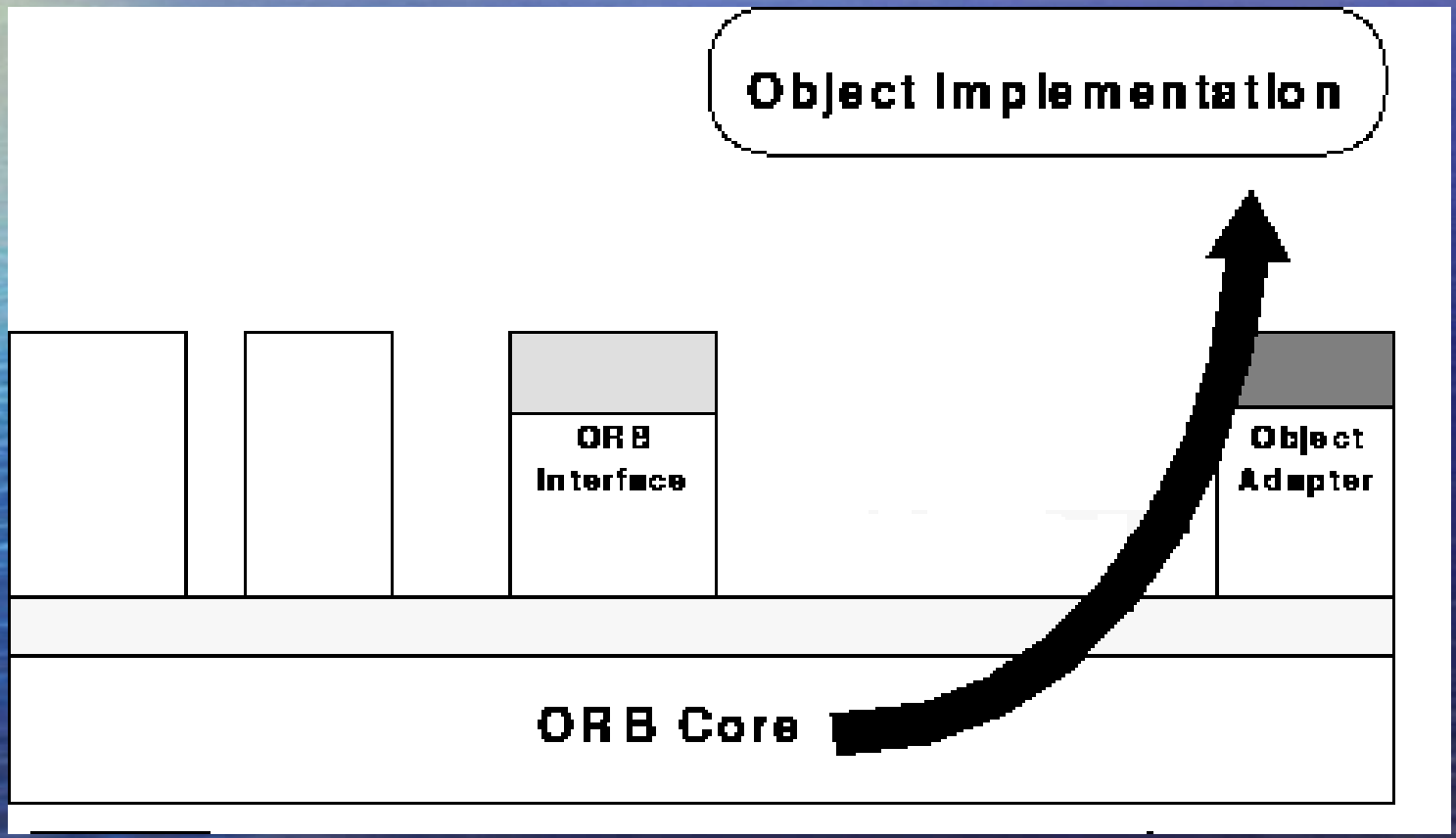
  client->deposit( 700 );
  client->withdraw( 250 );
  cout << "Balance is " << client->balance() << endl;
  return 0;
}
```

客户端C++ mapping(续四)

- Account::_narrow
  - 安全地向下类型转换
- CORBA::Object
  - C++中，所有CORBA对象(如Account)的基类
- Account::_duplicate
  - 复制对象引用
- CORBA::is_nil
  - 测试空引用
- 异常处理
- 参数处理

服务器端C++ mapping

- 对象与OA进行通信



服务器端C++ mapping(续一)

- IDL编译器生成对象的框架类，接口方法为纯虚函数，所以必须派生一个新的类(servant)，并实现接口的方法。

```
#include “account.h”
class Account_impl : virtual public Account_skel
{
  private:
     CORBA::Long _current_balance;
  public:
      Account_impl() {   _current_balance = 0;   };

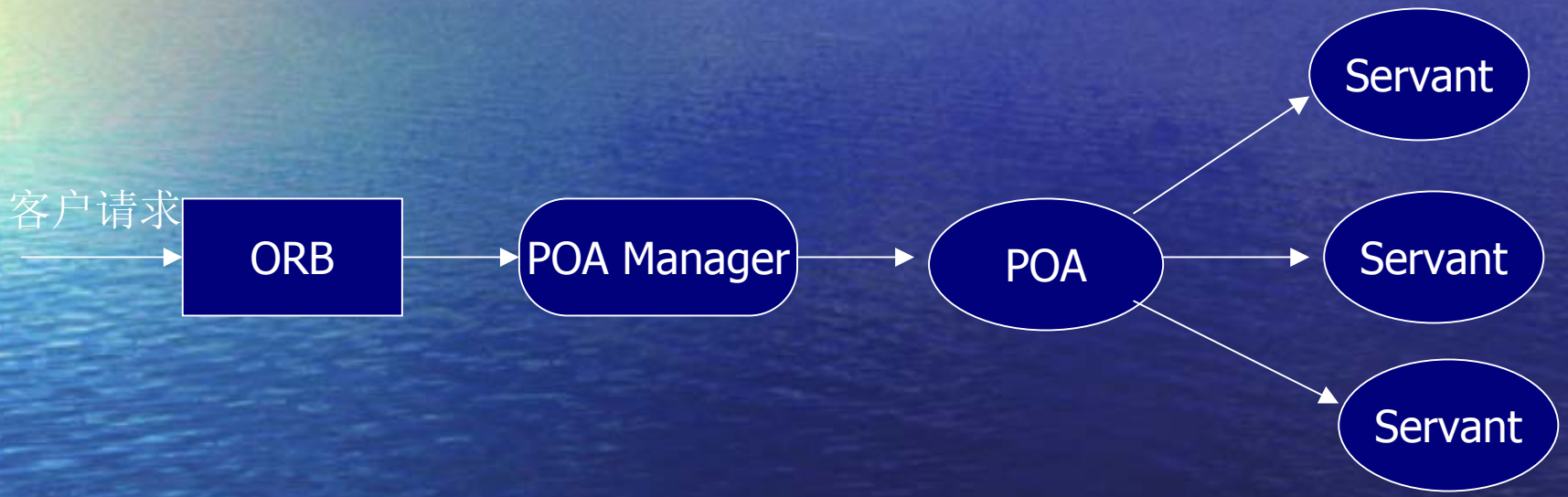
      void deposit( CORBA::ULong amount )
               {   _current_balance += amount;  };

      void withdraw( CORBA::ULong amount )
               {  _current_balance -= amount;  };

     CORBA::Long balance()
               {    return _current_balance;  };
};
```

服务器端C++ mapping(续二)

- Servant和CORBA对象



服务器端C++ mapping(续三)

```
int main (int argc, char *argv[])
{
 CORBA::ORB_var orb = CORBA::ORB_init (argc, argv, ”local-orb”);

 CORBA::Object_var poaobj = orb->resolve_initial_references ("RootPOA");
  PortableServer::POA_var poa = PortableServer::POA::_narrow (poaobj);
  PortableServer::POAManager_var mgr = poa->the_POAManager();

  Account_impl * account = new Account_impl;

  PortableServer::ObjectId_var oid = poa->activate_object (account);

  …… // Write reference to file

  mgr->activate ();
  orb->run();

  poa->destroy (TRUE, TRUE);
  delete micocash;
  return 0;
}
```

POA: Portable Object Adapter

- POA负责创建对象引用、激活对象以及分发请求
- POA负责从CORBA对象到servant之间的对应
  - CORBA对象与servant对象的生命周期相互分开
- POA对于CORBA服务器在性能、资源利用、可伸缩性等方面具有重要的意义
- POA策略，控制：
  - 对象生命周期：永久对象、暂态对象
  - 对象标识符：在POA范围内唯一
  - POA Active Object Map：从ObjectID到servant的映射
  - 请求处理策略：servant manager
  - 线程模型
  - ……

POA: Portable Object Adapter(续)

- POA的编程
  - 所有的工作都从RootPOA开始
  - 所有的POA构成一个树状层次结构
  - 利用RootPOA的create_POA创建新的POA
  - 在创建POA时指定策略，CORBA::PolicyList
- POA提供的功能
  - 创建对象
  - 注册servant，激活对象
  - 提供ObjectID、ObjectReference之间的转换
- servant manager编程
- POA manager：控制来自客户的请求
  - 多个POA可以共享同一个POA manager
- ORB事件处理

服务器端C++ mapping(续四)

- 利用idl生成的框架类编写servant类
- 编写main函数
  - 从ORB初始化开始
  - 考虑POA的策略，简单的程序可以使用默认的RootPOA
  - 创建servant并注册
  - 处理对象引用
  - 进入主循环
    - 阻塞方式：orb->run()
    - 非阻塞方式：
      orb->work_pending()
      orb->perform_work()

用C++开发CORBA应用

- 基本的C++知识，包括链表结构、智能指针的知识
- 针对当前使用的ORB产品，熟悉对于基本数据类型的支持，以及这些类型与当前平台上各数据类型的关系
- 理解CORBA的client/server应用模型
- 包含头文件
- 链接库文件
- 掌握IDL编译器
- 掌握ORB接口
- 如何传递对象引用

ORB接口

- 初始：CORBA::ORB_Init，通过命令行传递参数
- 对象引用到字符串的转换操作
  - 对象引用：CORBA::Object
- 获得初始接口
  - resolve_initial_reference

DII：Dynamic Invocation Interface

- 客户程序可以不通过IDL编译器生成的stub而调用对象的方法，步骤：
  - 客户首先得到对象引用：CORBA::Object_ptr
  - 创建一个请求对象CORBA::Request_ptr
  - 把调用参数放到链表中：CORBA::NVList_ptr，指明类型和数值，也指明返回值
  - 调用CORBA::Request_ptr->Invoke方法
- 服务器并不知道客户程序是通过stub还是DII调用其方法的
- 客户程序在编译时刻可以没有idl类型知识
- 比较：COM Automation中IDispatch接口的用法

用C++开发CORBA应用：客户

- 用IDL生成stub代码以及类型定义头文件
- 在客户的main函数中，初始化orb，获得orb对象
- 获得对象引用，并调用_narrow向下转换
- 调用对象的方法
- 如果使用DII，则需要用到请求对象，并处理参数
- 如果不用_var类型的话，释放orb和对象引用

DSI：Dynamic Skeleton Interface

- 服务器程序可以动态实现对象，而不利用IDL编译器生成的skeleton代码
- 服务器实现一个对象，如下所示

```
class GenericServant : virtual public PortableServer::DynamicImplementatio
{
        virtual void invoke (CORBA::ServerRequest_ptr req);
        virtual CORBA::RepositoryId _primary_interface (
                const PortableServer::ObjectId &,
                PortableServer::POA_ptr);
};
```

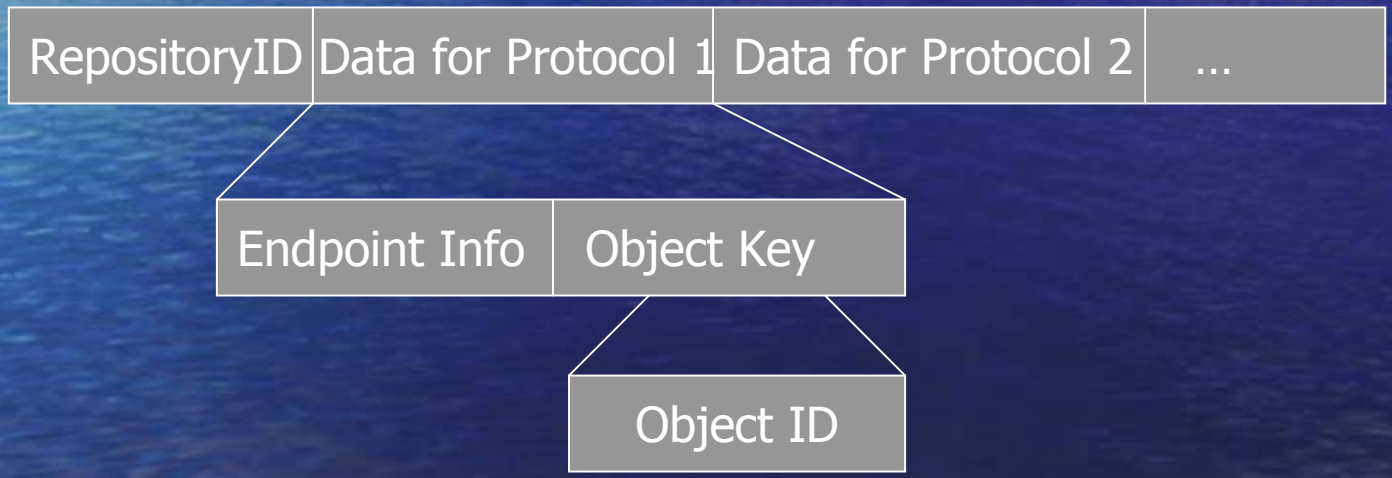
- main函数的处理不变
- 重点在于invoke函数，利用CORBA::ServerRequest对象获得所有的类型信息
- 对于客户透明
- 比较：COM Automation中IDispatch接口的实现

用C++开发CORBA应用：服务器

- 用IDL生成skeleton代码以及类型定义头文件
- 在客户的main函数中，初始化orb，获得orb对象
- 根据服务器应用的需要，处理POA
- 创建servant对象
- 如果使用DSI，那么invoke方法是servant对象的关键
- 如何把对象引用传递出去？字符串？名字服务？
- 调用orb->run()或者orb->perform_work()

CORBA IIOP和IOR

- General Inter-ORB Protocol：CORBA定义了GIOP作为其互用性框架结构，包括数据传输、数据表示、消息格式
- Internet Inter-ORB Protocol：IIOP是GIOP的具体实现，建立在TCP/IP基础上
- IOR：Interoperable object reference



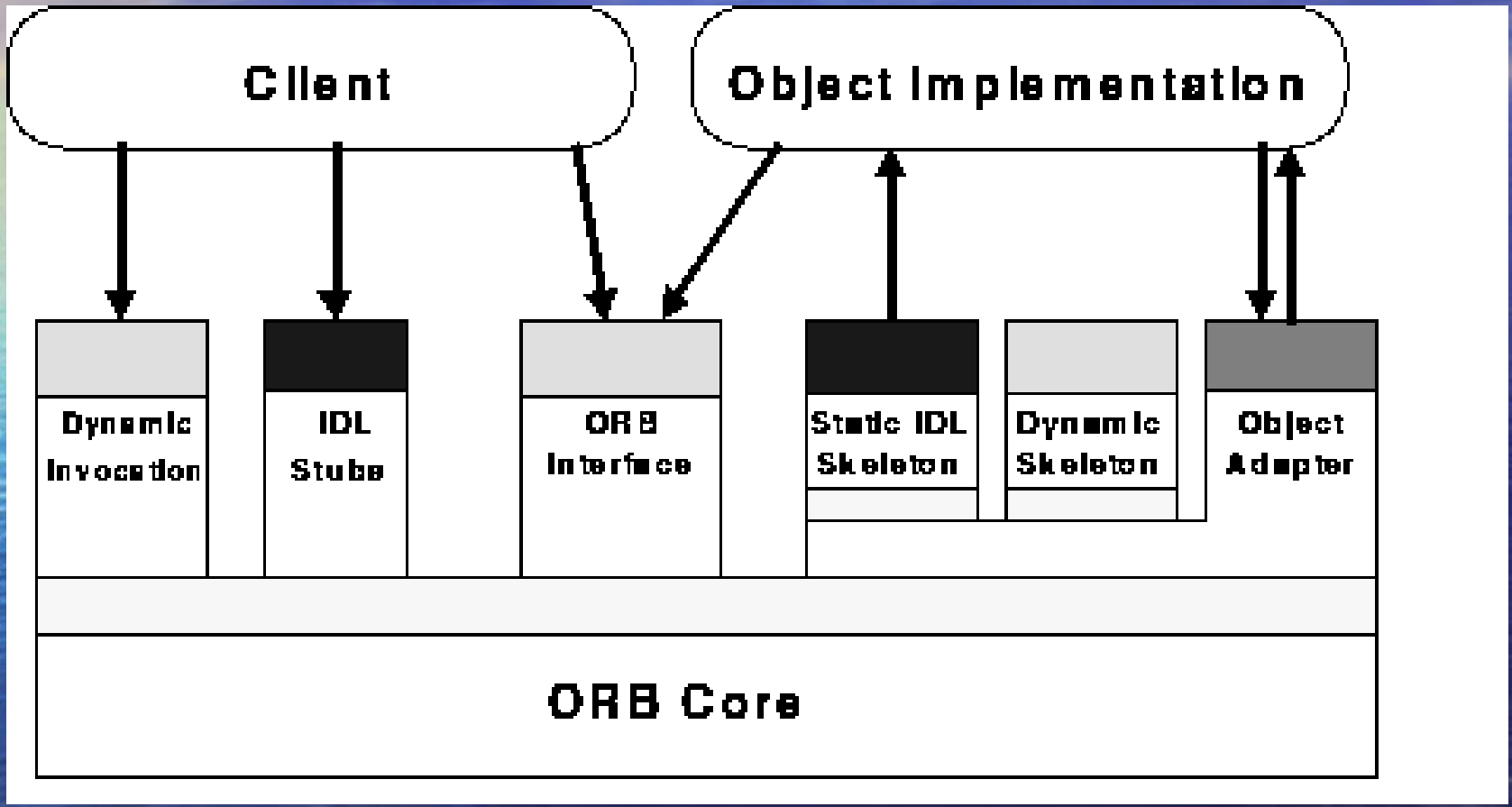
IOR例子

Repo Id: IDL:GenericServer:1.0

IIOP Profile
  Version: 1.0
  Address: inet:162.105.178.100:12122
  Location: corbaloc::162.105.73.196:12122//25607/991958392/%5f0
    Key: 2f 32 35 36 30 37 2f 39 39 31 39 35 38 33 39 32 /25607/991958392
      2f 5f 30 /_0

Multiple Components Profile
 Components: Native Codesets:
      normal: ISO 8859-1:1987; Latin Alphabet No. 1
       wide: ISO/IEC 10646-1:1993; UTF-16, UCS Transformation Format 16-bit form
    Key: 00 .

CORBA结构


Client
Object Implementation
Dynamic Invocation
IDL Stubs
ORB Interface
Static IDL Skeleton
Dynamic Skeleton
Object Adapter
ORB Core

CORBA services：naming service

- 建立起“名称-对象引用”的映射关系
- 在内部以层次结构的形式组织名字映射，类似于文件系统的结构。
- 所有的类型和接口定义位于CosNaming域中
- NamingContext接口负责所有有关命名服务相关的功能，例如：
  - 创建新的context、删除context
  - bind操作、unbind操作
  - resolve操作
- 怎样获得初始namingcontext？
  - 通过orb->resolve_initial_reference
- ORB厂商应该提供命名服务工具

naming service示例

```
CORBA::Object_var nsobj = orb->resolve_initial_references ("NameService");
CosNaming::NamingContext_var nc =
        CosNaming::NamingContext::_narrow (nsobj);
CosNaming::Name name;
name.length (1);
name[0].id = CORBA::string_dup ("myAccount");
name[0].kind = CORBA::string_dup ("");

// 服务器方
Account_ptr acc = new Account_impl ();
nc->bind (name, acc);

// 客户方
CORBA::Object_var obj;
obj = nc->resolve (name);
Account_var client = Account::_narrow( obj );
```

CORBA services：其他服务

- Event service
  - 提供松耦合事件模型的机制
- Trade Service
  - 提供了更加灵活的对象查找服务
- Time Service
  - 提供了与时间有关的服务，统一了时间的表达方式
- Concurrency service
- Security Service
- Transaction Service
- ……

ORB产品

- 考察ORB：支持平台、支持语言、性能
- 商业ORB
  - Orbix，IONA公司，完全支持CORBA 2.3规范
  - Visibroker，Inprise公司，4.0版完全支持2.3规范
    Netscape communicator浏览器嵌入Visibroker
- 自由ORB
  - ORBit，遵循CORBA 2.2规范，支持C语言，性能较高
  - mico，GNU，OpenSource
  - TAO，美国华盛顿大学分布式对象计算研究小组
  - omniORB，AT&T剑桥实验室

CORBA与COM的比较

- 标准的层次不同
  - CORBA与Java结合的优势
- CORBA跨平台优势
- COM在Windows平台上的优势
- COM的效率有优势
- 跨语言的策略不同
- COM组件丰富、CORBA开发简单
  - CORBA不适合开发UI组件
- CORBA与COM互补策略