Corba Job Interview Questions And Answers

Interview Questions Answers

https://interviewquestionsanswers.org/

About Interview Questions Answers

Interview Questions Answers . ORG is an interview preparation guide of thousands of Job Interview Questions And Answers, Job Interviews are always stressful even for job seekers who have gone on countless interviews. The best way to reduce the stress is to be prepared for your job interview. Take the time to review the standard interview questions you will most likely be asked. These interview questions and answers on Corba will help you strengthen your technical skills, prepare for the interviews and quickly revise the concepts.

If you find any **question or answer** is incorrect or incomplete then you can **submit your question or answer** directly with out any registration or login at our website. You just need to visit Corba Interview Questions And Answers to add your answer click on the *Submit Your Answer* links on the website; with each question to post your answer, if you want to ask any question then you will have a link *Submit Your Question*; that's will add your question in Corba category. To ensure quality, each submission is checked by our team, before it becomes live. This Corba Interview preparation PDF was generated at **Wednesday 29th November, 2023**

You can follow us on FaceBook for latest Jobs, Updates and other interviews material.
www.facebook.com/InterviewQuestionsAnswers.Org

Follow us on Twitter for latest Jobs and interview preparation guides.
https://twitter.com/InterviewQA

If you need any further assistance or have queries regarding this document or its material or any of other inquiry, please do not hesitate to contact us.

Best Of Luck.

Interview Questions Answers.ORG Team
https://InterviewQuestionsAnswers.ORG/
Support@InterviewQuestionsAnswers.ORG

Corba Interview Questions And Answers Guide.

Question - 1:

Explain the reason to implement a corba application with multi-threading?

Ans:

CORBA server applications may be multi-threaded for serveral reasons.
A particular CORBA object may support an operation whose implementation performs some blocking routine. This may be a disk read or database query. Let us assume that the server application processes all CORBA events within a single main thread. This means that the server will be unable to respond to incoming connection requests or invocation requests while the blocking operation is in progress. Multi-threading can be used to avoid these sorts of situations. The server can be more accessible if multiple threads are allowed to process (an block during) incoming CORBA events.
A single multi-threaded server process supporting many (>25) clients is much more efficient that many (>25) single-threaded server processes each supporting its own client. Running a single application with multiple threads requires less machine resources than running multiple applications. This advantage can be seen even if the operation invocations are of short duration and non-blocking.

View All Answers

Question - 2:

Explain Does Corba supports asynchronous communication?

Ans:

Kind of. At the lowest level CORBA supports two modes of communication:
A synchronous request/response which allows an application to make a request to some CORBA object and then wait for a response.
A deferred synchronous request/response which allows an application to make a request to some CORBA object. An empty result will be returned immediately to the application. It can then perform other operations and later poll the ORB to see if the result has been made available.
At the lowest level, the CORBA deferred synchronous communication does allow a certain degree of asynchronous communication. Polling for responses represents only one form of asynchronous communication. Other more sophisticated asynchronous communication can only be achieved by developing an architecture on top of the lowest levels of CORBA.

View All Answers

Question - 3:

Explain Can Corba application have call back?

Ans:

Yes. The words client and server are really only applicable in the context of a remote call. In other words, the client process can also receive calls on CORBA objects that it implements and hands out the references to.

View All Answers

Question - 4:

Explain some reason to avoid the development of multi-threaded Corba application?

Ans:

Building multi-threaded applications requires an additional efforts in the area of design, development and testing. Issues like concurrency and synchronization become more critical. Difficult to find software bugs are unfortunately easy to introduce. A specific set of application requirements can often be met without resorting to the use of threaded clients or servers. This is not true with all applications. Some do require multi-threading to achieve their desired level of concurrency, performance or scalability.

View All Answers

Question - 5:

Explain What is CORBA good for?

Ans:

CORBA is useful in many situations. Because of the easy way that CORBA integrates machines from so many vendors, with sizes ranging from mainframes through minis and desktops to hand-helds and embedded systems, it is the middleware of choice for large (and even not-so-large) enterprises. One of its most important, as well most frequent, uses is in servers that must handle large number of clients, at high hit rates, with high reliability. CORBA works behind the scenes in the computer rooms of many of the world's largest websites; ones that you probably use every day. Specializations for scalability and fault-tolerance support these systems. But it's not used just for large applications; specialized versions of CORBA run real-time systems, and small embedded systems.

View All Answers

Question - 6:

Explain Can Corba allow servers to cause client side events or notifications?

Ans:

CORBA communication is inherently asymmetric. Request messages originate from clients and responses originate from servers. The important thing to realize is that a CORBA server is a CORBA object and a CORBA client is a CORBA stub. A client application might use object references to request remote service, but the client application might also implement CORBA objects and be capable of servicing incoming requests. Along the same lines, a server process that implements CORBA objects might have several object references that it uses to make requests to other CORBA objects. Those CORBA objects might reside in client applications. By implementing a CORBA object within an client application, any process that obtains its object reference can ?notify? it by performing an operation on the client-located object.

View All Answers

Question - 7:

Give us high-level technical overview of Corba?

Ans:

CORBA applications are composed of objects, individual units of running software that combine functionality and data, and that frequently (but not always) represent something in the real world. Typically, there are many instances of an object of a single type - for example, an e-commerce website would have many shopping cart object instances, all identical in functionality but differing in that each is assigned to a different customer, and contains data representing the merchandise that its particular customer has selected. For other types, there may be only one instance. When a legacy application, such as an accounting system, is wrapped in code with CORBA interfaces and opened up to clients on the network, there is usually only one instance.
For each object type, such as the shopping cart that we just mentioned, you define an interface in OMG IDL. The interface is the syntax part of the contract that the server object offers to the clients that invoke it. Any client that wants to invoke an operation on the object must use this IDL interface to specify the operation it wants to perform, and to marshal the arguments that it sends. When the invocation reaches the target object, the same interface definition is used there to unmarshal the arguments so that the object can perform the requested operation with them. The interface definition is then used to marshal the results for their trip back, and to unmarshal them when they reach their destination.
The IDL interface definition is independent of programming language, but maps to all of the popular programming languages via OMG standards: OMG has standardized mappings from IDL to C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python, and IDLscript.
For more on OMG IDL, click here.
This separation of interface from implementation, enabled by OMG IDL, is the essence of CORBA - how it enables interoperability, with all of the transparencies we've claimed. The interface to each object is defined very strictly. In contrast, the implementation of an object - its running code, and its data - is hidden from the rest of the system (that is, encapsulated) behind a boundary that the client may not cross. Clients access objects only through their advertised interface, invoking only those operations that that the object exposes through its IDL interface, with only those parameters (input and output) that are included in the invocation.
Request flow
Figure 1 shows how everything fits together, at least within a single process: You compile your IDL into client stubs and object skeletons, and write your object (shown on the right) and a client for it (on the left). Stubs and skeletons serve as proxies for clients and servers, respectively. Because IDL defines interfaces so strictly, the stub on the client side has no trouble meshing perfectly with the skeleton on the server side, even if the two are compiled into different programming languages, or even running on different ORBs from different vendors.
In CORBA, every object instance has its own unique object reference, an identifying electronic token. Clients use the object references to direct their invocations, identifying to the ORB the exact instance they want to invoke (Ensuring, for example, that the books you select go into your own shopping cart, and not into your neighbor's.) The client acts as if it's invoking an operation on the object instance, but it's actually invoking on the IDL stub which acts as a proxy. Passing through the stub on the client side, the invocation continues through the ORB (Object Request Broker), and the skeleton on the implementation side, to get to the object where it is executed.

View All Answers

Question - 8:

What is CORBA? What does it do?

Ans:

CORBA is the acronym for Common Object Request Broker Architecture, OMG's open, vendor-independent architecture and infrastructure that computer applications use to work together over networks. Using the standard protocol IIOP, a CORBA-based program from any vendor, on almost any computer, operating system, programming language, and network, can interoperate with a CORBA-based program from the same or another vendor, on almost any other computer, operating system, programming language, and network.

View All Answers

Question - 9:

Tell me Can Corba application be multi-threaded?

Ans:

The CORBA specification does not currently address multi-threaded architectures. Provided that the CORBA product is thread safe, threaded CORBA applications can be developed. CORBA clients and servers can both be multi-threaded. Daemon processes provided with CORBA products may be implemented as multi-threaded servers by the CORBA vendor. Different multi-threaded models or multi-threaded architectures may be supported by a particular CORBA product. A particular ORB may provide frameworks to simplify the development of multi-threaded CORBA applications.

View All Answers

Question - 10:

Explain Do different Corba implementations perform at significantly different levels?

Ans:

Different CORBA implementations can vary significantly in performance. Good implementations should be fairly similar since network performance defines the maximum achievable performance characteristics. Network latency does represent the significant portion of distributed invocation latency.

View All Answers

Question - 11:

Explain What is the reason to implement Corba in client application application?

Ans:

Client-side CORBA applications might require multi-threading to allow it to perform other tasks while it is waiting for a synchronous remote invocation to return. It might desire this functionality for several different reasons.
A client application might wish to leverage the static request/response style of invocation but achieve some degree of asynchronous communication. Perhaps the client wishes to perform several synchronous invocations within their own application threads. This would allow a client to obtain results from several remote servers more quickly. There are several reasons the use of multi-threading might be preferred over the use of DII. DII might be complicate application source code. Application polling associated with the deferred synchronous invocation might result in a performance bottleneck.
A client-side CORBA application might need to respond to events such as incoming invocations, connect requests, or GUI events (mouse clicks, etc.) CORBA products that support only blocking style remote invocations will be unable to process any of these events. This would mean that a client-side application would be unable to respond to GUI events for the duration of any remote CORBA invocations. This is not an issue for short duration invocations but becomes a problem for longer invocations or in failure or time-out situations. Performing remote invocations within dedicated threads can avoid this issue.

View All Answers

Question - 12:

Explain Are there different threading models that can be used within Corba servers?

Ans:

There are several different common architectures that can be used within multi-threaded CORBA servers. A server process needs the ability to process CORBA messages. These messages are processed by one or more threads, as determined by the application architecture. The CORBA specification does not specifically address threading capabilities within CORBA compliant ORBs. An ORB vendor is free to support only single-threaded application or to support multi-threaded applications. If the ORB does support the development of multi-threaded applications, the ORB might only support a subset of the threading models listed below. Significant threading code might still need to be developed to achieve one of the models. For example, the ORB vendor might support a set of application hooks (i.e., interceptors or filters) and allow you to implement threading code with the native OS thread API. On the other hand, the ORB product might provide a built-in feature so no custom thread development needs to be done. The CORBA specification does not address this issue. When you consider different threading models, it is important to consider what kind of concurrency is desired. While it may be advantageous that two or more threads can be concurrent, it may also be disadvantageous. Also, the resources consumed by idle or active threads, and also the resources consumed for thread creation and deletion, need to be considered. Thread-Per-Request: With this architecture, the CORBA server ensures that each incoming message is processed in its own thread. This means that multiple requests will be processed concurrently. There are concurrency issues. If two or more requests (threads) are using the same object, then some form of concurrency control (locking) is needed. Also, if two or more requests (threads) are from the same client, then perhaps the requests should be serialized instead of allowed to execute concurrently. Thread-Per-Client: With this architecture, the CORBA server ensures that each incoming message from a distinct client is processed in its own thread. This is similar to Thread-Per-Request except multiple requests from the same client are serialized. Requests from distinct clients are concurrent. The way that one client is distinguished from another is an interesting problem. Typically, this is done by looking a the network connection and determining that the clients are the same or different. The server needs the ability to monitor client connections and the inception and termination of this connections (typically at a network level, not an application level). Thread-Per-Server-Object: With this architecture, the CORBA server ensures that each object in the server gets it own thread of execution. This means that multiple requests will be processed concurrently provided they are using different objects. Multiple requests using the same object are serialized. There are concurrency issues, and some locking strategy is needed. Also, deadlock is very possible. It may be that threading or locking at each object is too fine a grain, and a more appropriate choice is putting the thread/lock boundary around a group of coordinating objects. For each of the above threading architectures, the required server threads can be either created on demand or recycled through a thread pool. The advantage of creating threads on demand is that an arbitrary number of threads can be supported. A thread is created, used, and then reaped. The Thread-Per-Request model would create/reap a thread for each request; the Thread-Per-Client model would create/reap a thread for each client connection; the Thread-Per-Server-Object model would create/reap a thread for each CORBA object instantiated in the server. Thread creation and reaping has some cost, which may be large or small depending on the operating system thread support. A thread pool is an alternative to creating threads on demand. In this approach, a fixed number of threads are created and cycled in turn to meet the demand for threads. If the demand for threads exceeds the pool size, then further requests for threads are blocked until one of the existing threads is recycled. This approach has the advantage of capping the server resources.

View All Answers

Question - 13:

Explain Does Corba define high level application architectures?

Ans:

No, it is infrastructure. Which is good because the history of high-level? one size fits all? architectures hasn't been very good, has it?
CORBA provides low level request/response communication. It also provides general services that are implemented on top of request/response communication. The actual architecture used within a given application is not defined by CORBA. CORBA leaves these decisions up the application architect.

View All Answers

Question - 14:

Explain Does Corba support distributed reference counting architectures?

Ans:

CORBA does not directly support distributed reference counting. This was a conscious decision on the part of its designers. While CORBA does not directly support reference counting, it is possible to build reference counting into a particular distributed object architecture. This can be done through an explicit session management facility which can be exposed through factories or other remote interfaces. While it is possible to design reference counting into an application, it is the burden of the application designer/developer to ensure that such an approach is implemented correctly.

View All Answers

Question - 15:

Explain Can Corba application be tuned for better performance?

Ans:

There are a number of ways to tune CORBA applications for better performance. Remember that distribution should only be used if a reason to do so exists. Distribution does not make sense for the sake of distribution. If distribution does not serve a purpose then it should be avoided. Avoiding excessive distribution can result in better performance. Care should be taken when introducing distribution into an applications object model. IDL interfaces can be tuned to minimize network latency. Invoking remote operations requires transmitting data across the network. Network performance is typically optimized by ensuring adequate bandwidth. Once the required bandwidth is achieved raw network performance cannot be increased. One key to tuning an IDL interface is to reduce the number of network transfers that need to occur. Calling an operation that returns 100 bytes might take 5 milliseconds. Calling an operation that returns 200 bytes of data might take

around 6 milliseconds. Calling 2 operations that return 100 bytes might take a total of 10 milliseconds. One key to tuning IDL operations is to avoid implementing several get operations and to combine them into a single get operation which returns the appropriate combination of data. Caching results of remote operations can avoid network overhead associated with calling the same remote methods more than once. Many applications can perform remote operations upon startup rather than during normal usage. Users are often more willing to wait at startup time rather than during application usage. Many performance problems are associated with serialization and blocking conditions. For example, Let us assume that clients will be making remote operations to a single server. A single client?s request causes the server to block for a extended period of time, the entire client community might have to wait. Make sure that multiple distributed operations are not becoming serialized within a single server process. Utilize multiple server processes or threaded servers instead.

View All Answers

Question - 16:

Explain Are there important forms of asynchronous communication that are not supported directly by Corba?

Ans:

Yes, but we can fake it pretty easily.
While CORBA does support a deferred synchronous request/response, it does not directly support distributed requests with a callback driven response. A callback driven response allows an application to perform an operation on a distributed object, associate a callback with the response, continue with other processing. When the server responds, the associated callback is automatically executed within the original caller?s application.

View All Answers

Question - 17:

Explain How does Corba support interoperability?

Ans:

CORBAs goal is to address interoperability at various levels. There is a history to this.
In the early versions of CORBA, interoperability between platforms and programming languages was addressed. This included the standardization of IDL and the mapping of IDL to a programming language. While a client and server developed with the same vendors ORB could talk to one another, a client and server developed with different vendors? ORBs were not likely to interoperate.
CORBA 2.0 introduced interoperability between different ORB vendors. This resulted from the introduction of a standard wire protocol called General Inter-ORB Protocol (GIOP), and the incarnation for GIOP for the internet, known as Internet Inter-ORB Protocol (IIOP). So CORBA 2.0 compliant ORBs will interoperate. This means a client using ORB vendor A can talk to a server using ORB vendor B.
Interoperability is actually a broader issue than just have ORB vendor A talking to ORB vendor B. Fuller interoperability means that various services interoperate. For example, while a CORBA object can talk to a DCOM object via a protocol bridge, can the CORBA Transaction Service talk to the Microsoft Transaction Service to have a seamless transaction between systems? This broader interoperability at the service level is being addressed now.

View All Answers

Networking Most Popular & Related Interview Guides

1 : **CCNA Interview Questions and Answers.**

2 : MCSE Interview Questions and Answers.

3 : MCSA Interview Questions and Answers.

4 : CCNP Interview Questions and Answers.

5 : Network Administrator Interview Questions and Answers.

6 : Active Directory Interview Questions and Answers.

7 : CCNA Security Interview Questions and Answers.

8 : Basic Networking Interview Questions and Answers.

9 : System Administration Interview Questions and Answers.

10 : VPN Interview Questions and Answers.

Follow us on FaceBook
www.facebook.com/InterviewQuestionsAnswers.Org

Follow us on Twitter
https://twitter.com/InterviewQA

For any inquiry please do not hesitate to contact us.

Interview Questions Answers.ORG Team
https://InterviewQuestionsAnswers.ORG/
support@InterviewQuestionsAnswers.ORG