**Difference between a Queue and a Topic ?**

Topics

In JMS Topic implements publish and subscribe semantics. When you publish a message it goes to all the subscribers who are

interested - so zero to many subscribers will receive a copy of the message. Only subscribers who had an active subscription

at the time the broker receives the message will get a copy of the message.

Queues

A JMS Queue implements load balancer semantics. A single message will be received by exactly one consumer. If there are no

consumers available at the time the message is sent it will be kept until a consumer is available that can process the message.

If a consumer receives a message and does not acknowledge it before closing then the message will be re-delivered to another

consumer. A queue can have many consumers with messages load balanced across the available consumers.

Topics are for the publisher-subscriber model, while queues are for point-to-point model.

**Difference between java serialization and JSON string generation ?**

Java serialization mandates on the class to implement the marker Serializable interface, where as json serialization doesn't put any such constraints.

Java serialization serializes to non-human readable format, where as json serialization serializes into human readable string format.

Java serialization is much slower as compared to Json serialization.

**Difference between JAX-WS and JAX-RS?**

JAX-WS

Java API for the XML-Based Web Services - a standard way to develop a Web-Services in SOAP notation (Simple Object Access Protocol).

Calling of the Web Services is performed via remote procedure calls. For the exchange of information between the client and the Web Service is used SOAP protocol. Message exchange between the client and the server performed through XML- based SOAP messages.

Clients of the JAX-WS Web-Service need a WSDL file to generate executable code that the clients can use to call Web-Service.

JAX-RS

Java API for RESTful Web Services. RESTful Web Services are represented as resources and can be identified by Uniform Resource Identifiers (URI).

Remote procedure call in this case is represented a HTTP- request and the necessary data is passed as parameters of the query.

Web Services RESTful - more flexible, can use several different MIME- types. Typically used for XML data exchange or JSON (JavaScript Object Notation) data exchange.

**Scanner vs BufferReader -**

Though both are meant for standard input but Scanner is used for parsing tokens from the contents of the stream while BufferedReader just reads the stream and does not do any special parsing.

BufferedReader is synchronized and Scanner is not, so its up to you to decide.

**Comparable vs Comparator -**

Comparable lets a class implement its own comparison:

it's in the same class (it is often an advantage)

there can be only one implementation (so you can't use that if you want two different cases)

By comparison, Comparator is an external comparison:

it is typically in a unique instance (either in the same class or in another place)

you name each implementation with the way you want to sort things

you can provide comparators for classes that you do not control

**Shallow vs Deep Copy -**

Shallow copies duplicate as little as possible. A shallow copy of a collection is a copy of the collection structure, not the elements. With a shallow copy, two collections now share the individual elements.

Deep copies duplicate everything. A deep copy of a collection is two collections with all of the elements in the original collection duplicated.

**finalize() -**

The general contract of {finalize} is that it is invoked if and when the JavaTM virtual machine has determined that there is no longer any means by which this object can be accessed by any thread that has not yet died, except as a result of an action taken by the

finalization of some other object or class which is ready to be finalized. The {finalize} method may take any action, including making this object available again to other threads; the usual purpose of {finalize}, however, is to perform cleanup actions before the object is irrevocably discarded.

The finalize method is never invoked more than once by a Java virtual machine for any given object.

**sleep() vs wait() -**

sleep() is a method which is used to hold the process for the time wanted but in case of wait() method thread goes in waiting state and it won’t come back automatically until we call the notify() or notifyAll().

A wait can be "woken up" by another thread calling notify on the monitor which is being waited on whereas a sleep cannot. Also a wait (and notify) must happen in a block synchronized on the monitor object whereas sleep does not.

The major difference is that wait() releases the lock or monitor while sleep() doesn’t releases any lock or monitor while waiting.

synchronized(LOCK) {

Thread.sleep(1000); // LOCK is held

}

synchronized(LOCK) {

LOCK.wait(); // LOCK is not held

}

Wait is used for inter-thread communication while sleep is used to introduce pause on execution, generally.

**Arrays.asList(ia) vs new ArrayList<Integer>(Arrays.asList(ia)) -**

Arrays.asList takes an array and creates a wrapper that implements List<Integer>, which makes the original array available as a list. Nothing is copied and only a single wrapper object is created. Operations on the list wrapper are propagated to the original array.

new ArrayList creates a full, independent copy of the original one. Although here you create the wrapper using Arrays.asList as well, it is used only during the construction of the new ArrayList and is garbage-collected afterwards.

**Iterbale vs Iterator -**

An implementation of Iterable is one that provides an Iterator of itself:

public interface Iterable<T>

{

Iterator<T> iterator();

}

An iterator is a simple way of allowing some to loop through a collection of data without assignment privileges (though with ability to remove).

public interface Iterator<E>

{

boolean hasNext();

E next();

void remove();

}

**Iterator vs ListIterator -**

We can use Iterator to traverse Set and List and also Map type of Objects in forward direction only.

Methods in Iterator:

hasNext()

next()

remove()

While a ListIterator can be used to traverse for only List-type Objects in either directions.

Methods in ListIterator:

hasNext()

next()

previous()

hasPrevious()

remove()

nextIndex()

previousIndex()

**HashMap vs HashTable vs ConcurrentHashMap -**

╔═══════════════╦═══════════════════╦═══════════════════╦═════════════════════╗

║ Property ║ HashMap ║ Hashtable ║ ConcurrentHashMap ║

╠═══════════════╬═══════════════════╬═══════════════════╩═════════════════════╣

║ Null ║ allowed ║ not allowed ║

║ values/keys ║ ║ ║

╠═══════════════╬═══════════════════╬═════════════════════════════════════════╣

║Is thread-safe ║ no ║ yes ║

╠═══════════════╬═══════════════════╬═══════════════════╦═════════════════════╣

║ Lock ║ not ║ locks the whole ║ locks the portion ║

║ mechanism ║ applicable ║ map ║ ║

╠═══════════════╬═══════════════════╩═══════════════════╬═════════════════════╣

║ Iterator ║ fail-fast ║ fail-safe ║

╚═══════════════╩═══════════════════════════════════════╩═════════════════════╝

Fail-Fast iterators immediately throw ConcurrentModificationException if a collection is modified while iterating over it. Where as Fail-Safe iterators don’t throw any exceptions if a collection is modified while iterating over it. Because, they operate on the clone of the collection, not on the actual collection.

In computer science, a priority queue is an abstract data type which is like a regular queue data structure, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority.

The servlet life cycle is managed by the servlet container. The life cycle contains the following steps:

Step 1 : Load Servlet Class.

Step 2 : Create Instance of Servlet.

Step 3 : Call the servlets init() method.

Step 4 : Call the servlets service() method.

Step 5 : Call the servlets destroy() method.

Step 1, 2 and 3 are executed only once, when the servlet is initially loaded. By default the servlet is not loaded until the first request is received for it. You can force the container to load the servlet when the container starts up though. See web.xml Servlet Configuration for more details about that.

Step 4 is executed multiple times - once for every HTTP request to the servlet. service() is the main method of handling the requests to the server. This method determines the type of the request (POST, GET, etc) and acts accordingly, by calling the specified methods like doPost() and doGet().

Step 5 is executed when the servlet container unloads the servlet.

**Web Server vs Application Server -**

Web Server is designed to serve HTTP Content. App Server can also serve HTTP Content but is not limited to just HTTP. It can be provided other protocol support such as RMI/RPC.

Web Server is mostly designed to serve static content, though most Web Servers have plug-ins to support scripting languages like Perl, PHP, ASP, JSP etc. through which these servers can generate dynamic HTTP content.

Most of the application servers have Web Server as integral part of them, that means App Server can do whatever Web Server is capable of. Additionally App Server have components and features to support Application level services such as Connection Pooling, Object Pooling, Transaction Support, Messaging services etc.

As web servers are well suited for static content and App servers for dynamic content, most of the production environments have web server acting as reverse proxy to App server. That means while servicing a page request, static contents (such as images/Static HTML) are served by web server that interprets the request. Using some kind of filtering technique (mostly extension of requested resource), web server identifies dynamic content request and transparently forwards to App server

**REST vs SOAP -**

What is a REST Web Service - The acronym REST stands for Representational State Transfer, this basically means that each unique URL is a representation of some object.

You can get the contents of that object using an HTTP GET, you then might use a POST, PUT, or DELETE to modify the object (in practice most of the services use a POST for this).

Who's using REST? - All of Yahoo's web services use REST, including Flickr, del.icio.us API uses it, pubsub, bloglines, technorati, and both eBay, and Amazon have web services for both REST and SOAP.

Who's using SOAP? - Google seems to be consistent in implementing their web services to use SOAP, with the exception of Blogger, which uses XML-RPC. You will find SOAP web services in lots of enterprise software as well.

As you may have noticed the companies I mentioned that are using REST api's haven't been around for very long, and their apis came out this year mostly. So REST is definitely the trendy way to create a web service.

The main advantages of REST web services are:

Lightweight - not a lot of extra xml markup, Human Readable Results, Easy to build - no toolkits required

JAX-RS: Java API for RESTful Web Services (JAX-RS) is a Java programming language API spec that provides support in creating web services according to the Representational State Transfer (REST) architectural pattern.

SOAP also has some advantages:

Easy to consume - sometimes Rigid - type checking, adheres to a contract

JAX-WS: Java API for XML Web Services (JAX-WS) is a Java programming language API for creating web services, particularly SOAP services. JAX-WS is one of the Java XML programming APIs. It is part of the Java EE platform.

RESTEasy is a portable implementation of JAX-RS specification which provides a Java API for RESTful Web Services over the HTTP protocol. The Java API for RESTful web services (JAX-RS) defines a standard annotation-driven API that helps developers to create Web services in Java and invoke it.

RESTFul Web services are based on the following set of principles that are different from the SOAP based Web services -

\* Addressable resources: which means that everything can be identified as a resource, and each resource is uniquely identified by a URI.

\* The use standard HTTP methods (GET, POST, PUT, DELETE) to interact with the resource.

\* Representation-oriented: you interact with services using representations of that service. A resource referenced by one URI can have different formats, as different platforms need different formats. (e.g. browsers need HTML, JavaScript needs JSON and so on).

\* Communication between the client and the endpoint is stateless. All the associated state required by the server is passed by the client in each invocation.

**Topics vs Queues -**

In JMS a Topic implements publish and subscribe semantics. When you publish a message it goes to all the subscribers who are interested - so zero to many subscribers will receive a copy of the message. Only subscribers who had an active subscription at the time the broker receives the message will get a copy of the message.

In a publish/subscribe (pub/sub) product or application, clients address messages to a topic. Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to the content hierarchy. The system takes care of distributing the messages arriving from a topic's multiple publishers to its multiple subscribers. Topics retain messages only as long as it takes to distribute them to current subscribers.

A JMS Queue implements load balancer semantics. A single message will be received by exactly one consumer. If there are no consumers available at the time the message is sent it will be kept until a consumer is available that can process the message. If a consumer receives a message and does not acknowledge it before closing then the message will be redelivered to another consumer. A queue can have many consumers with messages load balanced across the available consumers.

A point-to-point (PTP) product or application is built around the concept of message queues, senders, and receivers. Each message is addressed to a specific queue, and receiving clients extract messages from the queue(s) established to hold their messages. Queues retain all messages sent to them until the messages are consumed or until the messages expire.

**JNDI -**

The Java Naming and Directory Interface (JNDI) is a Java API for a directory service that allows Java software clients to discover and look up data and objects via a name. Like all Java APIs that interface with host systems, JNDI is independent of the underlying implementation.

JNDI allows distributed applications to look up services in an abstract, resource-independent way.