Beaconfire Inc, Home Work, Week6 Day24.
By (Ping) Nalongsone Danddank.
ndanddank@gmail.com
wechatID: ndanddank
Short Answer:

1. What is the difference between Runnable and Callable interface?

• Runnable won't return anything.
• Callable will return a Future object which can be used to retrieve the actual result at the later point in time.

2. What is ExecutorService?

-> is the java Concurrency API that use as a higher level replacement for working with thread directly. It provides a pool of threads and API for assigning tasks to it, and manage the pool so we donot have to create the thread manually.
• All threads of the internal pool will be reused under the hood of revenant tasks, so we can run as many as concurrent tasks as we want throughout the life-cycle of our application with a single executor service.
• submit - Submits a task to the ExecutorService, take both Runnable and Callable interface.
• Runnable won't return anything.
• Callable will return a Future object which can be used to retrieve the actual result at the later point in time.
• shutDown - Initiates an orderly shutdown in which previously submitted tasks are executed, but no new tasks will be accepted.
• awaitTermination - Blocks until all tasks have completed execution after a shutdown request, or the timeout occurs, or the current thread is interrupted, whichever happens first.
• shutDownNow - Attempts to stop all actively executing tasks, halts the processing of waiting tasks, and returns a list of the tasks that were awaiting execution.
• invokeAll - Executes the given tasks, returning a list of Futures holding their status and results when all complete.
• invokeAny - Executes the given tasks, returning the result of one that has completed successfully (i.e., without throwing an exception), if any do.

3. Describe when you would use @Async and ExecutorService to improve the performance of your web application.

-> when we want to process on the independent tasks, and we can run them in different thread and combine the result until all threads finish. Ex: we want to get the user infomation by id from a controller by a service and we also want to take the address info from a database too. So we can use different services that use their own thread separate to take the info from database then combine then together at the end of finished running thread. We could get faster time processing and improve the performance of our web application.

4. Explain the difference between Future and CompletableFuture.

->
A Future is used as a reference to the result of an asynchronous computation. It provides an isDone() method to check whether the computation is done or not, and a get() method to retrieve the result of the computation when it is done. Limitations of Future:
It cannot be manually completed, You cannot perform further action on a Future's result without

blocking, Multiple Futures cannot be chained together, can not combine multiple Futures together, No Exception Handling.
CompletableFuture implements Future and CompletionStage interfaces and provides a huge set of convenience methods for creating, chaining and combining multiple Futures. It also has a very comprehensive exception handling support.
Transforming and acting on a CompletableFuture: can attach a callback to the CompletableFuture using thenApply(), thenAccept() and thenRun() methods
Combining two CompletableFutures together
• CompletableFuture is used for asynchronous programming in Java
• Asynchronous programming is a means of writing non-blocking code by running a task on a separate thread than the main application thread and notifying the main thread about its progress, completion or failure
• We can use completable future instead of implementing Runnable interface to create new thread, but under the hook, it still uses the Runnable to implement the multi-threading.
• Here I will introduce some functions that we need to use to implement asynchronous process in web application.

5. Can you list several Java built-in functional interface and their usage?
-> Supplier: just return the value, and take no parameter.
-> Consumer: take parameter but no return any value.
-> Predicate: take parameter then return only Boolean value, true or false.
-> Function: table parameter then return the value.
-> UnaryOperator: take a single parameter and return a parameter of the same type.
-> BinaryOperator: take two parameter and return a single value, both parameters and the return type must be same
type.