Executor Framework and Servlet Config

Executor Framework (a part of Java SE)
Introduced in Java 5.
What's earlier support ?
Extends Thread Implements Runnable

Why Executor Framework?
If you have thousands of task to be executed and if you create each thread for
thousands of tasks, you will get performance overheads as creation and maintenance
of each thread is an overhead.
Executor framework solves this problem.
In executor framework, you can create specified number of threads and reuse them to
execute more tasks once it completes its current task.
It simplifies the design of creating multithreaded application and manages thread
life cycles.
The programmer does not have to create or manage threads themselves, that's the
biggest advantage of executor framework.

Important classes / interfaces for executor framework.

1. java.util.concurrent.Executor
This interface is used to submit new task.
It has a method called "execute".

```
public interface Executor {
 void execute(Runnable task);
}
```

2. ExecutorService
It is sub-interface of Executor.
Provides methods for
Submitting / executing Callable/Runnable tasks
Shutting down service
Executing multiple tasks etc.

3. ScheduledExecutorService
It is sub-interface of executor service which provides methods for scheduling tasks
at fixed intervals or with initial delay.

4. Executors
This class provides factory methods for creating thread pool based executors.

Important factory methods(=public static method rets instance of ExecutorService)
of Executors are:

4.1 newFixedThreadPool: This method returns thread pool executor whose maximum size
is fixed.If all n threads are busy performing the task and additional tasks are
submitted, then they will have to wait in the queue until thread is available.
4.2 newCachedThreadPool: this method returns an unbounded thread pool. It doesn't
have maximum size but if it has less number of tasks, then it will tear down unused
thread. If a thread has been unused for keepAliveTime , then it will tear it down.

4.3 newSingleThreadedExecutor: this method returns an executor which is guaranteed to use the single thread.
4.4 newScheduledThreadPool: this method returns a fixed size thread pool that can schedule commands to run after a given delay, or to execute periodically.

Steps for Runnable
1. Create a thread-pool executor , using suitable factory method of Executors.
eg : For fixed no of threads
ExecutorService executor = Executors.newFixedThreadPool(10);

2. Create Runnable task

3. Use inherited method
public void execute(Runnable command)
Executes this Runnable task , in a separate thread.

4. Shutdown the service
public void shutdown()
Initiates an orderly shutdown in which previously submitted tasks are executed, but no new tasks will be accepted.

5. boolean awaitTermination(long timeout,TimeUnit unit)
                throws InterruptedException
Blocks until all tasks have completed execution after a shutdown request, or the timeout occurs.

6. List<Runnable> 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.

BUT disadvantages with Runnable interface
1. Can't return result from the running task
2. Doesn't include throws Exception .

Better API
java.util.concurrent.Callable<V>
V : result type of call method
Represents a task that returns a result and may throw an exception.
Functional i/f
SAM :
public V call() throws Exception
Computes a result, or throws an exception if unable to do so.

Steps in using Callable i/f
1. Create a thread-pool executor , using suitable factory method of Executors.
eg : For fixed no of threads
ExecutorService executor = Executors.newFixedThreadPool(10);

2. Create Callable task , which returns a result.

3. To submit a task to executor service , use method of ExecutorService i/f :
public Future<T> submit(Callable<T> task)
Submits a value-returning task for execution and returns a Future representing the
pending results of the task. It's a non blocking method (i.e rets immediately)

The Future's get method will return the task's result upon successful completion.

If you would like to immediately block waiting for a task, invoke get() on Future.
eg : result = exec.submit(aCallable).get();
OR
main thread can perform some other jobs in the mean time & then invoke get on
Future , to actually get the results. (get : blocking call ,waits till the
computation is completed n then rets result)

4. Other methods of ExecutorService i/f
public List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks) throws
InterruptedException

It's a blocking call.(waits till all tasks are complete)
Executes the given tasks, returning a list of Futures holding their status and
results when all complete. Future.isDone() is true for each element of the returned
list.

5. Shutdown the service
        public void shutdown()
Initiates an orderly shutdown in which previously submitted tasks are executed, but
no new tasks will be accepted.

6. boolean awaitTermination(long timeout,TimeUnit unit)
                  throws InterruptedException
Blocks until all tasks have completed execution after a shutdown request, or the
timeout occurs.

7. List<Runnable> 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.

                    Regarding SERVLET CONFIG

A servlet specific configuration object created by a servlet container to pass
information to a servlet during initialization.

1. Represents Servlet specific configuration.
Defined in javax.servlet.ServletConfig -- interface.

2. Who creates its instance ?
Web container(WC)

3. When ?
After WC creates servlet instance(via def constr), ServletConfig instance is
created & then it invokes init() method of the servlet.
4. Usage

To store servlet specific init parameters.
(i.e the init-param is accessible to one servlet only or you can say that the
init-param data is private for a particular servlet.)

5. Where to add servlet specific init parameters?
Can be added either in web.xml or @WebServlet annotation.

XML Tags
<servlet>
    <servlet-name>init</servlet-name>
    <servlet-class>ex.TestInitParam</servlet-class>
    <init-param>
      <param-name>name</param-name>
      <param-value>value</param-value>
    </init-param>
</servlet>
<servlet-mapping>
<servlet-name>init</servlet-name>
<url-pattern>/test_init</url-pattern>
</servlet-mapping>

6. How to access servlet specific init params from a servlet ?
6.1 Override init() method
6.2 Get ServletConfig
Method of Servlet i/f
public ServletConfig getServletConfig()
6.3 Get the init params from ServletConfig
Method of ServletConfig i/f
String getInitparameter(String paramName) : rets the param value.

Regarding javax.servlet.ServletContext (i/f)

1. Defined in javax.servlet package.
2. Who creates its instance -- WC
3. When -- @ Web application (=context) deployment time
NOTE : The ServletContext object is contained within the ServletConfig object,
which the WC provides the servlet when the servlet is initialized.

4. How many instances ? --one per web application

5. Usages
5.1 Server side logging

API public void log(String mesg)
5.2 To create context(=application) scoped attributes
API public void setAttribute(String nm,Object val)
NOTE : Access them always in thread safe manner (using synchronized blocks)

5.3 To access global(scope=entire web application) parameters
How to add context scoped parameters ?

In web.xml
<context-param>
  <param-name>name</param-name>
      <param-value>value</param-value>
</context-param>
How to access these params in a Servlet ?
(can be accessed from init method onwards)

1. Get ServletContext
API of GenericServlet
ServletContext getServletContext() --method inherited from GenericServlet

2. ServletContext API
String getInitparameter(String paramName) : rets the param value.
eg : ctx param name : user_name value : abc
In the Servlet : getServletContext().getInitparameter("user_name") ---abc

----

What is a Servlet Listener(or web application listener)?

During the lifetime of a typical web application, a number of events take place.
eg : requests are created or destroyed.
sessions are created & destroyed
Contexts(web apps) are created & destroyed.
request or session or context attributes are added, removed, or modified etc.

The Servlet API provides a number of listener interfaces that one can implement in
order to react to these events.

eg : Event Listener i/f
1. ServletRequestListener
2. HttpSessionListener
3. ServletContextListener

Event Handling Steps
1. Create a class , implementing from Listener i/f.
2. Register it with WC
2.1 @WebListener annotation(class level)
OR
2.2 XML tags in web.xml

```
<listener>
 <listener-class>F.Q cls name of listener</listener-class>
</listener>
```