Goal
—---
Currently, specifying the --module-path, the --upgrade-module-path, the --limit-modules, and the --patch-module option in the CDS dumping command line results in the following message:

```
[0.036s][info][cds] Info: the --module-path option is
ignored when dumping the shared archive
```

and specifying the --module options in the AppCDS dumping command line results in the following error:

```
Error occurred during initialization of VM
Cannot use the following option when dumping the
shared archive: -m
```

Support matrix:

	-Xshare:dump	**-Xshare:{on, auto}**
--add-modules	Allow	Allow
--limit-modules[4]	Disallow - exit if specified	Allow - disable CDS
--module	Allow	Allow
--module-path <mp>	Allow	Allow[1]
--patch-module[3]	Disallow - exit if specified	Allow - disable CDS
--upgrade-module-path[2]	Disallow - exit if specified	Allow - disable CDS

[1] Different <mp> may be specified during dump time vs run time. If an archived class K was loaded from mp1.jar during dump time, but the changes in <mp> causes it to be available from a different mp2.jar during run time, the archived version of K will be disregarded at run time. K will be loaded dynamically.

[2] Currently, only two system modules are upgradeable (java.compiler, jdk.internal.vm.compiler). It won't be used too much initially for user's

developed modules. Lab is using --upgrade-module-path to upgrade Graal. This can be added in a subsequent enhancement if CDS and Lab Graal are used often together.

[3] As documented in JEP 261, —patch-module is strongly discouraged for production use.

[4] --limit-modules is intended for testing purpose to limit the observable modules (as in the image only contains the limited observable modules). SQE runs tests with --limit-modules java.base (for example to verify if the tests pass with a minimal image) at run time.

CSR JDK-8199710 has been approved for the change in behavior for the last 3 options.

Two different ways of specifying a module in a --module-path:

1. modular jar
2. exploded module

This RFE only supports modular jar files.

Dump time

———

Currently, during dump time, a buffer (_classpath_entry_table) is allocated for storing the -Xbootclasspath/a and the -cp paths info. This is being done in early stage during vm initialization – indirectly called from the init_globals() function.

The _classpath_entry_table is required during loading some boot classes. However, the modules will not be define until at a later stage of vm initialization during initPhase2. Therefore, when the _classpath_entry_table is being allocation, the number of –module-path entries is unknown at that time and the exact size cannot be allocated. The allocation of the _classpath_entry_table (renamed as _shared_path_table) will be done after all the modules have been defined, in order to capture the location info of the modular jars in the –module-path. The _shared_path_table contains the existing runtime image entry, followed by the -Xbootclasspath/a entries, followed by the -cp entries entries, and then followed by the entries with the 'location' ends with '.jar' from the ModuleEntryTable associated with the AppClassLoader.

Each element in the table points to a SharedClassPathEntry.

Later when we also support the --upgrade-module-path, we will need to include the entries from the ModuleEntryTable(s) associated with the boot

and platform class loaders.

At dump time, prior to the initialization of the _shared_path_table (its size being 0), the vm is loading the required system classes from the runtime image. During that time, the _classpath_index is set to 0 for those loaded classes.
After the initialization of the _shared_path_table, we loop through the _shared_path_table and compare the path of each entry with the ClassFileStream's source. If there's a match, 3 cases to be considered:

1. the source of ClassFileStream starts with "jrt:" or the source is a runtime image
    a. the class is from the runtime image, set _classpath_index to 0
2. the class is from an unnamed module
    a. if the loader is system class loader, make sure the _shared_path_table index is within _app_class_paths_start_index and _app_modue_paths_start_index before assigning to the _classpath_index
    b. if the _shared_path_table index is >= 1 and < _app_class_paths_start_index, assign to _classpath_index (the class is from the -Xbootclasspath/a)
3. the class is from a named module (class from the --module-path)
    a. make sure the _shared_path_table index is within the range of the --module-path portion of the table before assigning to the _classpath_index

Run time
————
Unlike the -cp, there's no restriction that the –module-path at dump time must equal to or a prefix of the one during run time.

The origination of a module class can be verified by comparing the archive class location with the runtime module location (part of the runtime shared class visibility check). However, for classes on the -cp, vm has no direct information of the class runtime location. Therefore, the -cp path is handled with more restrictions.

For the --module-path info portion of the _shared_path_table in the archive header, we will validate the timestamp and size of the jar file.

If the --module-path has been changed during run time but it contains the same modular jar during dump time. In this case, at runtime the classes will be loaded from the modular jar instead of from the archive.

Enhance the shared class visibility check to handle classes from –module-path for the SystemDictionary::is_system_class_loader() case.

- Existing null PackageEntry check:
  - the SharedClassPathEntry is not a runtime image
  - classpath_index is within the index into the -cp

Adding the following condition:

  - classpath_index is less than the _app_module_paths_start_index

- Existing non-null ModuleEntry check:
  - the SharedClassPathEntry is a runtime image (the class is from the runtime image)
  - the class is from an unnamed module and the classpath_index is within the index into the -cp

Adding the following condition:

  - the class is from a named module *and*

the classpath_index is within the range of the module path indices *and* the 'location' from the SharedClassPathEntry is the same as the one in the ModuleEntry

Test scenarios

———————

1. Existing tests with -cp only
   a. all existing CDS/AppCDS should pass
2. Simple test with –module-path and -m options in the command line
   a. java --module-path mod_dir -m myModule
      i. location of module is the same during dump time and run time
         - dump time: the classes in the --module-path should be archived
         - run time: the classes in the --module-path should be loaded from the archive
      ii. locations of module are different during dump time and run time
         - dump time: the classes in the --module-path should be archived
         - run time: the classes in the --module-path should

not be loaded from the archive

iii. location of module is the same during dump time and run time; with an non-existence path appended to the --module-path during run time
- the non-existence path will be ignored, the results are the same as in case i.

3. Test with -cp and –module-path and -m
   a. if the main class is specified such as:
      i. java -cp my.jar --module-path mod_dir -m myModule
      the Main class is expected to be found in myModule where the mod_dir contains the necessary modular jar(s) for myModule
      - dump time: the Main class should be archived
      - run time: the Main class should be loaded from the archive

      ii. java -cp my.jar --module-path mod_dir myMainClass -m myModule
      the myMainClass will be loaded from my.jar
      - dump time: the Main class should be archived from my.jar
      - run time: the Main class should be loaded from the archive

      iii. java -cp my.jar -–module-path mod_dir -m myModule myMainClass
      the Main class will be loaded from myModule
      (for cases ii and iii, the first located “Main” class will be used)
      - dump time: the Main class should be archived from the jar in the --module-path
      - run time: the Main class should be loaded from the archive

4. Test a modular jar in -cp depends on a module in -–module-path
   a. java -cp my.jar -–module-path mod_dir -–add-modules myModule myMainClass
      i. (e.g. myMainClass is in my.jar; this is to test classes in both jar and module will be archived and the archived classes will be loaded during run time with AppCDS enabled)
      - dump time: myMainClass and its dependent classes should be archived

- run time: myMainClass and its dependent classes should be loaded from the archive

5. Test a regular jar in -cp depends on a module in -–module-path
   a. java -cp my.jar -–module-path mod_dir -–add-modules myModule -–add-exports myModule/myPkg=ALL-UNNAMED myMainClass
      i. similar to 4 but with a regular jar file in the -cp
         - note that an unnamed module can have dependency on a named module but not the other way around
         - the results should be the same as in case 4.
6. Test with the --upgrade-module-path, the --limit-modules, and the --patch-module options
   a. For each of the above option
      i. dump time: vm should exit with an error message
      ii. run time: sharing will be disabled; classes will not be loaded from the archive
7. Test with --add-modules
   a. Dump with the --add-modules option, e.g. java --module-path mod_dir -Xshare:dump -XX:SharedClassListFile=test.classlist --add-modules mod1,mod2
      i. assuming test.classlist contains the names of classes in mod1 and mod2, the classes should be archived
   b. Make sure the classes can be loaded from the archive during run time.
8. Test with 2 directories specified in the --module-path. Each directory containing a modular jar.
   a. During run time, change the location of one of the modular jar.
      1. The class from the unchanged location of the modular jar should be loaded from the archive.
      2. The class from the changed location of the module jar should be loaded from the jar.
9. Test if the timestamp of a modular jar has been changed after dumping the archive, run time should fail with the appropriate error message. The behavior should be the same as before with -cp.
10. Test to show that JvmtiEnv::AddToBootstrapClassLoaderSearch and JvmtiEnv::AddToSystemClassLoaderSearch should disable AppCDS. The behavior should be the same as before with -cp.