ClassScan

The `[classscan]` component aims to provide a general-purpose library that can be used or extended to satisfy any potential usage pattern requiring information about the identity and structure of various classes provided by a given `ClassLoader`.

Goals

- Implementation details (BCEL, ASM, etc.) shouldn't be allowed to "bleed out" into consumer-facing APIs.
- any `new URL()` handling in the scanner must be pluggable. The underlying file systems on different systems are just too different.
- Performance
- Flexibility

Desired Capabilities

Filtering

Marker File

Multiple frameworks have a "marker" file that indicates a jar is eligible for processing. A ClassScan consumer will want to be able to filter a jar based on the maker file criteria:

- JSR-299 CDI - META-INF/beans.xml
- JSR-314 (JSF2) - META-INF/faces-config.xml
- JPA - META-INF/persistence.xml
- service provider - META-INF/services/*interface.to.be.implemented*
- Servlet/JSP - WEB-INF/web.xml (this is also marked by war suffix)

Content

Consumers will require different levels of granularity:

- Elements in classpath (Jar or File locations)
- Classes in each location
- Interfaces of each class
- Fields and/or methods of each class
- Annotations on each of the above

Cache

- Scan classpath and classes once for multiple consumers
- Balance memory v. time

Development Ideas

These are **ideas** and might not all be adopted; that's what the consenus-building process is for!

- Decompose into multiple modules to implement various subtasks in pluggable fashion.
- Define SPIs for aforementioned subtasks such that a given module can simply declare which services it provides.
- For implementations of a given service that may/must wrap other implementations, consider providing a default impl + a wrapper structure a la JSF

High-Level Modules

api

- Provides meta-model rooted at `org.apache.commons.classscan.MetaRegistry` but centering on `oacc.MetaClassLoader` which emphasizes `[classscan]`'s per-classloader organization.

- Defines basic SPIs for other high-level modules (assuming SPI approach)
  - spi to plug in URL handling
  - spi to plug in class digester
  - spi to plug in introspection of different ClassLoader types
- Hierarchy Walking
- Cache

BCEL

- BCEL-based class digester available

xbean

- xbean-finder could be plugged as a separate service impl

xbean-finder skips certain aspects of scanning (e.g. find all implementations of interface `Foo`) unless configured otherwise. This is an interesting concept that might be extended profitably to an "iterative scan approach," allowing explicit requests for certain types of information to trigger a "deeper" scan than that for which a given scanner impl is configured to do by default.

Reactor

This is perhaps the right level at which to provide higher-level mechanisms for working with the info in a `MetaClassLoader`. Based on the ideas of Mark Struberg and concepts in the `[meiyo]` sandbox component (expected to be more or less subsumed by `[classscan]`)

- Domain Specific Language parsers on top of hierarchy walker to filter which classes are of interest

Cache

With this approach, `MetaRegistry` would have a default implementation that simply deferred to the available **scanner**. The default implementation could then be wrapped by a `MetaRegistry` implementation that would provide a general-purpose cache for the class metadata (see "wrapper structure" idea under "development ideas"). This way caching could be implemented independently, and reusably, from scanning. This might or might not be compatible with the "iterative scan approach" ideated in the **Scanners** section.

Testing the Implementation

Replace the xbean-finder clones in branches of the following

- OpenEJB
- TomEE
- Geronimo
- Struts
- XBean