## ClassLoader

每个模块绝对隔离。包括ClassLoader、Spring Application Context。

ModuleClassLoader

ModuleClassLoader

注册到container

ContainerClassLoader

注册到container

ContainerClassLoader、ModuleClassLoader都继承URLClassLoader，但重写loadClass方法。

ContainerClassLoader是ModuleClassLoader的父亲。

加载类的顺序：

|  |  |
| --- | --- |
| 容器 | 先在ContainerClassLoader中查找，如果找不到就到每个ModuleClassLoader中查找。 |
| 自己有特定ClassLoader的模块 | 先在自己的ModuleClassLoader中查找，如果找不到就到每个ContainerClassLoader中去查找。ContainerClassLoader会在自己和其他的ModuleClassLoader中查找。 |
| 自己没有特定ClassLoader的模块 | 与容器的处理方式相同 |

特别要注意很多framework都没提供设置ClassLoader的接口，这样导致扩展性不好，他们是直接从Thread.currentThread().getContextClassLoader()中获取的。

比如Hibernate就是如此。

由于有了ClassLoader的隔离，在我们代码中的反射就一定要注意，最好显式地指定ClassLoader，否则会可能找不到类。

另外，由于我们由容器来逐个加载、初始化每个Module，所有模块都是在同一thread中进行加载与初始化的，这样就无法使用Thread.currentThread().setContextClassLoader()来设置当前线程的ClassLoader。

默认所有Thread的ClassLoader是ContainerClassLoader。

如果某处需要使用Thread的ClassLoader，需要开发者自行设置进去。

## Spring

SpringService来更新共享的ApplicationContext

核心容器有一个根ApplicationContext，里面有一些各个模块都可以访问的共享Bean。

每个模块可以有自己的模块ApplicationContext，里面都是模块级的Bean，这些模块ApplicationContext可以设置其父ApplicationConext为容器的根ApplicationContext。

如果某个模块需要把自己的Bean公布给其他模块（比如DB需要公布其SessionFactory Bean），则可以通过核心容器的SpringSerivce服务来注册自己的模块ApplicationContext（其父ApplicationContext为容器的根ApplicationContext）为核心容器的根ApplicationContext，这样实际上，核心容器的ApplicationContext包含了多个ApplicationContext。

但现在有问题：

1，一些基础性的共享Bean，尤其是BeanFactoryPostProcessor的Bean，只有在自己的ApplicationContext才生效，如果被其他模块引用，则不会重新执行postProcessBeforeInitialization方法，这样会导致一些通用的拦截处理不会在其他模块里执行。

解决方案： 自己重写一个postProccesor，来重新执行容器中所有的我们扩展的BeanFactoryPostProcessor。

2，模块之间共享Bean不是一种好方法，容易造成模块间的依赖。只是在没有特别好的解决方案的时候才需要使用，我们平台只是提供这种机制，上层网管可以不使用这种机制。

3，对于DB的扩展，比如自动扫描package，由于加入的模块可能package不同，这样就导致DB模块配置的${ db.packagesToScan}需要修改。现在的问题是，到底是通过API来设置，还是修改此处？如果是API，将提高实现的难度，因为要每个模块加载的时候去重新scan package，目前还不清楚是否可以实现。

4，logger的问题。目前没有实现到真正的Module级隔离。这与DB的情况类似。需要重新设计。

## 配置的变化

Module上增加了“selfFirstClassLoader=true/false”来区分该模块是使用自己的ModuleClassLoader，还是使用默认的ContainerClassLoader。

如果某个模块使用Spring，每个模块只加载自己模块的Spring xml文件。所以<property name="configLocations" value="classpath:conf/appctx-db.xml" /> 里面必须制定到具体的spring xml文件，不要使用通配符，否则可能会加载其他Module中的spring XML。

## 模块与Service

2种观点。

1，Module的start,stop完全由开发者自己控制。完全不支持xml中配置service，不支持springBean。

2，开发者自己控制、框架支持Declarative Service两种方式都支持。并提供缺省的beforeStart, start, afterStart的实现，上层开发者可以不使用缺省实现，也可以重写。

对spring的引用

目前还是考虑支持，毕竟平台会使用Spring。

但允许用户上层应用不使用，或者替换为其他IOC/DI容器。

通过定义抽象接口来屏蔽这种差异。

Service不需要通用的基类，它的生命周期完全由上层开发者自行控制。

生命周期：

模块何时放入容器中。（load后应该就放入容器中）

服务何时放入容器中。（模块在afterStart后，该模块的所有Service肯定是已经放入到容器中的）