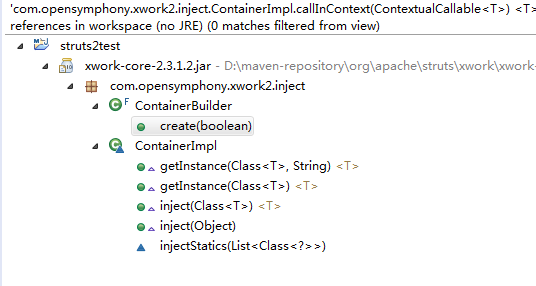
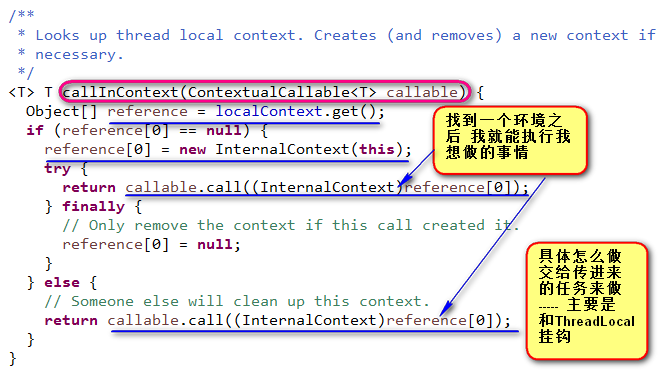
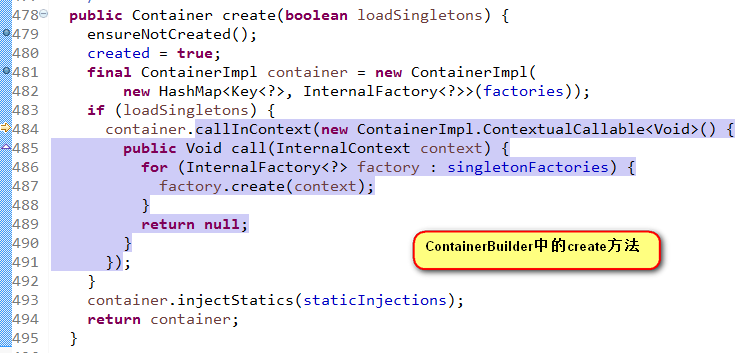
----- ContainerImpl#callInContext(ContextualCall<T>)<T>

方法名字的含义：在上下文中调用 ----- 非常含糊的一个名字----- 调用什么呢？ ---- 所以 怎么做也不知道 就是为了多线程环境下 获取可靠地环境 然后执行想要的操作 ---- 想要的操作从Callable参数传递进来

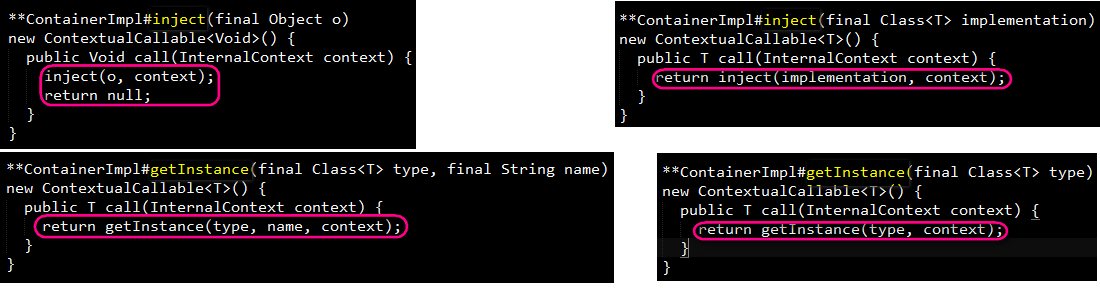




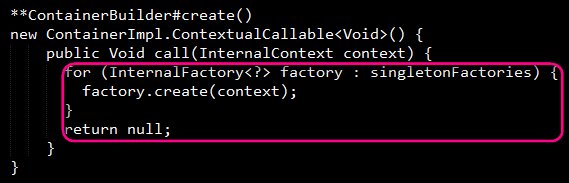
【以后要理解这个和ThreadLocal结合的具体好处】

 ------ 外部调用

------ ContainerImpl内部自己调用



这些方法无非是调用自己应该做的事情 该调用什么调用

 这个里面 就是调用工厂来创建自己的产品而已【ContainerBuilder里面】

【DefaultConfiguration里面 对Builder的构建 最后 创建出Container 这个是非常重要的过程 ----- builder的过程如何构建<Key<?>, InternalFactory> map的这个过程 很重要】

【还有就是 Container的构建使用了构造者模式 ----分析一下】

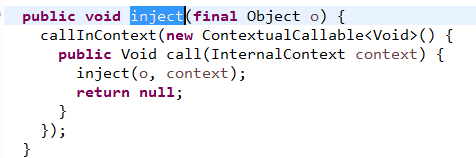
【再分析一下 通过ContainerImpl#getInstance() 获取对象 是每一次都使用工厂去生产对象 ----- 这个原因 主要是Struts2的Action 之类的对象 都不是单例的 为了和Servlet的模式分开 Action是每次请求都要创建的 ----- 所以 这就可以明白 Struts2中的容器缓存的不是对象的实例 是对象的工厂 因为对象在web环境可能要求是多例的 不是单例的 那么 你要是缓存对象 该缓存多少个？ 定不下来 好吧 那我们就缓存工厂 要的时候 直接使用工厂 ------ 所以 工厂很重要 所以 spring和struts2中的Container是不一样的】

【正因为是多例 还想提高执行效率 所以 用工厂 不是反射工厂 里面就是封存了new的过程！！！ 所以 这里面 都没有用到反射 提高getInstance ------ Inject的执行效率 但是 没有缓存set的map 所以 set使用的是反射】

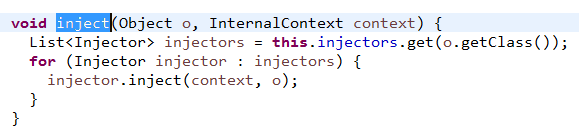
【FieldInjector和methodInjector是Injector这个接口的具体实现 是用来注入 实现了inject方法】

【归纳ContainerImpl中有哪些是其内部类 总结出来】

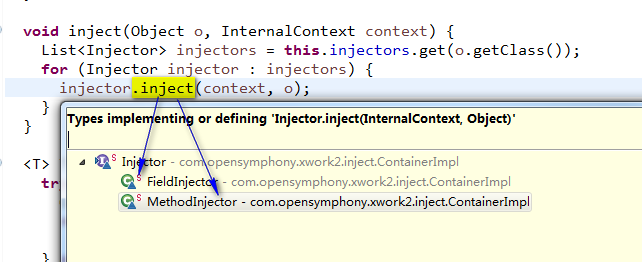
【归纳ContainerImpl的初始化流程（准备数据） ------ getInstance之后（内部对象 应该是在这个之后 还调用了inject） ----- 和 inject的流程 】

可以inject(Object o) -> 

点进去inject



每一个injector都是inject ----- 点击这个inject



这样才知道FieldInjector和methodInjector的意义

所以 因为web环境是每次进来都是新对象 所以 我们存储的是Map<Key, InternalFactory> factories; 和 Map<Class, List<Injector>> injectors; 这两个动态操作的缓存对象 ------- 这样 每次进来 我都通过factories生成新的实例（这个是new出来的） + inject (注入新的成员 ----- 这个和Spring不同 Spring是单例的 所以 DI的inject是要查找出来的 这个里面 不用查找 直接new出来 ----- 这是不是也是Struts比较卡的地方呢)】

【ObjectFactory中方法的总结

】

Enumeration和Iterator的比较

<http://www.cnblogs.com/skywang12345/p/3311275.html>

<http://cache.baiducontent.com/c?m=9f65cb4a8c8507ed4fece763105392230e54f73260878e482a958448e435061e5a27baf062714c51d0c57d6106b20b41ebf13571370627a09bbfd20c82e5866d72c8713b284fc21205d368f897017a9660c31cb2f245&p=8c7dd1078e934eac58ec8c2d021497&newp=882a9645d7871ce64bf7c7710f089c231610db2151d7d2106b82c825d7331b001c3bbfb423231b01d6c37e6506a84956ecf3317436072ba3dda5c91d9fb4c57479cc7a7d6306&user=baidu&fm=sc&query=Java+Enumeration&qid=fb99ef1d00028ae2&p1=3>

总之 是Enumeration和Iterator之间的转换