**Spring Study Notes**

**@author Count Monte Cristo**

**@create D20150731**

**@description 记录Spring重要的知识**

**@motto VICTORY LOVES PREPARATION**

**@criteria 凯旋基诺**

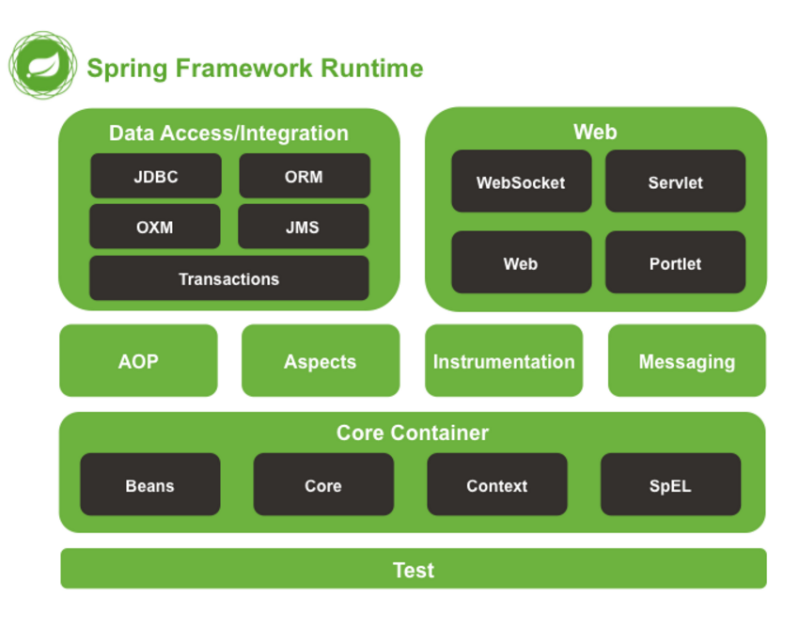
**@version 2016年05月01日 19:56**

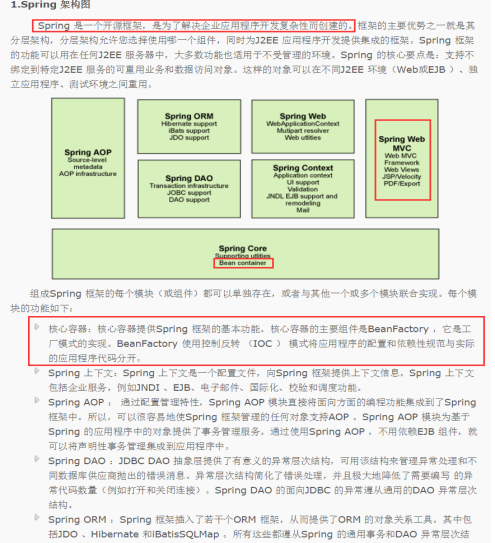
**2016年04月08日 13:31:15**

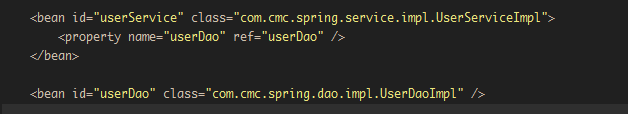
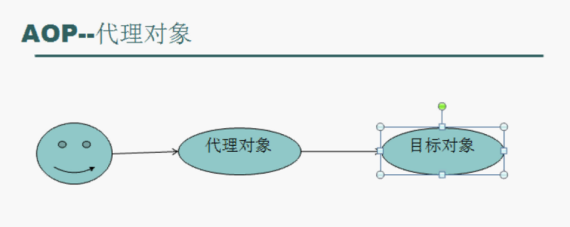
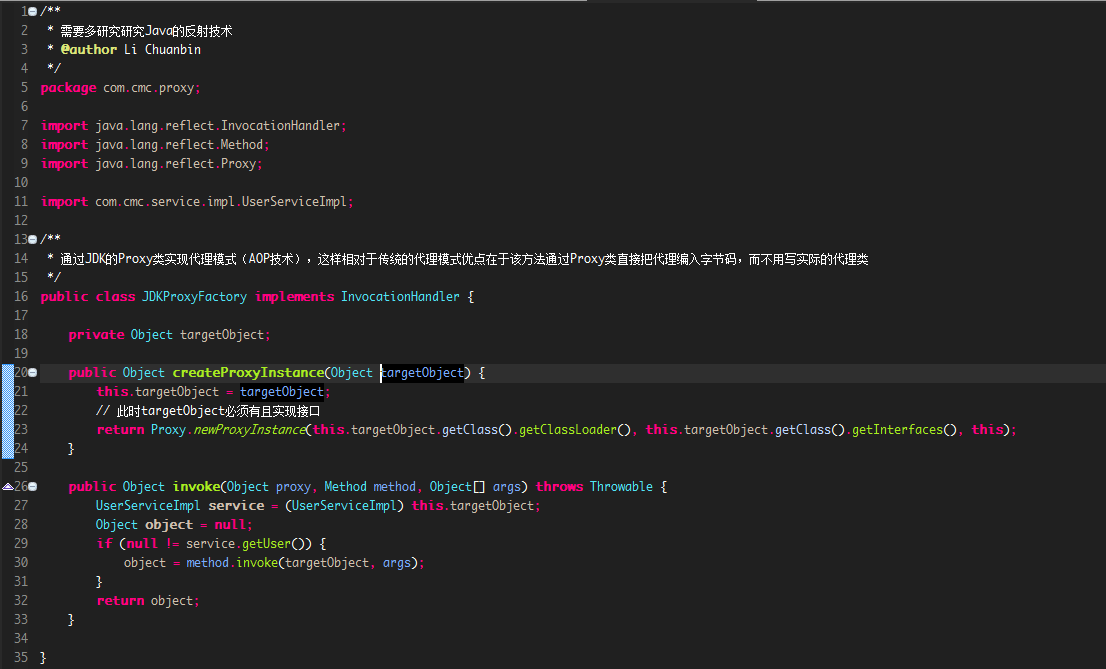
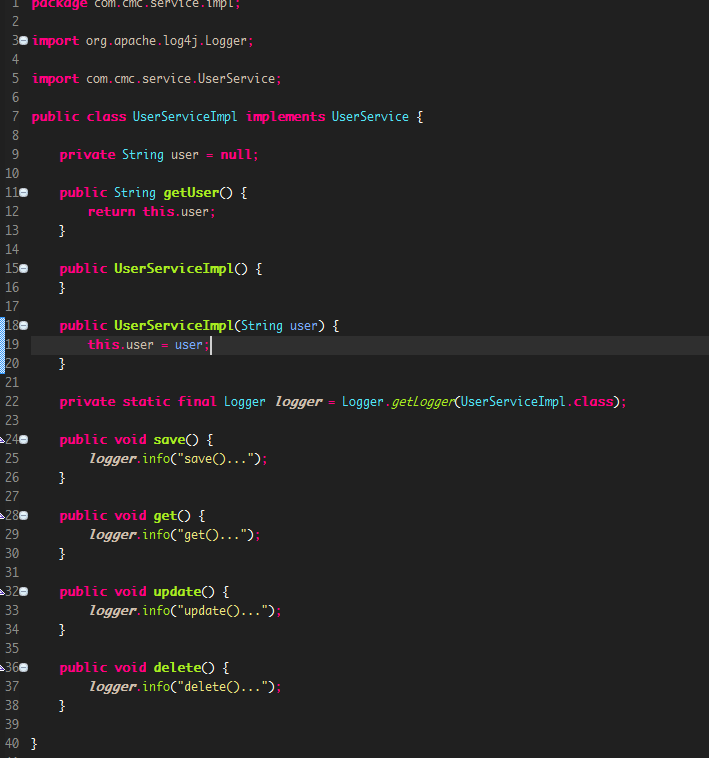
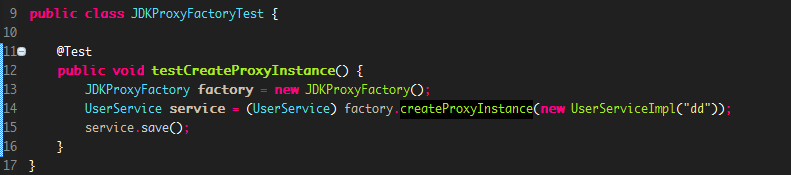
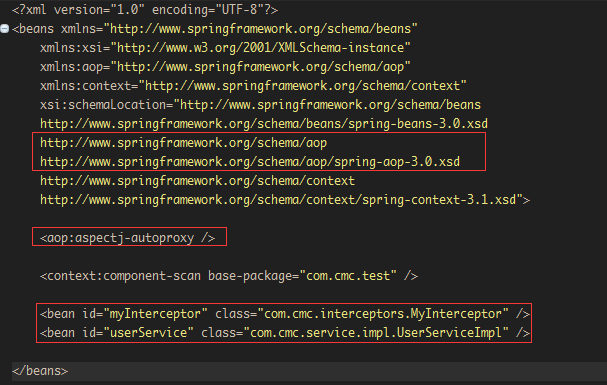
**v1.0.6.20160224\_213615\_alpha**

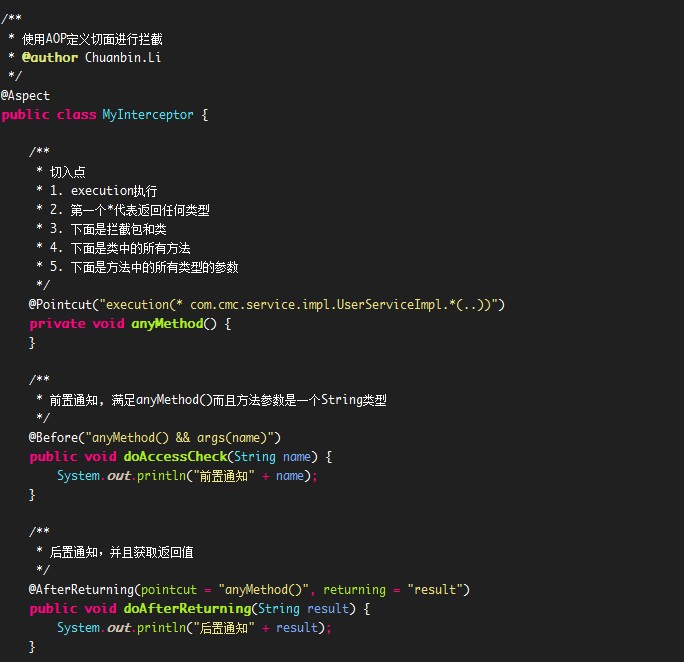
**~~D20150731~~ ~~D20151103~~ ~~D20151201~~ ~~D20151228~~ ~~D20160126~~ ~~D20160121~~**

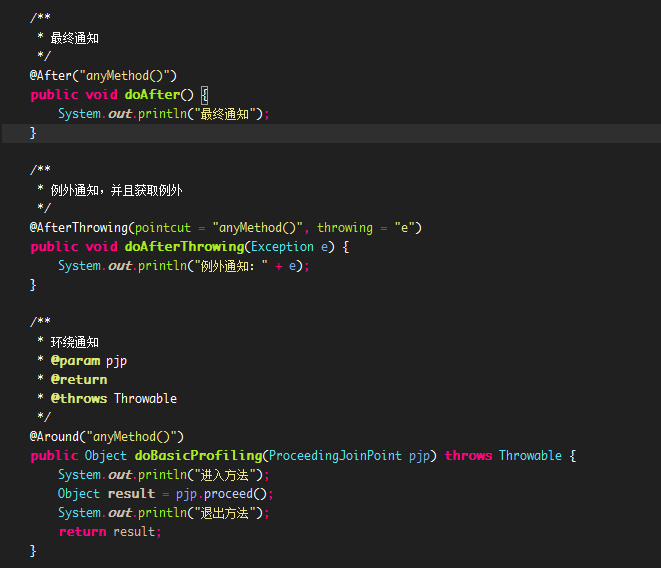
**目录**

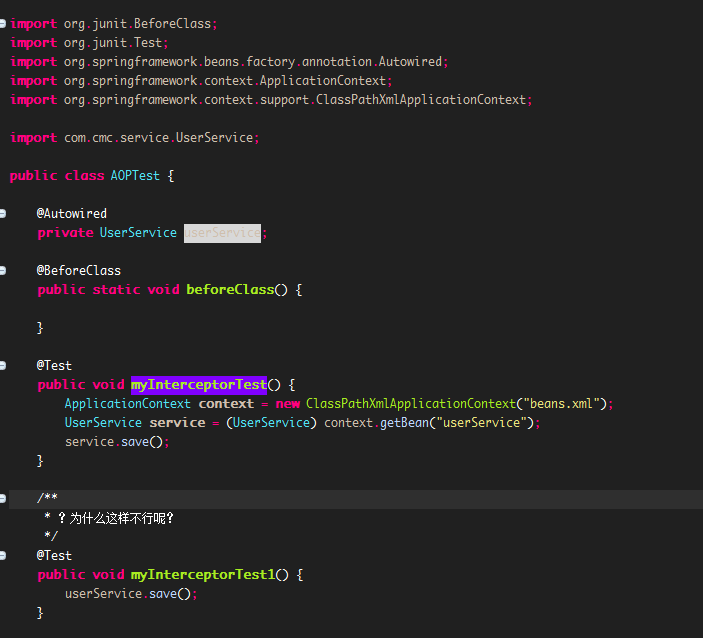
1. **Spring架构图**
2. **Spring两大特性**
3. **Spring的好处**
4. **Bean的作用域**
5. **Bean注入**
6. **@Autowired和@Resource**
7. **Spring管理Bean**
8. **Spring AOP（Proxy）技术**
9. **Spring集成JDBC**
10. **其他**
11. **Spring 架构图**

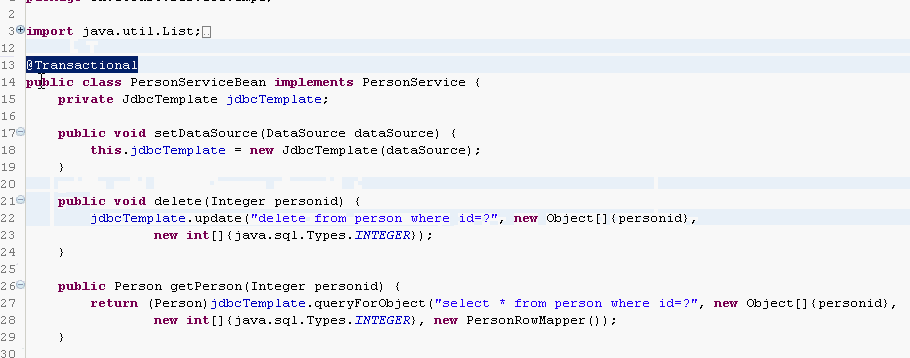


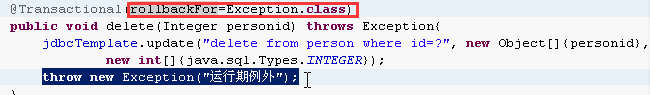
1. **Spring两大特性**
   1. **控制反转（IoC）**
      1. 含义：应用本身不负责依赖对象的创建和维护，而是由外部容器负责的。这样控制权就由应用转移到了外部容器，控制权的转移就是控制反转。
      2. 扩展：只是此“控制反转”的控制权指的是应用对象的创建和维护的控制权。
   2. **依赖注入（DI）**
      1. 含义：在运行期，由外部容器动态地将依赖对象注入到组件中。
2. **Spring的好处**
   1. 降低组件（类）之间的耦合度（组件还是通过Spring调用了要调用的对象，耦合度？难道是应为被Spring容器承担了就降低了原来组件的耦合度？？？，@answer 降低耦合度是对基本单位类而言的，也就是说耦合度是指类与类之间的关系，Spring通过容器管理Bean然后自动注入，这样的话就不用new需要的对象，这样就减少了“个性”,提高了共性，减少与其他具体类之间的耦合），实现软件各层之间的解耦：Controller、Service和Dao层；
   2. 可以使用容器提供众多的服务，如：事务管理服务、消息服务、JMS服务、Spring Core服务、持久化服务等等。当我们使用容器提供的服务的时候，如事物管理服务，此时开发人员就不需要手工控制服务，不需要处理事物其他复杂的操作（如事物传播？）（@inspiration 此种服务比较高级的，以后可以买这样的服务，有待深入研究）；
   3. **容器提供了AOP技术，使用它很容易实现权限拦截、运行监控等功能**
   4. 容器提供了众多辅助类，使用它们可以进行快速开发
   5. 对主流的应用框架提供了集成支持，如Hibernate、JPA、Struts等，便于降低集成困难
3. **Bean的作用域**
   1. Singleton（单例）、Prototype（新的实例对象）
   2. **Spring中默认情况下Controller是也是singleton的，一般情况下控制器中不应该出现动态变量，因为singleton线程不安全，如果必须使用动态变量，那么要将控制器bean的scope配置为prototype（一个请求一个实例）**
   3. 实例
      1. 默认情况下bean的scope都是singleton的，如果需要改变可以通过制定scope的值进行修改，如果希望一批bean的scope都需要改变，则可以在beans节点设置scope
      2. singleton的bean在容器加载的时候就会进行实例化，而prototype的则会在使用bean的时候才会被实例化
      3. Singleton的bean也可以通过init-lazy属性（true、false）修改类实例化的时间（Spring容器实例化的时候还是被调用的时候），此时如果设置init-method属性，则会在加载Spring容器的时候执行init-method指定的方法，不管有没有设置init-lazy，destory-method属性则是在bean被销毁（关闭Spring容器的时候bean才会被销毁 ）的时候执行的方法。
      4. 注意：如果在被调用的时候实例化的话，如果bean的配置以及bean内部（如连接数据库）发生错误，就不能在Spring容器加载的时候检测到，但是却可以减少Spring容器初始化的负担
4. **Bean注入**
   1. 属性注入
      1. 实例
      2. 对应的UserServiceImpl一定要有一个userDao属性并且该属性必须要有set方法，此时的userDao的名称是根据UserServiceImpl对应的set方法指定的
      3. 也可以使用内部bean（参考官方文档）进行注入属性
   2. 集合注入
   3. 构造方法注入
      1. Bean中配置<constructor-arg>子标签（如何配置，参见官方开发手册），和具体bean中的构造方法对应
   4. 注入依赖对象可以采用**手工装配（在bean标签中设置autowire属性，参照reference，自动装配也分为构造器注入、setter方法注入和Field注入，但是此时不用写@autowired或者@Resource了）和自动装配，在实际应用中建议使用手工装配（xml和annotation方式），因为自动装配容器产生“不可预见”的错误。**
5. **@Autowired和@Resource**
   1. @Autowired默认按照类型进行装配（此时如果要注入serviceImpl的话可以直接在controller中要注入的service上加上@Autowired，此时会自动（多态）注入serviceImpl，**因为serviceImpl也是service的一种类型**）
   2. @Resource默认按照名称进行装配，找不到对应名称才按照类型进行装配
   3. 注解注入（手动注入）是不需要写setter方法的（Spring已经做了setter的工作，通过反射），如果是xml注入则需要写setter方法，当时还没有使用反射帮助类写setter方法，只能利用类本身的setter方法
   4. 注意
      1. **上下文配置**
         1. 使用<context:annotation-config /> 进行隐式注册注释的处理器（可以参照reference）
            1. AutowiredAnnotationBeanPostProcessor -- @Autowired
            2. CommonAnnotationBeanPostProcessor -- @Resource
            3. PersistenceAnnotationBeanPostProcessor -- 持久化
            4. RequiredAnnotationBeanPostProcessor -- @Required
         2. 使用<context:component-scan base-package="com.ucmed.common.web" />
            1. 该标签默认包含标签<context:annotation-config />的功能
6. **Spring管理Bean**
   1. **容器加载Bean的实质**
      1. 先XML解析获取Spring配置文件中的bean，然后利用java的反射机制实例化bean（Class.forName(Clazz.class)）之类
   2. 打开主键扫描控制项 <context:component-scan base-package=”” /> ，把加上注解（@Component、@Controller、@Service、@Repository）的类纳入Spring容器管理，使用该配置就可以去掉<context:annotation-config />
   3. 具体注解解析
      1. @Component：泛指组件，标注的类都会纳入Spring容器管理
      2. @Controller：Controller层
      3. @Service：Service层
      4. @Repository：Dao层
      5. @Qualifier：限定词
         1. 只使用@Autowired注解进行注入的时候要求匹配的bean有且只有一个，当有多个的时候Spring允许通过@Qualifier注解进行具体Bean名称的指定，这样@Autowired和@Qualifier结合就从byType注入方式变成了byName，当没有一个bean匹配的时候可以使用@Autowired(Require = false)，此时并不一定需要注入标注的bean。
      6. 注意：修改bean的名称默认是AAA为aAA，可以指定bean的名称
   4. Bean默认是单例模型可以通过@scope设置作用域范围
   5. Bean初始化注解javax.annotation.PostConstruct.@PostConstuct（构造之后） -- init-method
   6. Bean销毁前注注解javax.annotation.PreDestory.@PreDestory（销毁之前） -- destory-method
7. **Spring AOP（Proxy）技术**
   1. **示意图**
   2. **使用CGLIB实现AOP**
      1. 此处省略，@reference 16\_传智播客Spring2.5视频教程\_使用CGLIB实现AOP功能与AOP概念解释
   3. **使用JDK中的Proxy技术实现AOP功能**
      1. JDKProxyFactory
      2. **UserServiceImpl**
      3. **JDKProxyFacotyTest**
   4. **AOP名词简介**
      1. 切面（aspect）：横切性关注点的抽象（关于连接点所写的类）即为切面，它与类类似，只是两者的关注点不同，类是对物体特征的描述，而切面是横切性关注点的描述。@description 切面就是关于横切性关注点（往往是某个业务中的某个环节）所写的一系列方法，如权限控制类等
      2. 连接点（joinpoint）：指那些被拦截的点（方法）。注意：在Spring中这些点指的是方法，实际上还可以是属性或者构造器
      3. 切入点（pointcut）：就是指对连接点进行拦截的定义
      4. 通知（advice）：拦截到连接点之后要做的事情，分为前置通知（符合某种要求开始执行拦截点之前的处理）、后置通知、异常通知、最终通知（finally{}里面的处理）和环绕通知（还没有判断是否符合某种要求之前进行的处理）
      5. 目标对象（target）：代理的类
      6. 织入（weave）：指将切面应用到目标对象并导致proxy对象创建的过程
      7. 引入（introduction）：在不修改代码的前提下，在运行期为类动态添加一些方法或者field
   5. **实例**
      1. **几种通知的位置**
      2. **实例（基于注解版本）**
         1. **Beans.xml**
         2. **基于AOP的Interceptor**

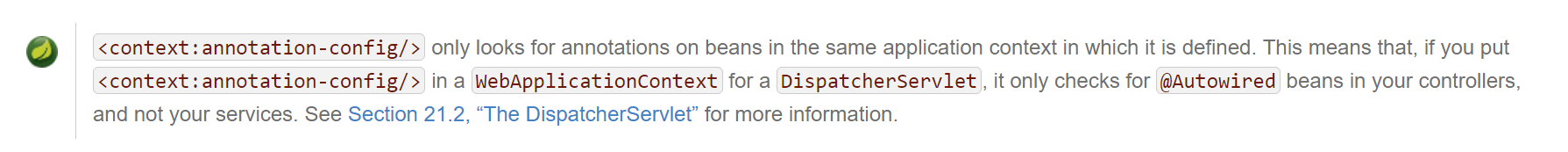




* + - 1. **测试方法**
         1. **方法内部加载上下文（**因为在方法内部加载的上下文，所以不能使用注解的方式注入Bean，通过context获取Bean**）**
         2. **方法外部加载上下文（**因为在类上加载了上下文文件，所以可以通过注解的方式注入Bean**）**
    1. **注意**
       1. 注意点已经全部写在了注释上

1. **Spring集成JDBC**
   1. 引入事务dtd文件（tx）
   2. 配置事务Bean
   3. 启动事务注解
   4. 实例
   5. 注意
      1. @Transactional注解可以加在类上也可以加载在方法上
      2. 运行期例外RuntimeException（unchecked例外，也就是说该异常系统不会捕获）事务会回滚的，否则（checked例外，该异常系统会捕获）事务不会回滚（前面的都是默认情况下），当然可以通过属性指定所有异常都要回滚或者某些异常不需要回滚



* + 1. Spring事务范围只是其context中，而ContextLoaderListener和ServletContext不是共用同一个context
       1. **@reference Spring Reference**

1. **其他**
   1. 事务是相对于一个Connection而言的，两个Connection不可能共用一个事务？
   2. 轻量级应用和重量级应用
      1. 是根据一个应用提供的服务(已经打开，没有打开的不算，因为有的容器的功能是默认关闭的)的多少来确定的