#### 一、IOC概述

1. **概念：控制反转（Inverse of Control）DI 依赖注入**

某一接口具体实现类的选择控制权从调用类中移除，转交给spring容器由Bean配置来进行控制。

1. IOC的类型：构造函数注入、属性注入、接口注入。
2. 通过容器完成依赖注入：通过new XmlBeanFactory("beans.xml")等方式启动容器，在容器启动时，spring根据配置文件的描述信息，自动实例化bean并完成依赖关系的装配，从容器中即可返回Bean实例。

#### 二、相关的Java基础知识

1. 类装载器：Class文件由类加载器装载后，在JVM中将形成一份描述Class结构的元信息对象，通过该元信息对象可以获取Class的结构信息。用户借助这个元信息对象间接调用Class对象的功能。
2. 类装载器的工作机制和方法

根装载器使用C++编写，在java中看不到，它负责装载JRE的核心类库。根装载器->ExtClassLoader->AppClassLoader。JVM使用全盘负责委托机制。先委托父装载器，没法再有子。

#### 三、资源访问器

  1.spring设计了一个Resource接口，exitst()资源是否存在、isOpen()资源是否打开、getURL()、getFile()、getInputStream()返回资源对应的输入流。

  如果有一个文件位于Web应用的类路径下，用户可以通过以下方式访问：

  1）通过FileSystemResource以文件系统绝对路径的方式进行访问。

  2）通过ClassPathResource以类路径的方式进行访问。

  3）通过ServletContextResource以相对于web应用根目录的方式进行访问。

   在获取资源后，用户可以通过Resource接口定义的方法访问文件的数据和信息。getFileName()获取文件名、getFile()获取File对象、getInputStream()获取文件的输入流。

   而WritableResource接口，通过getOutputStream()方法直接获取文件的输入流。

public class FileSourceExample {  
     
   public static void main(String[] args) {  
      try {  
         String filePath = "D:/cnslp/hello.txt";  
         //使用系统文件路径方式加载文件  
         WritableResource res1 = new PathResource(filePath);  
         //使用类路径方式加载文件  
         Resource res2 = new ClassPathResource("conf/file1.txt");  
         //使用WritableResource接口写资源文件  
         OutputStream stream1 = res1.getOutputStream();  
         stream1.write("欢迎光临\ncnslp论坛".getBytes());  
         stream1.close();  
         //使用Resource接口读取资源文件  
         InputStream ins1 = res1.getInputStream();  
         InputStream ins2 = res2.getInputStream();  
         ByteArrayOutputStream baos = new ByteArrayOutputStream();  
         int i;  
         while((i=ins1.read())!=-1){  
            baos.write(i);  
         }  
         System.out.println(baos.toString());  
            System.out.println("res1:"+res1.getFilename());  
            System.out.println("res2:"+res2.getFilename());              
      } catch (IOException e) {  
         e.printStackTrace();  
      }  
   }  
}

   2.通过EncodeResource对资源进行编码，以保证内容的正确性。

Resource res = new ClassPathResource("conf/file1.txt");  
EncodedResource encRes = new EncodedResource(res,"UTF-8");//使用UTF-8编码  
String content  = FileCopyUtils.copyToString(encRes.getReader());  
System.out.println(content);

3.资源加载

 通过资源地址的特殊标识来访问相对应的资源

 1）classpath：从类路径中加载资源，classpath:和classpath:/是等价的；

 2）file: 使用url从文件系统目录中装载资源；

 3）http:// 使用UrlResource从web服务器中装载资源；

 4）ftp://  使用UrlResource从FTP服务器中装载资源；  
注意：classpath\*:会扫描所有jar包及类路径；

Ant风格：？匹配文件名的一个字符

     \*匹配文件名中任意字符

     \*\*匹配多层路径

ResourcePatternResolver resolver = new PathMatchingResourcePatternResolver();  
Resource resource = resolver.getResource("classpath:conf/file1.txt");  
System.out.println(resource.getFilename());  
System.out.println(resource.getDescription());

注意：使用Resource操作文件，如果项目发布打包，则不能使用Resource的getFile()方法，应该使用getInputStream()来读取。

#### 四、BeanFactory和ApplicationContext

  应用上下文建立在BeanFactory基础上，一般称BeanFactory为IOC容器、ApplicationContext为应用上下文或者spring容器。

  1.BeanFactory类的通用工厂，不过开发一般直接使用ApplicationContext

  2.ApplicationContext

  1）ApplicationContext类的体系结构

  ApplicationContext的主要实现类是ClassPathXmlApplicationContext(默认从类路径加载配置文件)和FileSystemXmlApplicationContext(默认从文件系统中装载配置文件)

注：BeanFactory在初始化容器时，并未实例化bean，直到第一次使用才实例化，而ApplicationContext则在初始化上下文的时候就实例化bean。

@Configuration//表示是一个配置信息提供类  
public class Beans {  
   @Bean(name = "car")//定义一个Bean  
   public Car buildCar() {  
      Car car = new Car();  
      car.setBrand("红旗CA72");  
      car.setMaxSpeed(200);  
      return car;  
   }  
}

public static void main(String[] args) {  
   //通过一个带@Configuration的POJO装载Bean配置  
   ApplicationContext ctx = new AnnotationConfigApplicationContext(Beans.class);  
   Car car =ctx.getBean("car",Car.class);  
}

  2）WebApplicationContext类的体系结构

  WebApplicationContext专门为web应用准备的，它允许从相对应Web根目录路径中装配配置文件完成初始化。WebApplicationContext初始化需要ServletContext实例，必须在web容器的前提下才能完成启动工作，在web.xml中配置自启动的Servlet或定义Web容器监听器可以完成启动spring web应用上下网的工作。

  由于WebApplicationContext需要使用日志功能，所以用户把log4j的配置文件放在WEB-INF/classes下，这时log4j引擎即可顺利启动。如果log4j配置文件放在其他位置，需要在web.xml中指定器位置。

<context-param>  
<param-name>log4jConfigLocation</param-name>  
<param-value>/WEB-INF/classes/log4j.properties</param-value>  
</context-param>  
  
<servlet>  
    <servlet-name>log4jServlet</servlet-name>  
    <servlet-class>org.springframework.web.util.Log4jConfigServlet</servlet-class>  
    <load-on-startup>1</load-on-startup>  
</servlet>  
  
<servlet>  
    <servlet-name>springServlet</servlet-name>  
    <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>  
    <load-on-startup>2</load-on-startup>  
</servlet>

对于<context-param>主要是用来定义全局变量给整个项目共用。

* 容器初始化过程：
  1. 在启动Web项目时，容器(比如Tomcat)会读web.xml配置文件中的两个节点<listener>和<contex-param>。
  2. 接着容器会创建一个ServletContext(上下文),应用范围内即整个WEB项目都能使用这个上下文。
  3. 接着容器会将读取到<context-param>转化为键值对,并交给ServletContext。
  4. 容器创建<listener></listener>中的类实例,即创建监听（备注：listener定义的类可以是自定义的类但必须需要继承ServletContextListener）。
  5. 在监听的类中会有一个contextInitialized(ServletContextEvent event)初始化方法，在这个方法中可以通过event.getServletContext().getInitParameter("contextConfigLocation") 来得到context-param 设定的值。在这个类中还必须有一个contextDestroyed(ServletContextEvent event) 销毁方法.用于关闭应用前释放资源，比如说数据库连接的关闭。
  6. 得到这个context-param的值之后,你就可以做一些操作了.注意,这个时候你的WEB项目还没有完全启动完成.这个动作会比所有的Servlet都要早。

**容器对于web.xml的加载过程是context-param >> listener  >> fileter  >> servlet**