在《Spring源码学习一：源码分析概述》的示例代码中使用如下方式去加载Spring的配置文件并初始化容器，

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationgContext.xml");

在web应用中，配置文件都是自动加载的，示例代码中的方式就不能满足需求了。在web应用中使用Spring，需要在web.xml中添加如下配置。

```
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath:applicationContext.xml;
    </param-value>
</context-param>
```

先了解一下ServletContextAttributeListener、ServletContextListener。ServletContext定义了一些方法方便ServletContext容器进行通讯，在一个web应用中所有的Servlet都公用一个ServletContext，Spring在将web应用组合使用的时候，是将Spring的容器存到ServletContext中的，通俗的说就是将一个ApplicationContext存储到ServletContext的一个Map属性中；而ServletContextListener用于监听ServletContext一些事件，外部只有ContextLoaderListener开始，在web应用启动读取web.xml时，发现配置了ContextLoaderListener，而ContextLoaderListener实现了ServletContextListener接口，因此会执行ContextLoaderListener实中的contextInitialized方法，方法的具体代码如下，

```
public void contextInitialized(ServletContextEvent event) {
    initWebApplicationContext(event.getServletContext());
}
```

继续进入initWebApplicationContext方法，这个方法在其父类ContextLoader实现，根据方法名可以看出这个方法是用于初始化一个WebApplicationContext，简单理解就是初始化一个Web应用下的Spring容器，方法的具体代码如下，

```
public WebApplicationContext initWebApplicationContext(ServletContext servletContext) {
    if (servletContext.getAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE) != null) {
        throw new IllegalStateException(
                "Cannot initialize context because there is already a root application context present - " +
                "check whether you have multiple ContextLoader* definitions in your web.xml!");
    }

    Log logger = LogFactory.getLog(ContextLoader.class);
    servletContext.log("Initializing Spring root WebApplicationContext");
    if (logger.isInfoEnabled()) {
        logger.info("Root WebApplicationContext: initialization started");
    }
    long startTime = System.currentTimeMillis();

    try {
        if (this.context == null) {
            this.context = createWebApplicationContext(servletContext);
        }
        if (this.context instanceof ConfigurableWebApplicationContext) {
            ConfigurableWebApplicationContext cwac = (ConfigurableWebApplicationContext) this.context;
            if (!cwac.isActive()) {
                if (cwac.getParent() == null) {
                    ApplicationContext parent = loadParentContext(servletContext);
                    cwac.setParent(parent);
                }
                configureAndRefreshWebApplicationContext(cwac, servletContext);
            }
        }

        servletContext.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE, this.context);

        ClassLoader ccl = Thread.currentThread().getContextClassLoader();
        if (ccl == ContextLoader.class.getClassLoader()) {
            currentContext = this.context;
        }else if (ccl != null) {
            currentContextPerThread.put(ccl, this.context);
        }

        if (logger.isDebugEnabled()) {
            logger.debug("Published root WebApplicationContext as ServletContext attribute with name [" +
                    WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE + "]");
        }
        if (logger.isInfoEnabled()) {
            long elapsedTime = System.currentTimeMillis() - startTime;
            logger.info("Root WebApplicationContext: initialization completed in " + elapsedTime + " ms");
        }

        return this.context;
    }catch (RuntimeException ex) {
        logger.error("Context initialization failed", ex);
        servletContext.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE, ex);
        throw ex;
    }catch (Error ex) {
        logger.error("Context initialization failed", ex);
        servletContext.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE, ex);
        throw ex;
    }
}
```

方法的第一行就是检查servletContext是否已经存储了一个默认名称的WebApplicationContext，因为在一个应用中Spring容器只能有一个，所以需要校验，至于这个默认的名称为什么是WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE，看到后面就会慢慢明白。直接关注重点代码，代码如下。

```
if (this.context == null) {
    this.context = createWebApplicationContext(servletContext);
}
```

这里的context是在ContextLoader的一个变量，声明代码如下。

```
private WebApplicationContext context;
```

继续进入createWebApplicationContext方法，具体代码如下。

```
protected WebApplicationContext createWebApplicationContext(ServletContext sc) {
    Class<?> contextClass = determineContextClass(sc);
    if (!ConfigurableWebApplicationContext.class.isAssignableFrom(contextClass)) {
        throw new ApplicationContextException("Custom context class [" + contextClass.getName() +
                "] is not of type [" + ConfigurableWebApplicationContext.class.getName() + "]");
    }
    return (ConfigurableWebApplicationContext) BeanUtils.instantiateClass(contextClass);
}
```

这个方法主要用于创建一个WebApplicationContext对象，因为WebApplicationContext只是一个接口，不能创建对象，所以需要找到一个WebApplicationContext接口的实现类，determineContextClass方法就是用于寻找实现类，如果开发人员在web.xml中配置了一个参数名为contextClass，值为WebApplicationContext接口实现类，那就会返回这个配置的实现类Class：如果没有配置，则会返回Spring默认的实现类XmlWebApplicationContext。直接进入determineContextClass方法体，代码如下，

```
protected Class<?> determineContextClass(ServletContext servletContext) {
    String contextClassName = servletContext.getInitParameter(CONTEXT_CLASS_PARAM);
    if (contextClassName != null) {
        try {
            return ClassUtils.forName(contextClassName, ClassUtils.getDefaultClassLoader());
        }catch (ClassNotFoundException ex) {
            throw new ApplicationContextException(
                    "Failed to load custom context class [" + contextClassName + "]", ex);
        }
    }else {
        contextClassName = defaultStrategies.getProperty(WebApplicationContext.class.getName());
        try {
            return ClassUtils.forName(contextClassName, ContextLoader.class.getClassLoader());
        }catch (ClassNotFoundException ex) {
            throw new ApplicationContextException(
                    "Failed to load default context class [" + contextClassName + "]", ex);
        }
    }
}
```

上面的代码中使用了defaultStrategies，用于获取Spring默认的WebApplicationContext接口实现类XmlWebApplicationContext.java，它的声明如下，

```
private static final String DEFAULT_STRATEGIES_PATH = "ContextLoader.properties";
private static final Properties defaultStrategies;
static {
    try {
        ClassPathResource resource = new ClassPathResource(DEFAULT_STRATEGIES_PATH, ContextLoader.class);
        defaultStrategies = PropertiesLoaderUtils.loadProperties(resource);
    }catch (IOException ex) {
        throw new IllegalStateException("Could not load 'ContextLoader.properties': " + ex.getMessage());
    }
}
```

也就是说在ContextLoader.java的路径下，有一个ContextLoader.properties文件，查找并打开这个文件，文件内容如下。

org.springframework.web.context.WebApplicationContext=org.springframework.web.context.support.XmlWebApplicationContext

这里配置了Spring默认的WebApplicationContext接口实现类XmlWebApplicationContext.java。回到createWebApplicationContext方法，WebApplicationContext接口实现类的Class已经找到，然后就是使用构造函数进行初始化完成WebApplicationContext对象创建。

继续回到initWebApplicationContext方法，此时这个context就指向了刚刚创建的WebApplicationContext对象，因为XmlWebApplicationContext间接实现了ConfigurableWebApplicationContext接口，所以将会执行如下代码，

```
if (this.context instanceof ConfigurableWebApplicationContext) {
    ConfigurableWebApplicationContext cwac = (ConfigurableWebApplicationContext) this.context;
    if (!cwac.isActive()) {
        if (cwac.getParent() == null) {
            ApplicationContext parent = loadParentContext(servletContext);
            cwac.setParent(parent);
        }
        configureAndRefreshWebApplicationContext(cwac, servletContext);
    }
}
```

这里关注重点代码configureAndRefreshWebApplicationContext(cwac, servletContext)，configureAndRefreshWebApplicationContext方法具体代码如下，

```
protected void configureAndRefreshWebApplicationContext(ConfigurableWebApplicationContext wac, ServletContext sc) {
    if (ObjectUtils.identityToString(wac).equals(wac.getId())) {
        String idParam = sc.getInitParameter(CONTEXT_ID_PARAM);
        if (idParam != null) {
            wac.setId(idParam);
        }else {
            wac.setId(ConfigurableWebApplicationContext.APPLICATION_CONTEXT_ID_PREFIX +
                    ObjectUtils.getDisplayString(sc.getContextPath()));
        }
    }

    wac.setServletContext(sc);
    String configLocationParam = sc.getInitParameter(CONFIG_LOCATION_PARAM);
    if (configLocationParam != null) {
        wac.setConfigLocation(configLocationParam);
    }

    ConfigurableEnvironment env = wac.getEnvironment();
    if (env instanceof ConfigurableWebEnvironment) {
        ((ConfigurableWebEnvironment) env).initPropertySources(sc, null);
    }

    customizeContext(sc, wac);
    wac.refresh();
}
```

还是一样，关注重点代码，看一下CONFIG_LOCATION_PARAM这个常量的值是"contextConfigLocation"，OK，这个就是web.xml中配置applicationContext.xml的，这个参数如果没有配置，在XmlWebApplicationContext中是有默认值的，具体的值如下，

```
public static final String DEFAULT_CONFIG_LOCATION = "/WEB-INF/applicationContext.xml";
```

也就是说，如果没有配置contextConfigLocation参数，将会使用/WEB-INF/applicationContext.xml，关注configureAndRefreshWebApplicationContext方法的最后一行代码wac.refresh()，是不是有点眼熟，继续跟踪代码，refresh方法的具体代码如下，

```
public void refresh() throws BeansException, IllegalStateException {
    synchronized (this.startupShutdownMonitor) {
        prepareRefresh();
        ConfigurableListableBeanFactory beanFactory = obtainFreshBeanFactory();
        prepareBeanFactory(beanFactory);
        try {
            postProcessBeanFactory(beanFactory);
            invokeBeanFactoryPostProcessors(beanFactory);
            registerBeanPostProcessors(beanFactory);
            initMessageSource();
            initApplicationEventMulticaster();
            onRefresh();
            registerListeners();
            finishBeanFactoryInitialization(beanFactory);
            finishRefresh();
        }catch (BeansException ex) {
            logger.warn("Exception encountered during context initialization - cancelling refresh attempt", ex);
            destroyBeans();
            cancelRefresh(ex);
            throw ex;
        }
    }
}
```

这个refresh方法就是《Spring源码学习四： BeanDefinition装载前奏曲》介绍的那个refresh方法，用于完成Bean的解析、实例化及注册。

继续分析，回到initWebApplicationContext方法，将执行如下代码，

```
servletContext.setAttribute(WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE, this.context);
```

可以看到，这里将初始化后的context存到了servletContext中，具体的就是存到了一个Map变量中，key值就是WebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE这个常量。

使用Spring的WebApplicationContextUtils工具类获取这个WebApplicationContext方式如下，

```
WebApplicationContext applicationContext = WebApplicationContextUtils.getWebApplicationContext(request.getServletContext());
```