整合**ssm**项目我们到底在干嘛？

为什么要**tomcat**。

- tomcat起到了什么角色：所有的项目开发都脱离不web容器，这个web容器就是tomcat。其实不论你是jsp/servlet、ssh、ssm项目包括后面springboot项目，微服务项目都需要它。
- tomcat启动到底在做什么事情呢？
  - 运行tomcat自身，就会开启一个java虚拟机去运行tomcat自身。它是一个进程，内部实现线程池 + NIO运行模型。
  - 它就开始加载tomcat默认目录下的webapp下javaweb项目或者idea中部署的war包。把它们统统加载到jvm内存中进行运行
  - tomcat加载项目的时候又在做什么事情呢？
    - tomcat底层提供：servlet容器 ，所有的java应用程序开发都是面向tomcat编程。
    - tomcat加载项目过程如下：
      - 加载tomcat自身的web.xml
        - 这个文件为什么要加载：提供基础配置，开发的时候其实理论项目的web.xml是可以不用的。都应该配置到默认web.xml即可。那这样疯了，这样不利用项目开发，也破坏了扩展的能力。所以这个主web.xml提供一些基本的配置，比如：欢迎页，session过期时间，mine-

type，然后浏览能够识别内容的配置信息等等。然后用一种类似于继承的概念，让每个项目自身去扩展。从而达到了继承的目的。

- 加载项目web.xml
  - 专门用来配置项目的servlet和filter，listerner器等等。
- 也就是告诉你一个道理：你每天在不停的启动项目其实就是在加载你项目中的web.xml文件。

Tomcat加载web.xml在干嘛呢？(项目)

读取web.xml文件 开始加载：上下文，拦截器，监听器，servlet

上下文：

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring.xml</param-value>
</context-param>
```

1、application.setAttribute("contextConfigLocation","classpath:spring.xml")

servlet初始化上下文(application)有两种方式：

- 用web.xml的context-param来初始化和设置上下文的内容

- 用程序代码：
  request.getServletContxt().setAttribute(key,value);

监听器：

```
1   <listener>
2         <listener-
  class>org.springframework.web.context.ContextLoade
  rListener</listener-class>
3     </listener>
```

2：

监听器：是servlet提供去监听作用域的一种机制，分别监听：applicaiton，session，request .其中有一个监听器接口：ServletContextListener 专门来监听上下文。如果有实现类继承了ServletContextListener接口。tomcat的servlet容器，内部就调用执行方法contextInitialized(); 很明显的是：spring框架提供上下文的监听器恰好实现了ServletContextListener接口。在启动tomcat就开始去运行ContextLoaderListener.javav中的方法contextInitialized();

```
1  this.initWebApplicationContext(event.getServletCo
  ntext());
```

解析上下文提供的spring框架的核心配置文件：spring.xml ，在这个方法中你们就看到spring容器的初始化：

```
1 ApplicationContext context = new
  WebApplicationContext("classpath:spring.xml");
```

上面就是ioc容器的初始化，就开始去解析spring.xml文件，把里面所有的bean，全部解析出来，放入到ioc容器中，并且把项目中所有的需要依赖注入的bean，全部初始化，然后项目启动完毕。你就可以去访问你的项目了。这也就印证了你在学习spring的ioc的时候为什么要学习容器的初始化：

```
ApplicationContext context = new
ClassPathXmlApplicationContext("classpath:spring.x
ml");
```

开始加载spring.xml文件

```
<!-- 扫描基本包 过滤controller层 -->
<context:component-scan base-package="com.shsxt" >
    <context:exclude-filter type="annotation"
expression="org.springframework.stereotype.Control
ler" />
</context:component-scan>
```

1：上面是sprign3.x提供一个包扫描的机制 + 注解，它的作用就自动完成你指定的包下面的类如果加了注解就把它们加载ioc容器中去

2：如果传统的方式的就必须一个个配置和完成的映射关系，这个文件是非常庞大很臃肿。

3：上面代码，为什么要排除Controller注解呢？必须重复加载，因为springmvc和spring是一个父子容器的关系。controller这层不需要spring管理，而交由springmvc自身来管理。如果你擅自扫描进去可能会引发加载过程的冲突问题。

4：所以上面的diamante，其实是一种偷懒写法。但是也是一种一劳永逸的写法。但是其实建议大家需要扫描配置，但是往往在程序开发中，很多架构师，架构项目的时候并不是特别规范或者不清晰，所以就扫描整个包排除controller。为什么因为一个项目其实很多包和类其实不需要让spring去加载比如：common/util/config等，这些是不需要去加载到ioc容器，如果你配置是总目录。说明会增加类的排除和查找的工作。就好比：我本来就只要加载100类到ioc容器中就可以，但是我们要500个类去找100个，性能就会下降。

<aop:aspectj-autoproxy />

aop是一个思想：实现技术有两个：

aop思想：代理对象执行方法，进行横切面拦截，在拦截过程中我们可以注入和增强业务逻辑。从而达到增强业务的能力。

- jdk动态代理 spring自身实现动态代理完成aop的思想。
    - 面向接口编程
- cglib代理aspectj - spring用aspectj组织提供的，也完成的aop思想
    - 面向类编程

- spring的ioc是基础，如果没有ioc都不用谈了。aop的构建和依托ioc。

无论是使用jdk和cglib的方式，它都在做一件事情，就是把springioc中的初始化好的bean，全部变成代理对象，其中下面的这个配置：

```
<aop:aspectj-autoproxy />
```

- 传统的xml必须自己去开启和指定，默认指定是：jdk动态代理。底层做了切换处理
- springboot在2.x版本全部改成cglib代理。

过滤器

```
 <filter>
        <description>char encoding filter</description>
        <filter-name>encodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

servlet

```
<servlet>
        <servlet-name>springMvc</servlet-name>
        <servlet-
class>org.springframework.web.servlet.DispatcherS
ervlet</servlet-class>
        <init-param>
            <param-
name>contextConfigLocation</param-name>
            <param-value>classpath:servlet-
context.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
      <servlet-name>springMvc</servlet-name>
      <url-pattern>/</url-pattern>
    </servlet-mapping>
```

springboot为什么不需要web.xml

- spingboot内置了tomcat。
- 为什么不需要web.xml， 以为tomcat的servlet版本也在升级和更新，servelt4.0的api就已经可以不需要web.xml，通过面向对象方式启动tomcat。请看飞哥javaweb的课程。-- 自定义springmvc框架。

一句话：我们所有的javaweb开发都是面向tomcat的servlet开发。只不过springmvc ,strtus2进行了优化和处理。有了请求以后------数据就持久化，流动（curd） myabtis /hibernate/jpa/jdbc ----- 响应 freemaker/thymeleaf/jsp等。