**tomcat在初始化中做了什么事：**https://blog.csdn.net/yangsnow\_rain\_wind/article/details/80049506

Bootstrap中的main方法启动，调用init()方法，在initClassLoaders()方法中通过加载catalina.properties配置文件来初始化类加载器。

Bootstrap的load()中通过反射调用Catalina类中的load()方法，加载并解析server.xml，

初始化server实例，初始化service容器包括Engine、Executor、MapperListener、Connector并按照初始化顺序启动

**tomcat调优**：https://blog.51cto.com/8248183/2062343、https://www.cnblogs.com/zhuawang/p/5213192.html

Tomcat内存优化

Tomcat并发优化：使用线程池，设置合理的maxThreads，acceptAccount

Tomcat连接模式类型优化：关闭ajp端口、BIO模式、NIO模式、APR模式

启用Tomcat压缩

**spring AOP ：**

AOP是一种思想，本质是在一系列纵向的控制流程汇中，把相同的子流程提取成一个横向的面，使用java反射（动态代理技术）将这些子流程加入到主流程中。

spring AOP不会把代码加入到源文件中，但会影响最终的机器代码。

传统的OOP开发中的逻辑是自上而下，在这些自上而下的过程中会产生一些横切性的问题，这些横切性的问题和我们的主业务逻辑关系不大，会散落在到吗的各个地方，造成难以维护。AOP的编程思想就是把业务逻辑和横切问题进行分离，从而达到解耦的目的，是代码的重用性和开发效率提高。

AOP应用场景：

日志记录，权限验证，效率（性能）检查，事务管理等。

AOP底层技术（实现）：JDK动态代理和cglib。具体使用哪种技术看spring配置： @EnableAspectJAutoProxy(proxyTargetClass=false)。 默认为false

1. 如果为true：不管是否实现接口，都是用cglib。
2. 如果为false：如果实现了接口，使用JDK动态代理；如果没有实现接口，使用cglib。

Spring AOP 和Aspectj的关系：

Spring的代理类是在初始化的时候生成的，获取的时候从singletonObject（是一个concurrentHashMap，spring容器的核心）中根据bean name获取。

**spring IOC:**是一种机制，一种思想，使用这种机制的用例很多，实现方式也很多。

是spring的核心，对于spring框架来说就是由spring来负责控制对象的生命周期和对象的关系。bean 容器，管理bean的生命周期

**spring DI:**是IOC的一种具体用例的具体实现方式。

**spring有哪些组件：**7大模块

①Spring Core核心容器：提供spring框架的基本功能，实现了IOC模式。此模块中包含的BeanFactory类是spring的核心类，负责javaBean的配置与管理，采用了工厂模式实现了DI。

②Spring Context应用上下文：构建于 core 封装包基础上的 context 封装包，提供了一种框架式的对象访问方法。

③Spring AOP：提供了面向切面的编程实现，让你可以自定义拦截器、切点等。

④Spring DAO：Data Access Object 提供了JDBC的抽象层。

⑤Spring ORM：使用Hibernate框架

⑥Spring Web：提供了针对 Web 开发的集成特性，例如文件上传，利用 servlet listeners 进行 ioc 容器初始化和针对 Web 的 ApplicationContext。

⑦Spring MVC：spring 中的 mvc 封装包提供了 Web 应用的 Model-View-Controller（MVC）的实现。

**spring bean是单例吗：**

默认是单例的（并不保证是线程安全的），但是可以配置。xml配置中bean标签的scope属性singleton和prototype，注解配置使用@Scope。

**spring 支持几种 bean 的作用域：**

spring 支持 5 种作用域，如下：

①singleton：spring ioc 容器中只存在一个 bean 实例，bean 以单例模式存在，是系统默认值；

②prototype：每次从容器调用 bean 时都会创建一个新的示例，既每次 getBean()相当于执行 new Bean()操作；

Web 环境下的作用域：

③request：每次 http 请求都会创建一个 bean；

④session：同一个 http session 共享一个 bean 实例；

⑤global-session：用于 portlet 容器，因为每个 portlet 有单独的 session，globalsession 提供一个全局性的 http session。

注意： 使用 prototype 作用域需要慎重的思考，因为频繁创建和销毁 bean 会带来很大的性能开销。

**spring bean的生命周期：**

**spring单例bean是怎么实现的：**

**spring的BeanFactory和FactoryBean有什么区别：**https://www.cnblogs.com/aspirant/p/9082858.html

BeanFactory接口，负责生产和管理bean。spring提供的实现类有XMLBeanFactory，ApplicationContext，DefaultListenableBeanFactory

FactoryBean接口，是一个bean接口，用户可以通过实现该接口重写getobject()方法实现定制实例化逻辑。给bean的实现加上了一个工厂模式和装饰者模式。

**spring 事务实现方式有哪些：**

声明式事务：声明式事务也有两种实现方式，基于 xml 配置文件的方式和注解方式（在类上添加 @Transaction 注解）。

编码方式：提供编码的形式管理和维护事务。

**spring 的事务隔离：**

spring 有五大隔离级别，默认值为 ISOLATION\_DEFAULT（使用数据库的设置），其他四个隔离级别和数据库的隔离级别一致：

①ISOLATION\_DEFAULT：用底层数据库的设置隔离级别，数据库设置的是什么我就用什么；

②ISOLATIONREADUNCOMMITTED：未提交读，最低隔离级别、事务未提交前，就可被其他事务读取（会出现幻读、脏读、不可重复读）；

③ISOLATIONREADCOMMITTED：提交读，一个事务提交后才能被其他事务读取到（会造成幻读、不可重复读），SQL server 的默认级别；

④ISOLATIONREPEATABLEREAD：可重复读，保证多次读取同一个数据时，其值都和事务开始时候的内容是一致，禁止读取到别的事务未提交的数据（会造成幻读），MySQL 的默认级别；

⑤ISOLATION\_SERIALIZABLE：序列化，代价最高最可靠的隔离级别，该隔离级别能防止脏读、不可重复读、幻读。

脏读 ：表示一个事务能够读取另一个事务中还未提交的数据。比如，某个事务尝试插入记录 A，此时该事务还未提交，然后另一个事务尝试读取到了记录 A。

不可重复读 ：是指在一个事务内，多次读同一数据。

幻读 ：指同一个事务内多次查询返回的结果集不一样。比如同一个事务 A 第一次查询时候有 n 条记录，但是第二次同等条件下查询却有 n+1 条记录，这就好像产生了幻觉。发生幻读的原因也是另外一个事务新增或者删除或者修改了第一个事务结果集里面的数据，同一个记录的数据内容被修改了，所有数据行的记录就变多或者变少了。

**@RequestMapping的访问原理：**

将 http 请求映射到相应的类/方法上。

**如何实现跨域：**

实现跨域有以下几种方案：

服务器端运行跨域 设置 CORS 等于 \*；

在单个接口使用注解 @CrossOrigin 运行跨域；

使用 jsonp 跨域；

**JSONP 实现原理：**

jsonp：JSON with Padding，它是利用script标签的 src 连接可以访问不同源的特性，加载远程返回的“JS 函数”来执行的。

**spring MVC运行流程：**

①spring mvc 先将请求发送给 DispatcherServlet。

②DispatcherServlet 查询一个或多个 HandlerMapping，找到处理请求的 Controller。

③DispatcherServlet 再把请求提交到对应的 Controller。

④Controller 进行业务逻辑处理后，会返回一个ModelAndView。

⑤Dispathcher 查询一个或多个 ViewResolver 视图解析器，找到 ModelAndView 对象指定的视图对象。

⑥视图对象负责渲染返回给客户端。

**spring mvc 有哪些组件：**

前置控制器 DispatcherServlet。

映射控制器 HandlerMapping。

处理器 Controller。

模型和视图 ModelAndView。

视图解析器 ViewResolver。

**spring boot 核心配置文件：**

spring boot 核心的两个配置文件：

bootstrap (. yml 或者 . properties)：boostrap 由父 ApplicationContext 加载的，比 applicaton 优先加载，且 boostrap 里面的属性不能被覆盖；

application (. yml 或者 . properties)：用于 spring boot 项目的自动化配置。

**spring boot 有哪些方式可以实现热部署：**

使用 devtools 启动热部署，添加 devtools 库，在配置文件中把 spring. devtools. restart. enabled 设置为 true；

使用 Intellij Idea 编辑器，勾上自动编译或手动重新编译。

**spring cloud 的核心组件有哪些：**

Eureka：服务注册与发现。

Feign：基于动态代理机制，根据注解和选择的机器，拼接请求 url 地址，发起请求。

Ribbon：实现负载均衡，从一个服务的多台机器中选择一台。

Hystrix：提供线程池，不同的服务走不同的线程池，实现了不同服务调用的隔离，避免了服务雪崩的问题。

Zuul：网关管理，由 Zuul 网关转发请求给对应的服务。

**spring mvc 和 struts 的区别是什么：**

拦截级别：struts2 是类级别的拦截；spring mvc 是方法级别的拦截。

数据独立性：spring mvc 的方法之间基本上独立的，独享 request 和 response 数据，请求数据通过参数获取，处理结果通过 ModelMap 交回给框架，方法之间不共享变量；而 struts2 虽然方法之间也是独立的，但其所有 action 变量是共享的，这不会影响程序运行，却给我们编码和读程序时带来了一定的麻烦。

拦截机制：struts2 有以自己的 interceptor 机制，spring mvc 用的是独立的 aop 方式，这样导致struts2 的配置文件量比 spring mvc 大。

对 ajax 的支持：spring mvc 集成了ajax，所有 ajax 使用很方便，只需要一个注解 @ResponseBody 就可以实现了；而 struts2 一般需要安装插件或者自己写代码才行。

**拦截器和过滤器的区别：**

过滤器filter：

<filter>

<filter-name>encoding</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>

<init-param>

<param-name>forceEncoding</param-name>

<param-value>true</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encoding</filter-name>

<servlet-name>/\*</servlet-name>

</filter-mapping>

配置在web.xml中，例如设置字符编码。

范围：依赖于Servlet容器，基于函数回调的实现，几乎可以对所有请求进行拦截。

拦截器Interceptor：

<mvc:interceptors>

<mvc:interceptor>

<mvc:mapping path="/\*\*" />

<bean class="com.scorpios.atcrowdfunding.web.LoginInterceptor"></bean>

</mvc:interceptor>

<mvc:interceptor>

<mvc:mapping path="/\*\*" />

<bean class="com.scorpios.atcrowdfunding.web.AuthInterceptor"></bean>

</mvc:interceptor>

</mvc:interceptors>

配置在springmvc配置文件中。

范围：依赖于web框架，在springmvc中就依赖于springmvc框架，基于java反射机制，

属于aop的一种运用，动态代理就是拦截器的简单实现，只能对controller请求进行拦截。

**servlet生命周期：**

继承HTTPServlet类，servlet多线程单实例，service方法是多线程的，init，destory方法在一个servlet生命周期中只会执行一次。

**JSP 和 servlet 有什么区别：**

JSP 是 servlet 技术的扩展，本质上就是 servlet 的简易方式。servlet 和 JSP 最主要的不同点在于，servlet 的应用逻辑是在 Java 文件中，并且完全从表示层中的 html 里分离开来，而 JSP 的情况是 Java 和 html 可以组合成一个扩展名为 JSP 的文件。JSP 侧重于视图，servlet 主要用于控制逻辑。Tomcat获得jsp文件会将jsp转成java文件再编译成class文件。jsp转换后的类继承HttpJSPBase类，而HttpJspBase继承了HTTPServlet。

**JSP 内置对象及其作用：**https://www.cnblogs.com/whgk/p/6427759.html

JSP 有 9 大内置对象：

request：封装客户端的请求，其中包含来自 get 或 post 请求的参数；

response：封装服务器对客户端的响应；

pageContext：通过该对象可以获取其他对象；

session：封装用户会话的对象；

application：封装服务器运行环境的对象；

out：输出服务器响应的输出流对象；

config：Web 应用的配置对象；

page：JSP 页面本身（相当于 Java 程序中的 this）；

exception：封装页面抛出异常的对象。

**JSP 的 4 种作用域：**

page：代表与一个页面相关的对象和属性。

request：代表与客户端发出的一个请求相关的对象和属性。一个请求可能跨越多个页面，涉及多个 Web 组件；需要在页面显示的临时数据可以置于此作用域。

session：代表与某个用户与服务器建立的一次会话相关的对象和属性。跟某个用户相关的数据应该放在用户自己的 session 中。

application：代表与整个 Web 应用程序相关的对象和属性，它实质上是跨越整个 Web 应用程序，包括多个页面、请求和会话的一个全局作用域。

**session 和 cookie 有什么区别：**

存储位置不同：session 存储在服务器端；cookie 存储在浏览器端。

安全性不同：cookie 安全性一般，在浏览器存储，可以被伪造和修改。

容量和个数限制：cookie 有容量限制，每个站点下的 cookie 也有个数限制。

存储的多样性：session 可以存储在 Redis 中、数据库中、应用程序中；而 cookie 只能存储在浏览器中。