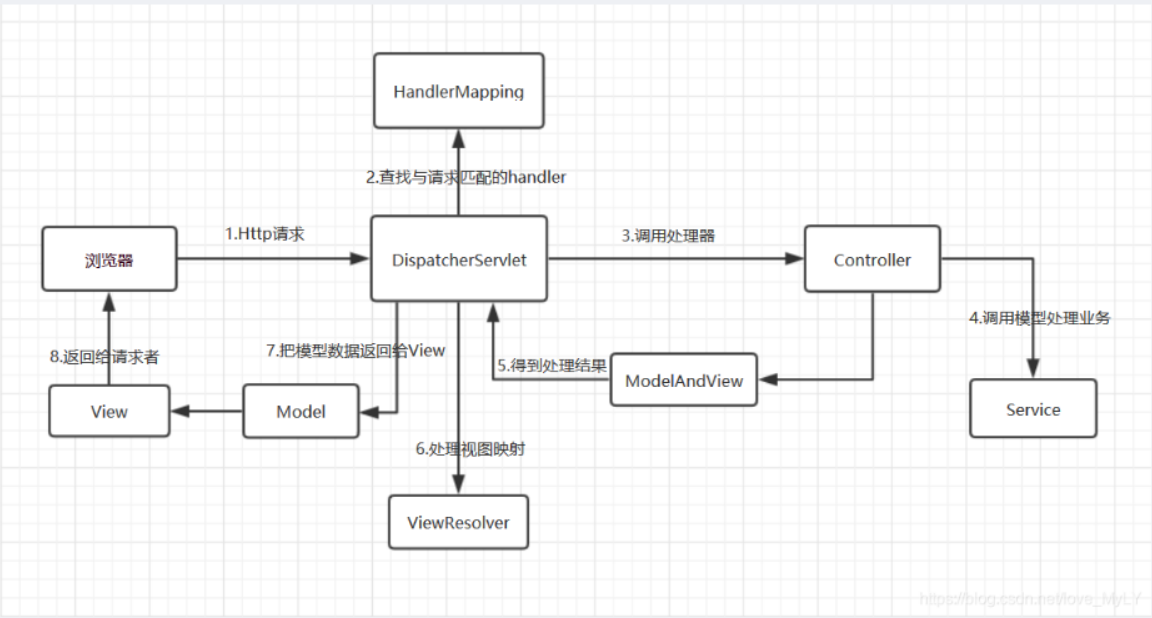
前端控制器

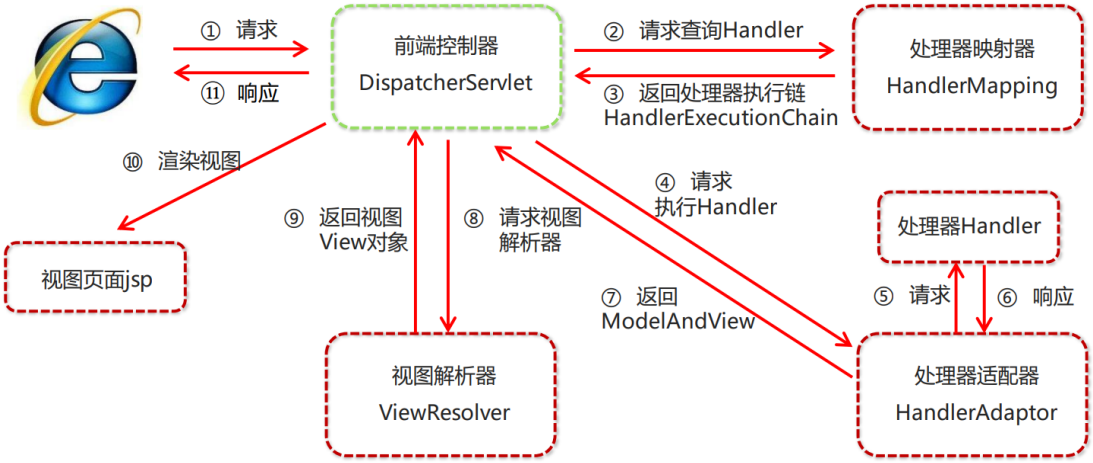
SpringMVC 原理，执行过程。

SpringMVC原理：SpringMVC以请求为驱动，围绕Servlet设计，将请求发给控制器，然后通过模型对象，分派器来展示请求结果视图。其中核心类是DispatcherServlet，它是一个Servlet，顶层是实现的Servlet接口。

执行过程：



1. 客户端（浏览器）发送请求，请求提交到DispatcherServlet。
2. DispatcherServlet调用HandlerMapping查询请求信息，找到对应的Controller。
3. DispatcherServlet调用对应Controller
4. Controller会根据请求信息来调用Service，Service会处理相应的业务逻辑。
5. Service处理完业务后，会返回一个ModelAndView对象，Model是返回的数据对象，View是个逻辑上的View。
6. DispatcherServlet调用ViewResolver，ViewResolver 会根据逻辑View查找实际的View。
7. DispaterServlet把返回的Model传给View。
8. 通过View返回处理结果给请求者客户端（浏览器）并显示

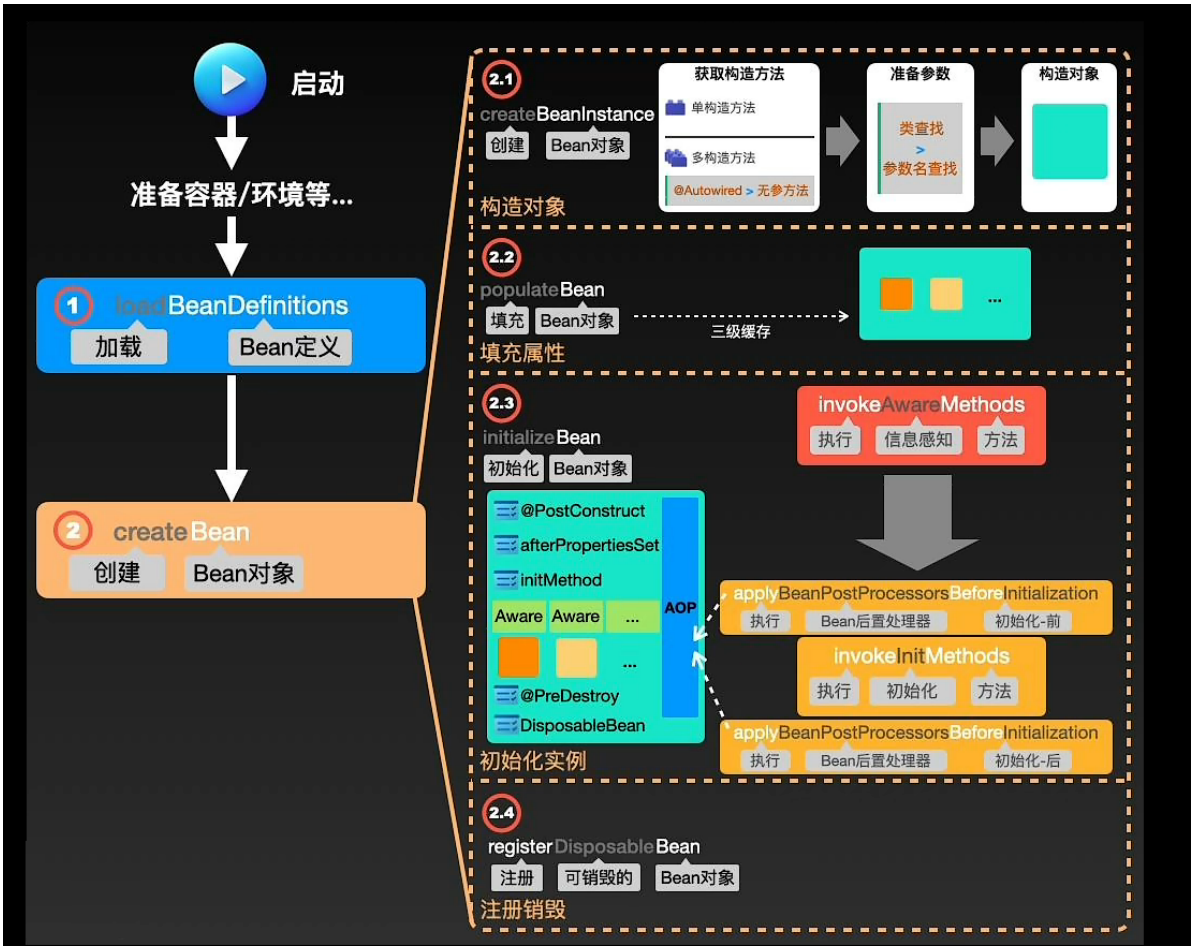


- 每一个过滤器都必须指定一个int类型的order值，**order值越小，优先级越高，执行顺序越靠前。**
- GlobalFilter通过实现Ordered接口，或者添加@Order注解来指定order值，由我们自己指定
- 路由过滤器和defaultFilter的order由Spring指定，默认是按照声明顺序从1递增。
- 当过滤器的order值一样时，会按照 defaultFilter > 路由过滤器 > GlobalFilter的顺序执行。

路由断言（predicates）：判断路由的规则，

Bean生命周期

创建Bean对象



通过.“加载Bean定义”，loadBeanDefinitions方法 进行扫描 定义的bean

放入 beanDefinitionMap 集合当中,

遍历 beanDefinitionMap 集合

create Bean 创建 Bean对象

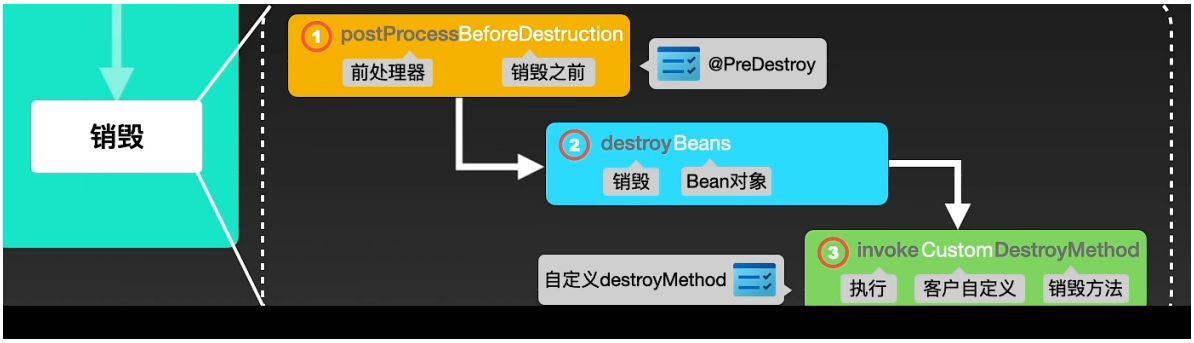
1. 构造对象：使用createBeanInstance方法进行对象的构造
    - 获取构造方法：单构造直接注入 | 多构造注入@Autuword
    - 准备构造方法需要的参数： 在单例池中 根据参数的Class类进行查找 ， 多个实例再根据参数名进行匹配，没有报错
    - 构造对象：进行对bean对象
2. 属性填充：通过populateBean方法为Bean内部的所需的属性进行赋值填充
    - 通过 "三级缓存" 机制进行填充，也就是"依赖注入"，比如@Autowired等
3. 初始化实例：通过initializeBean方法初始化实例
    - 初始化的第一步是初始化容器相关信息

- 通过invokeAwareMethods方法为实现了各种Aware接口的Bean设置容器信息，如beanName、beanFactory
- 通过invokeInitMethods方法执行Bean的初始化，可以通过实现InitalizingBean接口的afterPropertiesSet方法 【Bean属性填充后执行】
- 然后执行 定义@Bean(initMethod = "") 的init方法 @PostConstruct
- 在执行初始化方法之前和之后 还需要对 "Bean后置处理器" BeanPostProcessors进行处理
- 初始化之前和之后的各种Bean的后置处理器，Spring提供的AOP处理，负责 构造后@PostConstruct 和 销毁前 @PreDestroy处理
- 再或者 通过实现 BeanPostProcessor接口的 自定义处理器
- 可以通过实现 PriorityOrdered几口指定顺序

4. 注册销毁：上面步骤之后Bean就可以使用了，为了让Bean优雅的销毁
    - 通过注册销毁registerDisposableBean方法
    - 将实现了销毁接口 DisposableBean的Bean进行注册
    - 销毁时就可以执行destroy方法了。

将这些完整的Bean对象通过addSingleton方法 放入单例池 singletonObjects中就可以被获取和使用了

销毁：



销毁之前要先执行 "销毁前处理器" postProcessBeforeDestruction

也就是执行 在Bean中加了 @PreDestroy注解的方法了

然后通过destroyBeans方法逐一 销毁 Bean

@Bean(destroyMethod = "")

三级缓存

```
AService的bean生命周期

0. creatingSet[AService]
1. 实例化--->AService的普通对象---->singletonFactories<beanName, () -> getEarlyBeanReference(beanName, mbd, bean)>
2. 填充bService--->单例池--->创建BService

    BService的bean生命周期
    2.1. 实例化--->BService的普通对象
    2.2. 填充aService-->单例池-->createingSet--->循环依赖--->earlySingletonObjects--->singletonFactories--->AOP--->AService的代理对象--->earlySingletonObjects
    2.3. 填充其他属性
    2.4. 其他步骤（包括AOP）
    2.5. 加入到单例池中

2. 填充cService--->单例池--->创建CService

    CService的bean生命周期
    2.1. 实例化--->CService的普通对象
    2.2. 填充aService-->单例池-->createingSet--->循环依赖--->earlySingletonObjects
    2.4. 其他步骤（包括AOP）
    2.5. 加入到单例池中



3. 填充其他属性
4. 其他步骤（包括AOP）--->AService代理对象
5. 加入到单例池中
```

```
@After("execution(* com.kid..service.*.*(..))")
```