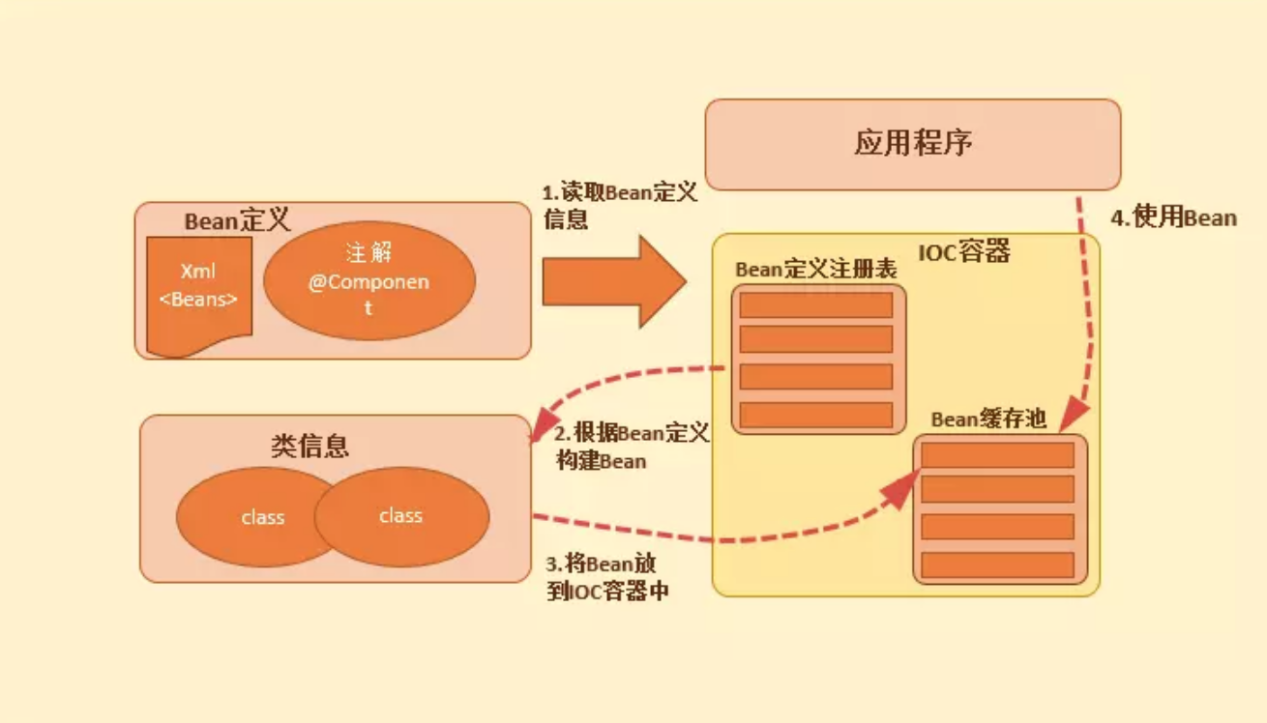
# springIOC

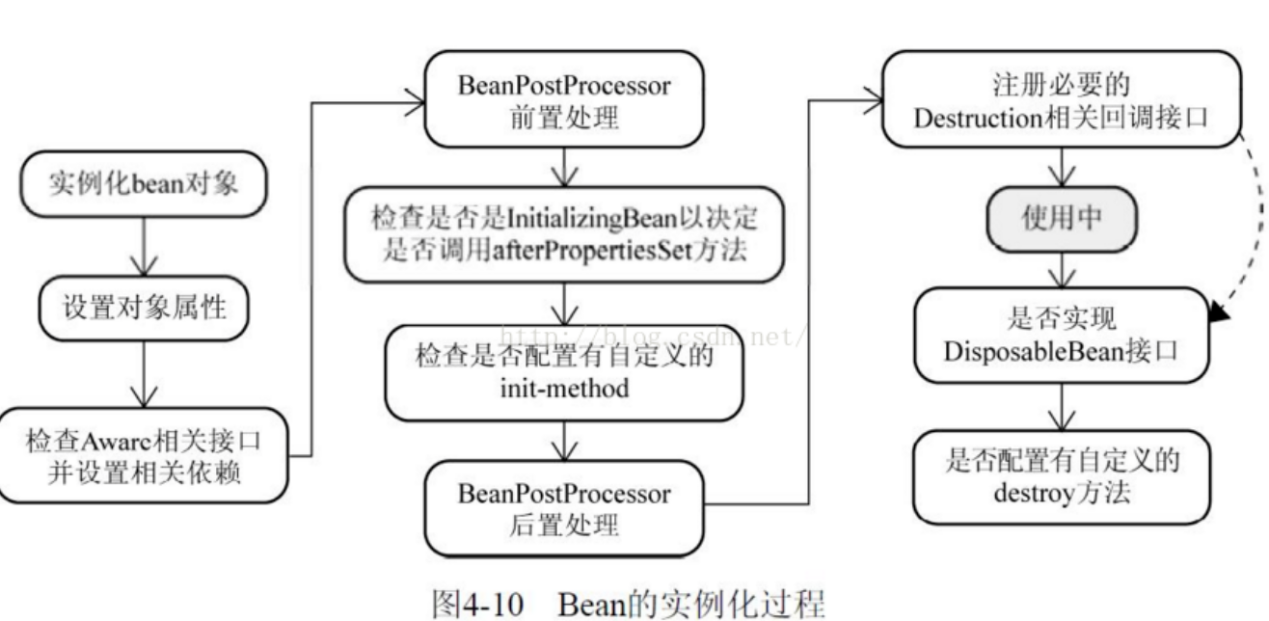
## IOC是什么

将设计好的对象交由spring容器进行控制。换句话说就是IOC控制了对象的生成，从而控制了外部资源的获取(主要是对象，甚至是文件)，最终在我们需要使用的时候，容器帮我们查找所需的对象。

## IOC容器示意图



## Spring bean生命周期



1. Spring对bean进行实例化，默认bean是单例；

2. Spring对bean进行依赖注入；

3. 如果bean实现了BeanNameAware接口，spring将bean的id传给setBeanName()方法；

4. 如果bean实现了BeanFactoryAware接口，spring将调用setBeanFactory方法，将BeanFactory实例传进来；

5. 如果bean实现了ApplicationContextAware接口，它的setApplicationContext()方法将被调用，将应用上下文的引用传入到bean中；

6. 如果bean实现了BeanPostProcessor接口，它的postProcessBeforeInitialization方法将被调用；

7. 如果bean实现了InitializingBean接口，spring将调用它的afterPropertiesSet接口方法，类似的如果bean使用了init-method属性声明了初始化方法，该方法也会被调用；

8. 如果bean实现了BeanPostProcessor接口，它的postProcessAfterInitialization接口方法将被调用；

9. 此时bean已经准备就绪，可以被应用程序使用了，他们将一直驻留在应用上下文中，直到该应用上下文被销毁；

10. 若bean实现了DisposableBean接口，spring将调用它的distroy()接口方法。同样的，如果bean使用了destroy-method属性声明了销毁方法，则该方法被调用；

## bean的作用域

1. **Singleton，单例**

Spring 容器里只有一个共享的bean实例，并且所有对bean的请求，只要id与该bean定义相匹配，则只会返回bean的同一实例。

1. **protoType, 多个实例**

表示一个bean定义对应多个对象实例，在创建容器的时候并没有实例化，而是当获取bean的时候才会去创建一个对象，而且每次获取到的对象都不是同一个对象。

1. **Request**

当一个bean的作用域为Request，表示在一次HTTP请求中，一个bean定义对应一个实例；即每个HTTP请求都会有各自的bean实例。

<bean id="loginAction" class=cn.csdn.LoginAction" scope="request"/>

针对每次HTTP请求，Spring容器会根据loginAction bean的定义创建一个全新的LoginAction bean实例，且该loginAction bean实例仅在当前HTTP request内有效，因此可以根据需要放心的更改所建实例的内部状态，而其他请求中根据loginAction bean定义创建的实例，将不会看到这些特定于某个请求的状态变化。当处理请求结束，request作用域的bean实例将被销毁。

1. **Session**

当一个bean的作用域为Session，表示在一个HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。考虑下面bean定义：

<bean id="userPreferences" class="com.foo.UserPreferences" scope="session"/>

针对某个HTTP Session，Spring容器会根据userPreferences bean定义创建一个全新的userPreferences bean实例，且该userPreferences bean仅在当前HTTP Session内有效。与request作用域一样，可以根据需要放心的更改所创建实例的内部状态，而别的HTTP Session中根据userPreferences创建的实例，将不会看到这些特定于某个HTTP Session的状态变化。当HTTP Session最终被废弃的时候，在该HTTP Session作用域内的bean也会被废弃掉。

5. **global session**

当一个bean的作用域为Global Session，表示在一个全局的HTTP Session中，一个bean定义对应一个实例。典型情况下，仅在使用portlet context的时候有效。该作用域仅在基于web的Spring ApplicationContext情形下有效。考虑下面bean定义：

<bean id="user" class="com.foo.Preferences "scope="globalSession"/>

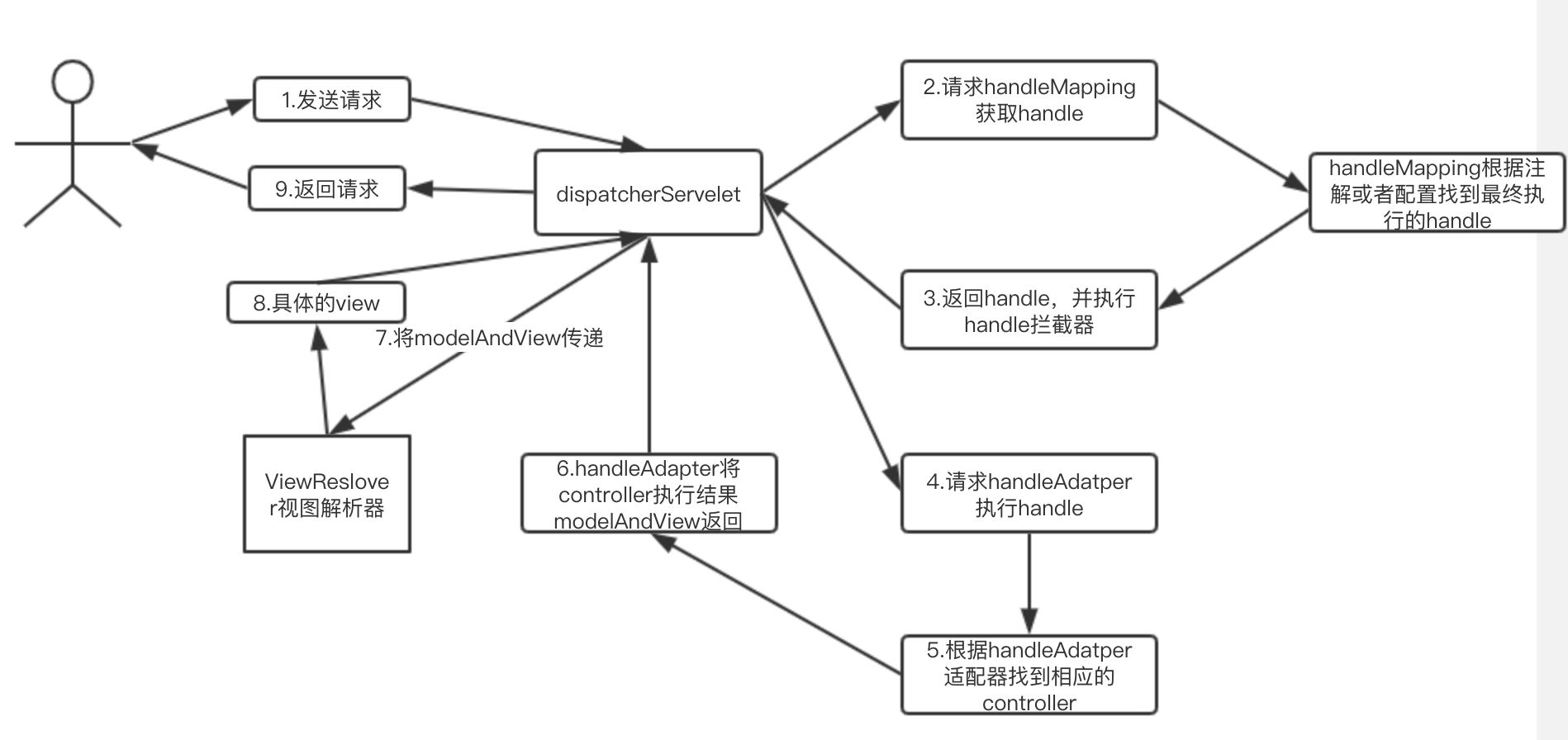
　　global session作用域类似于标准的HTTP Session作用域，不过仅仅在基于portlet的web应用中才有意义。Portlet规范定义了全局Session的概念，它被所有构成某个portlet web应用的各种不同的portlet所共享。在global session作用域中定义的bean被限定于全局portlet Session的生命周期范围内。

# springAOP

springBoot+AspectJ框架

# springMVC

## 工作原理



1. 用户发送请求至前端控制器DispatcherServlet

2. DispatcherServlet收到请求调用HandlerMapping处理器映射器

3. 处理器映射器找到具体的处理器(可以根据xml配置、注解进行查找)，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet

4. DispatcherServlet调用HandlerAdapter处理器适配器

5. HandlerAdapter经过适配调用具体的处理器(Controller，也叫后端控制器)

6. Controller执行完成返回ModelAndView

7. HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet

8. DispatcherServlet将ModelAndView传给ViewReslover视图解析器

9. ViewReslover解析后返回具体View

10. DispatcherServlet根据View进行渲染视图（即将模型数据填充至视图中）

11. DispatcherServlet响应用户

## 过滤器(Filter)

### Filter是如何进行拦截的

Filter接口中有一个doFilter的方法，这个方法实现了对用户请求的过滤。具体流程：

1. 用户发送请求到web服务器，请求会先到过滤器
2. 过滤器会对请求进行一些处理比如过滤请求的参数，判断是否可以让用户进行访问。
3. 用户请求相应完成
4. 进行一些自定义的处理

## 实现过滤器的两种方式

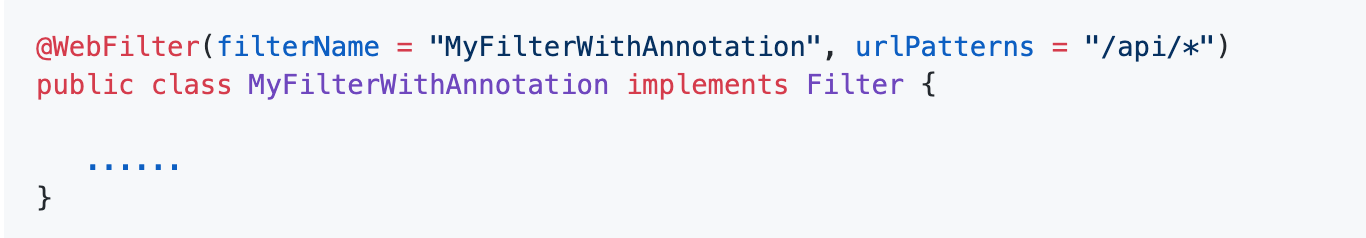
### 手动注册配置实现

自定义的 Filter 需要实现javax.Servlet.Filter接口，并重写接口中定义的3个方法。

在配置中注册自定义的过滤器。

### 使用注解的方式实现

在自己的过滤器的类上加上@WebFilter 然后在这个注解中通过它提供好的一些参数进行配置。



另外，为了能让 Spring 找到它，你需要在启动类上加上 @ServletComponentScan 注解。

如果有多个过滤器，通过@Order来进行配置

## 拦截器

当请求发送到controller时，在被controller处理之前，必须经过Interceptors。

1. 通过继承HandlerInterceptorAdapter，并实现里面的3个方法：prehandle、posthandle、afterCompletion。

当preHandle方法全为true时，执行下一个拦截器,直到所有拦截器执行完。再运行被拦截的Controller。然后进入拦截器链，运行所有拦截器的postHandle方法,完后从最后一个拦截器往回执行所有拦截器的afterCompletion方法

当preHandle方法返回false时，从当前拦截器往回执行所有拦截器的afterCompletion方法，再退出拦截器链。也就是说，请求不继续往下传了，直接沿着来的链往回跑。

1. 通过实现WebMvcConfigurer来进行实现拦截器的配置，在配置类中，可以自定义拦截器的执行熟悉，以及业务关联。

## 拦截器与过滤器的区别

1.两者都是AOP编程思想的实现，都能够实现权限控制和日志记录等问题的处理，但是两者粒度不同拦截对象不一样

2.适用范围不同：Filter是servlet的规范，只能用于web程序，但是拦截器可以用于application等程序。

3.规范不同：Filter是servlet的规范。但是Interceptor是spring容器支撑，有spring框架支持。

4. 使用资源不一样：spring的拦截器由于依赖spring，也是spring的一个组件，因此能够在拦截器中使用spring的任何资源和对象。例如service对象，数据源，事务管理等，通过ioc注入拦截器即可，而filter不能

5. 粒度不同：Filter只能在servlet的前后起作用，而拦截器能在方法前后异常前后执行，更加灵活，粒度更小，spring框架程序优先使用拦截器。

