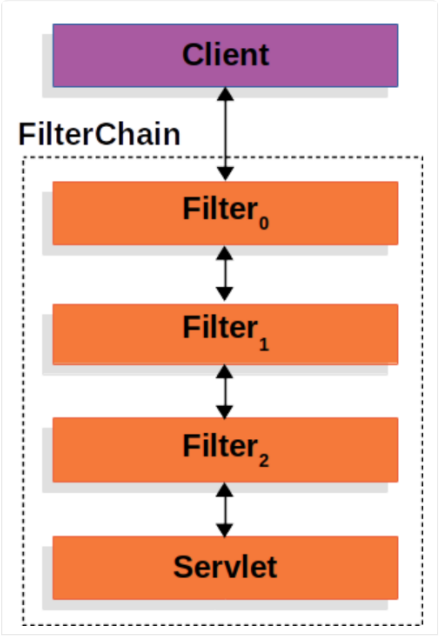
过滤器与 Security

> Spring Security 的 Servlet 支持基于 Servlet Filters

一、 单个 HTTP 请求的处理程序



单个 HTTP 请求的处理程序的典型分层。

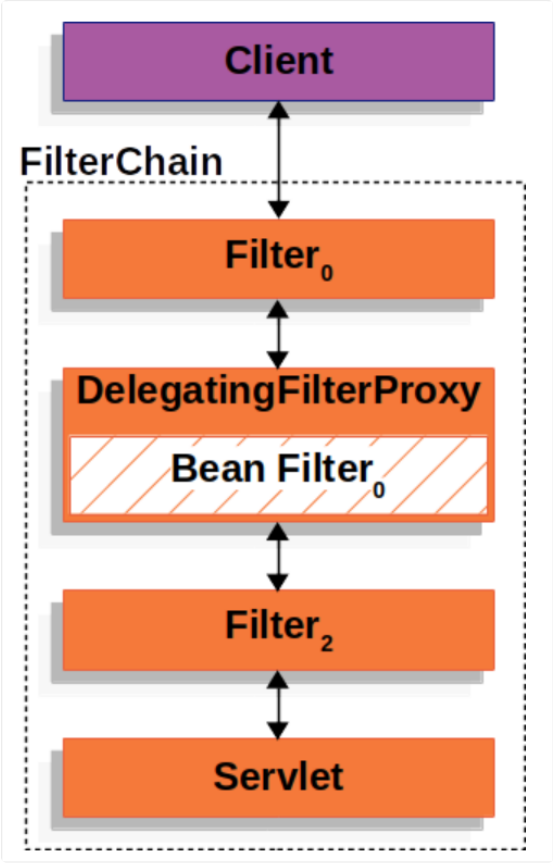
客户端向应用程序发送了一个请求，容器会创建一个 FilterChain 包含了根据请求的 URI 路径来处理 HttpServletRequest 的 Filter 和 Servlet

在 Spring MVC 应用程序中，Servlet 是 DispatcherServlet 的一个实例，最多可以使用一个 Servlet 来处理 HttpServletRequest 和 HttpServletResponse，但是可以使用多个 Filter 来处理请求和响应

过滤器只影响下游过滤器和 Servlet

二、 过滤器委派代理 DelegatingFilterProxy

Spring 提供了一个名为 DelegatingFilterProxy 的过滤器实现，它允许在 Servlet 容器的生命周期和 Spring 的 ApplicationContext 之间进行桥接。 Servlet 容器允许使用自己的标准注册过滤器，但不知道 Spring 定义的 Bean。 DelegatingFilterProxy 可以通过标准的 Servlet 容器机制注册，但将所有工作委托给实现 Filter 的 Spring Bean



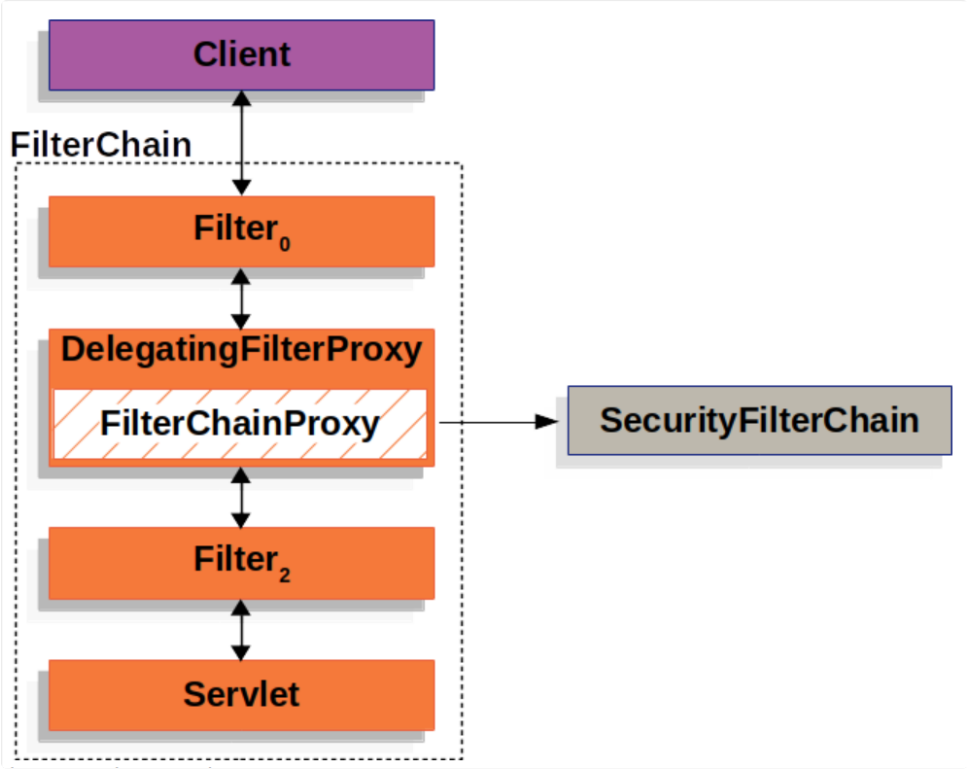
DelegatingFilterProxy 从 ApplicationContext 中查找 Bean Filter0，然后调用 Bean Filter0。

```java
public void doFilter(ServletRequest request, ServletResponse response, Filt
    // 懒加载Spring管理的过滤器Bean
    // 获取上图的的Filter0实例
    Filter delegate = getFilterBean(someBeanName);
    delegate.doFilter(request, response);
}
```

DelegatingFilterProxy 的另一个好处是它允许延迟查找 Filter bean 实例。这很重要，因为容器需要在容器启动之前注册过滤器实例。但是，Spring 通常使用 ContextLoaderListener 来加载 Spring Bean

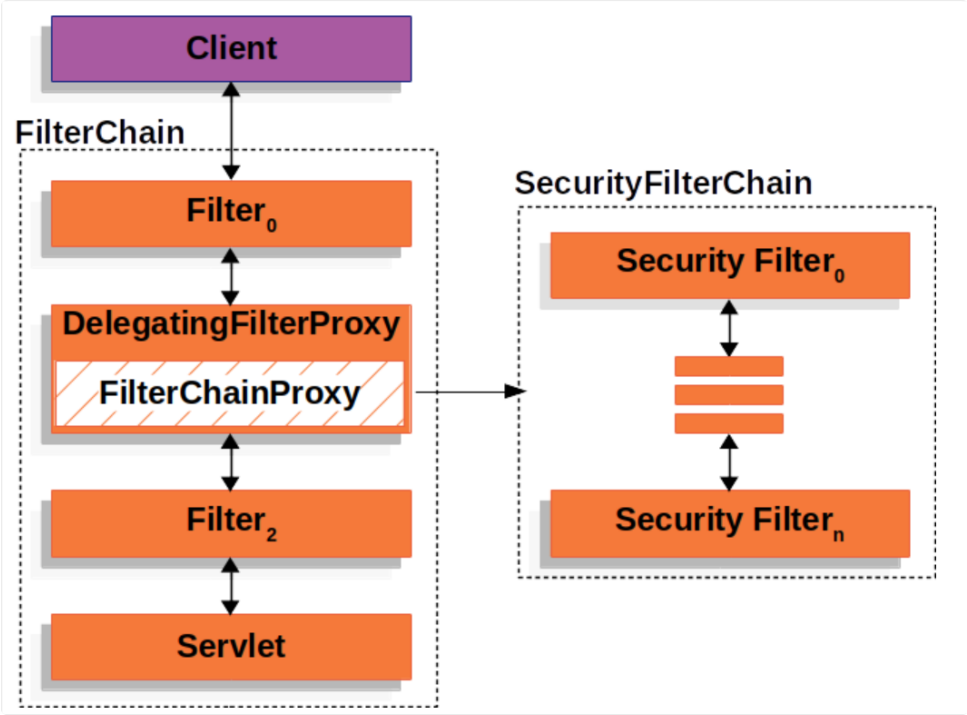
三、过滤器链代码 FilterChainProxy

Spring Security 对 Servlet 的兼容是通过 FilterChainProxy 来实现的。FilterChainProxy 是 Spring Security 提供的一个特殊 Filter，它允许通过 SecurityFilterChain 委托给多个 Filter 实例。由于 FilterChainProxy 是一个 Bean，它通常被包装在一个 DelegatingFilterProxy 中



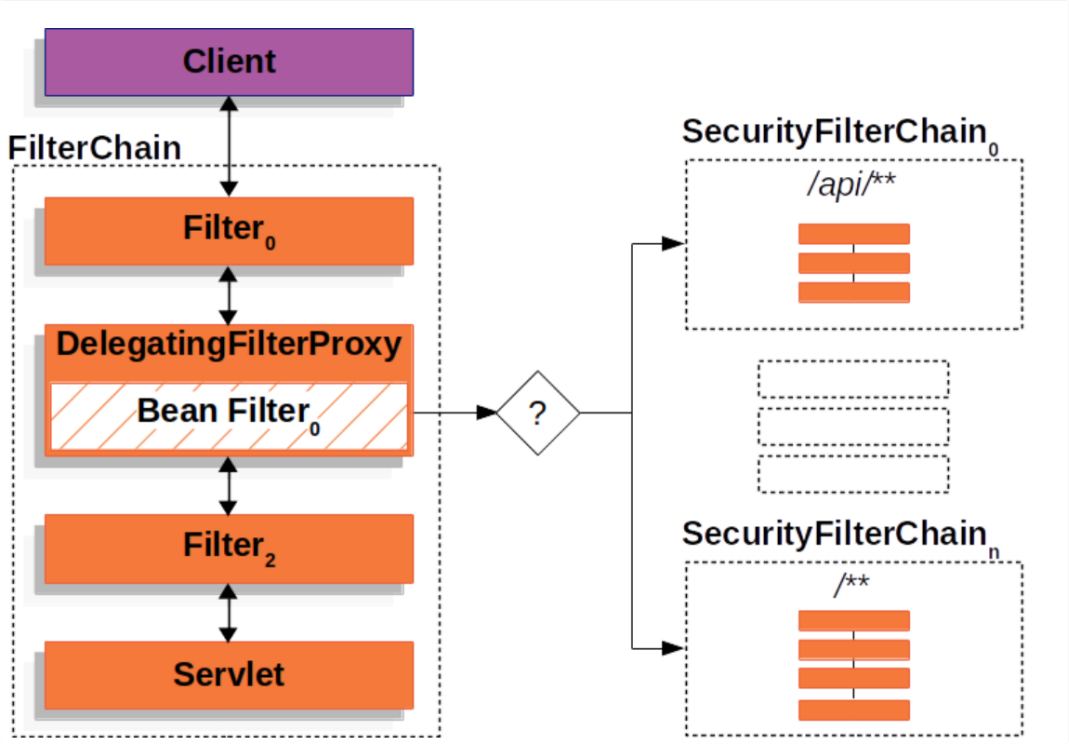
四、Security 过滤器链 SecurityFilterChain

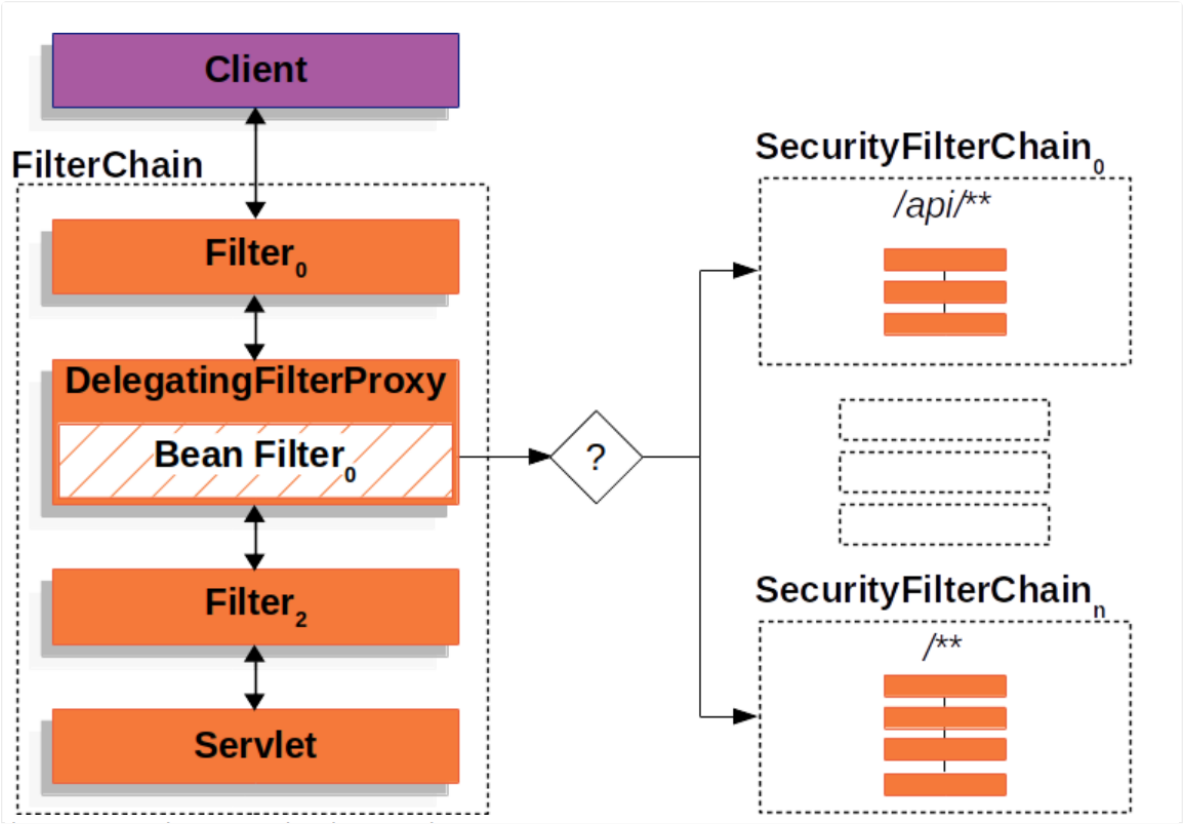
SecurityFilterChain 由 FilterChainProxy 使用，以确定该请求应调用哪些 Spring Security Filter



- Spring Security 的过滤器通常是 Spring 容器管理的 Bean, 通常是由 FilterChainProxy 来注册的而不是 DelegatingFilterProxy 来注册的。与直接使用 Servlet 容器或者 DelegtingFilterProxy 来注册过滤器相比，FilterChainProxy 注册过滤器有许多的好处。

1. 它为 Spring Security 的所有 Servlet 支持提供了一个起点。出于这个原因，如果您尝试对 Spring Security 的 Servlet 支持进行故障排除，在 FilterChainProxy 中添加一个调试点是一个很好的起点。
2. 由于 FilterChainProxy 是 Spring Security 使用的核心，它可以执行一些非强制性的任务。例如，它清除 SecurityContext 以避免内存泄漏。它还应用 Spring Security 的 HttpFirewall 来保护应用程序免受某些类型的攻击
3. 此外，它在确定何时调用 SecurityFilterChain 会更为灵活。在 Servlet 容器中，过滤器仅根据 URL 被调用。然而，FilterChainProxy 可以通过利用 RequestMatcher 接口，根据 HttpServletRequest 中的任何内容确定调用
4. FilterChainProxy 可以用来确定应该使用哪个 SecurityFilterChain。如果您的应用程序，这允许为不同的模块提供不同的配置





多个 Spring Security Filter chain

在上图展示的多个过滤器链中，会根据过滤器链的匹配顺序进行匹配，当一个 Spring 安全过滤器链被命中时其他的过滤器链都不会被匹配，当所有过滤器链都没有被成功匹配时会匹配 SecurityFilterChain `n`

每个 SecurityFilterChain 都有自己单独的过滤器配置，如果有需要可以实现一个没有任何过滤器的过滤器链来实现放行某些请求

五、Security 过滤器 Security Filters

> 安全过滤器是通过 SecurityFilterChain API 插入 FilterChainProxy 中的。过滤器的顺序很重要。通常没有必要知道 Spring Security 的 Filter 的顺序。

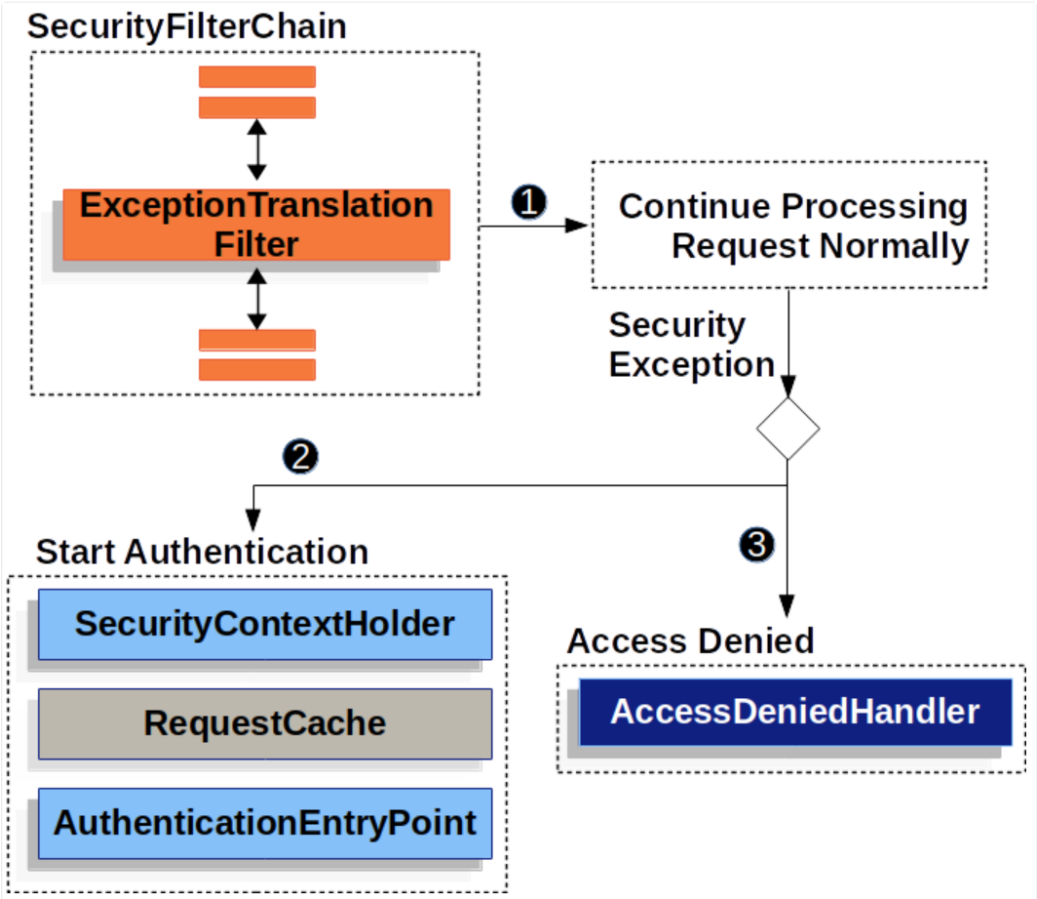
- Spring Security Filter 排序的完整列表
  - ChannelProcessingFilter
  - ConcurrentSessionFilter 并发会话过滤器
  - WebAsyncManagerIntegrationFilter
  - SecurityContextPersistenceFilter

- HeaderWriterFilter
- CorsFilter
- CsrfFilter
- LogoutFilter
- OAuth2AuthorizationRequestRedirectFilter
- Saml2WebSsoAuthenticationRequestFilter
- X509AuthenticationFilter
- AbstractPreAuthenticatedProcessingFilter
- CasAuthenticationFilter
- OAuth2LoginAuthenticationFilter
- Saml2WebSsoAuthenticationFilter
- UsernamePasswordAuthenticationFilter
- ConcurrentSessionFilter
- OpenIDAuthenticationFilter
- DefaultLoginPageGeneratingFilter
- DefaultLogoutPageGeneratingFilter
- DigestAuthenticationFilter
- BearerTokenAuthenticationFilter
- BasicAuthenticationFilter 基本身份验证过滤器
- RequestCacheAwareFilter
- SecurityContextHolderAwareRequestFilter
- JaasApiIntegrationFilter
- RememberMeAuthenticationFilter
- AnonymousAuthenticationFilter 匿名用户身份验证过滤器
- OAuth2AuthorizationCodeGrantFilter

- SessionManagementFilter 会话管理过滤器
- ExceptionTranslationFilter 异常处理过滤器
- FilterSecurityInterceptor 过滤器安全拦截器
- SwitchUserFilter 用户切换过滤器

六、 Security 异常处理 Handling Security Exceptions

在 ExceptionTranslationFilter 中处理授权和认证的异常，这个过滤器会将 AccessDeniedException and AuthenticationException 翻译为 HTTP 响应结果



1. 首先 ExceptionTranslationFilter 通过调用 FilterChain 的 doFilter(request, reponse)方法来调用下游的过滤器以及 Servlet。
2. 如果用户没有认真或者抛出了 AuthenticationException 异常，就会开始进入认证流程

a. 首先会清理 SecurityContextHolder

b. HttpServletRequest 保存在 RequestCache 中。当用户认证成功后，使用