1.Filter 的基本功能：

A.对 Servlet 容器调用 Servlet 的过程进行拦截，从而在 Servlet 进行响应处理的前后实现一些特殊的功能。

B.Filter 程序可以拦截 Jsp，静态图片文件和静态 html 文件。

2.在 Servlet API 中定义了三个接口类来开供开发人员编写 Filter 程序：Filter, FilterChain, FilterConfig

3.**Filter 程序是一个实现了 Filter 接口的 Java 类**，与 Servlet 程序相似，**它由 Servlet 容器进行调用和执行；**

4.和 Servlet 的结构类似，Filter 也有 init()方法和 destroy()方法，不同的是：

A.提供服务的方法名不同，Servlet 使用 **service()提供服务**，Filter 使用 **doFilter()方法提供服务；**

B.方法参数不同：

B1.init()方法中参数对象不同，Servlet 的是 ServletConfig 对象，Filter 的是 FilterConfig 对象；

B2.服务方法参数不同，

Servlet 的请求和响应类型是 **HttpServletRequest 和 HttpServletResponse；**

Filter 的参数类型是 **ServletRequest , ServletResponse , FilterChain ；**

B3.类继承或实现不同：

```
public class FromServlet extends HttpServlet
public class EncodingFilter implements Filter
```

5.Filter.doFilter()与 chain.doFilter()的区别：

A.Filter.doFilter()表示当前的过滤器的处理方法，可包含 chain.doFilter()，不包含时是**拦截原始请求**，包含时是**传递原始请求。**

B.chain.doFilter(),**chain 代表当前 Filter 链的对象，**在特定的操作完成后，可以在当前 Filter 对象的 doFilter 方法内部需要调用 FilterChain 对象的 chain.doFilter(request,response)方法才能**把请求交付给 Filter 链中的下一个 Filter 或者目标 Servlet 程序去处理，也可以直接向客户端返回响应信息**，或者利用 RequestDispatcher 的 forward()和 include()方法，以及 HttpServletResponse 的 sendRedirect()方法将请求转向到其他资源。

调用该方法将使过滤器链中的下一个过滤器被调用。如果是最后一个过滤器，会调用目标资源。

6.Filter 和 Servlet 的服务方法参数对象关系：

```
public abstract interface HttpServletRequest extends ServletRequest
public abstract interface HttpServletResponse extends ServletResponse
```

因此在 Filter 的服务方法中，若需要使用 HttpSession 等对象，可以将 **ServletRequest 和 ServletResponse 向下转型！**

7.FilterConfig 接口：该接口类似于 ServletConfig 接口，由容器实现。

Servlet 规范将代表 ServletContext 对象和 Filter 的配置参数信息都封装在该对象中。Servlet 容器将其作为参数传入过滤器对象的 init()方法中。

- String getFilterName()：得到描述符中指定的过滤器的名字。
- String getInitParameter(String name)： 返回在部署描述中指定的名字为 name 的初始化参数的值。如果不存在返回 null.

- Enumeration getInitParameterNames()：返回过滤器的所有初始化参数的名字的枚举集合。
- public ServletContext getServletContext()：返回 Servlet 上下文对象的引用。

8.过滤器的关键字：拦截--链--过滤

拦截 ----表明过滤器的执行要先于 Servlet，如果不先于，则做不到拦截；

链 ----表明过滤器可以是多个串成一条过滤链；

过滤 ----就是实现过滤器要实现的功能。

每一个过滤器都可以理解成在执行 Servlet 前的一道关卡，对发来的请求进行审核，这个请求是广义上的请求，即<dispatcher> 子元素可以设置的值

9.<dispatcher> 子元素可以设置的值及其意义：

–**REQUEST**：当用户直接访问页面时，Web 容器将会调用过滤器。

如果目标资源是通过 RequestDispatcher 的 include()或 forward()方法访问时，那么该过滤器就不会被调用。

–**INCLUDE**：如果目标资源是通过 RequestDispatcher 的 include()方法访问时，那么该过滤器将被调用。除此之外，该过滤器不会被调用。

–**FORWARD**：如果目标资源是通过 RequestDispatcher 的 forward()方法访问时，那么该过滤器将被调用，除此之外，该过滤器不会被调用。

–**ERROR：如果目标资源是通过声明式异常处理机制调用时，那么该过滤器将被调用。除此之外，过滤器不会被调用。**

10.和 Servlet 一样要使用 Filter，在创建完成后需要配置，配置方式有两种：

A.XML 的传统配置方式

B.注解的新式配置方式。

11.XML 的传统配置方式:

在 web.xml 文件中进行注册和设置它所能拦截的资源：若有多个 Filter 程序对某个 Servlet 程序的访问过程进行拦截，当针对该 Servlet 的访问请求到达时，web 容器将把这多个 Filter 程序组合成一个 **Filter 链**(过滤器链)。

注册与设置可以通过**<filter>和<filter-mapping>**元素来完成的。

<filter>元素用于在 Web 应用程序中注册一个过滤器。

在<filter>元素内

–<filter-name>用于为过滤器指定一个名字，该元素的内容不能为空。

–<filter-class>元素用于指定过滤器的完整的限定类名。

–<init-param>元素用于为过滤器指定初始化参数，它的子元素<param-name>指定参数的名字， <param-value>指定参数的值。在过滤器中，可以使用 FilterConfig 接口对象来访问初始化参数。

<filter-mapping>元素用于设置一个 Filter 所负责拦截的资源。

一个 Filter 拦截的资源可通过两种方式来指定：Servlet 名称和资源访问的请求路径(url 样式)

–<filter-name>用于设置 filter 的注册名称。该值必须是在<filter>元素中声明过的过滤器的名字

–<url-pattern>设置 filter 所拦截的请求路径(过滤器关联的 URL 样式)

–<servlet-name>指定过滤器所拦截的 Servlet 名称。

–<dispatcher>**指定过滤器所拦截的资源被 Servlet 容器调用的方式**，

可以是 REQUEST,INCLUDE,FORWARD 和 ERROR 之一，**默认 REQUEST**. **可以设置多个<dispatcher> 子元素用来指定 Filter 对资源的多种调用方式进行拦截**

例如：

```
<filter>
<filter-name>EncodingFilter</filter-name>
<filter-class>com.ddb.javaweb.filter.EncodingFilter</filter-class>
<init-param>
<param-name>charset</param-name>
<param-value>UTF-8</param-value>
</init-param>
</filter>
<filter-mapping>
<filter-name>EncodingFilter</filter-name>
<servlet-name>MyLoginServletDemo</servlet-name>
<dispatcher>REQUEST</dispatcher>
<dispatcher>INCLUDE</dispatcher>
<dispatcher>FORWARD</dispatcher>
<dispatcher>ERROR</dispatcher>
</filter-mapping>
```

注意：Servlet 的 name，一定要是注册的名字。

12.注解的配置方式：

使用的注解是：@WebFilter()

注解使用的位置是：在 Filter 类的上面。

查看注解支持的所有属性，可以查看 API 手册；也可以利用 Eclispe 的 JD 反编译插件，查看注解的源码，方法是：将鼠标移动到注解上，按下 ctrl 键，光标变成手型，单击即可打开源码。

例如:

```
@WebFilter(
//filterName="f03",
dispatcherTypes = {
DispatcherType.REQUEST,
DispatcherType.FORWARD,
DispatcherType.INCLUDE,
DispatcherType.ERROR
}
,
description = "默认使用 UTF-8 的字符集编码格式",
//urlPatterns = { "/servlet/MyLoginServlet2" },
servletNames={"MyLoginServletDemo"},
initParams = {
@WebInitParam(name = "charset", value = "UTF-8", description = "utf-8 编码
")
```

})

注意：注解配置的 Servlet 的名称配置要使用 name 属性，例如：

```
@WebServlet(
name="MyLoginServletDemo",
urlPatterns={"/servlet/MyLoginServlet2"}
)
```

13.过滤器常见的应用场景：

A.使浏览器不缓存页面的过滤器：

有 3 个 HTTP 响应头字段都可以禁止浏览器缓存当前页面，它们在 Servlet 中的示例代码如下：

```
response.setDateHeader("Expires",-1);
response.setHeader("Cache-Control","no-cache");
response.setHeader("Pragma","no-cache");
```

并不是所有的浏览器都能完全支持上面的三个响应头，因此最好是同时使用上面的三个响应头.

B.字符编码的过滤器

通过配置参数 encoding 指明使用何种字符编码,以处理 Html Form 请求参数的中文问题.

C.检测用户是否登陆的过滤器：

情景：系统中的某些页面只有在正常登陆后才可以使用，用户请求这些页面时要检查 session 中有无该用户信息，但在所有必要的页面加上 session 的判断相当麻烦的事情

解决方案：编写一个用于检测用户是否登陆的过滤器，如果用户未登录，则重定向到指的登录页面

要求：需检查的在 Session 中保存的关键字； 如果用户未登录，需重定向到指定的页面(URL 不包括 ContextPath)；不做检查的 URL 列表(以分号分开，并且 URL 中不包括 ContextPath)都要采取可配置的方式

14. 多个 Filter **执行顺序：**----在 Servlet 之前执行是肯定的！

A. 在注解中是 Filter 类名的首字母在 26 个字母中的顺序，

可以通过调整首字母改变过滤器的执行顺序。

B. 在 XML 中，过滤器置的顺序，就是其执行的顺序！

Filter 的启动顺序与 Filter 的名称有关，可以使用单个字母加编号，如：

f01，f02、来控制启动顺序；1-2-3 ---超过 10 又有变化

filter01，filter02，filter03；3-2-1 ---超过 10 又有变化

15. **编码过滤器若是在其他过滤器后面，可能导致编码失效！**