WebAsyncTask

```
处理线程：处理线程 属于 web 服务器线程，负责 处理用户请求，采用 线程池 管理
异步线程：异步线程 属于 用户自定义的线程，可采用 线程池管理


springmvc 中 HandlerMethodReturnValueHandler
    -> 用来处理实际的请求出来
    -> 处理参数校验
    等等


DeferredResultMethodReturnValueHandler
    ->  设置异步环境
    ->  通过 WebAsynManager.startCallableProcessing 提交到线程池，并设置对应回调
    ->  线程执行之后， 设置异步环境关闭，重新调用 request.dispatch() 将结果返回给客户端


WebAsynManager
    -> WebAsyncTask 的核心支持类
    -> 提交请求到线程池
    -> 管理请求回调
    -> 触发结果返回给用户的流程
```

Schedule

```
ScheduledFutureTask
    DelayedWorkQueue  ->  最小堆
    Runnable + trigger 为任务的最小单元
    ScheduledMethodRunnable
```

spring session

```
org.springframework.session.web.http.OncePerRequestFilter
SessionRepositoryFilter.doFilterInternal(request, response,  filterChain)
```

```java
@Override
protected void doFilterInternal(HttpServletRequest request,
        HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {
    request.setAttribute(SESSION_REPOSITORY_ATTR, this.sessionRepository);

    SessionRepositoryRequestWrapper wrappedRequest = new SessionRepositoryRequestWrapper(
            request, response, this.servletContext);
    SessionRepositoryResponseWrapper wrappedResponse = new SessionRepositoryResponseWrapper(
            wrappedRequest, response);

    try {
        filterChain.doFilter(wrappedRequest, wrappedResponse);
    }
    finally {
        wrappedRequest.commitSession();
    }
}
```

```java
/**
 * Uses the {@link HttpSessionIdResolver} to write the session id to the response
 * and persist the Session.
 */
private void commitSession() {
    HttpSessionWrapper wrappedSession = getCurrentSession();
    if (wrappedSession == null) {
        if (isInvalidateClientSession()) {
            SessionRepositoryFilter.this.httpSessionIdResolver.expireSession( request: this,
                    this.response);
        }
    }
    else {
        S session = wrappedSession.getSession();
        clearRequestedSessionCache();
        SessionRepositoryFilter.this.sessionRepository.save(session);
        String sessionId = session.getId();
        if (!isRequestedSessionIdValid()
                || !sessionId.equals(getRequestedSessionId())) {
            SessionRepositoryFilter.this.httpSessionIdResolver.setSessionId( request: this,
                    this.response, sessionId);
        }
    }
}
```