## Spring session介绍

session一直都是我们做集群时需要解决的一个难题，过去我们可以从serlvet容器上解决，比如开源servlet容器-tomcat提供的tomcat-Redis-session-manager、memcached-session-manager。   
或者通过nginx之类的负载均衡做ip\_hash，路由到特定的服务器上..   
但是这两种办法都存在弊端。

spring-session是spring旗下的一个项目，把servlet容器实现的httpSession替换为spring-session，专注于解决 session管理问题。可简单快速且无缝

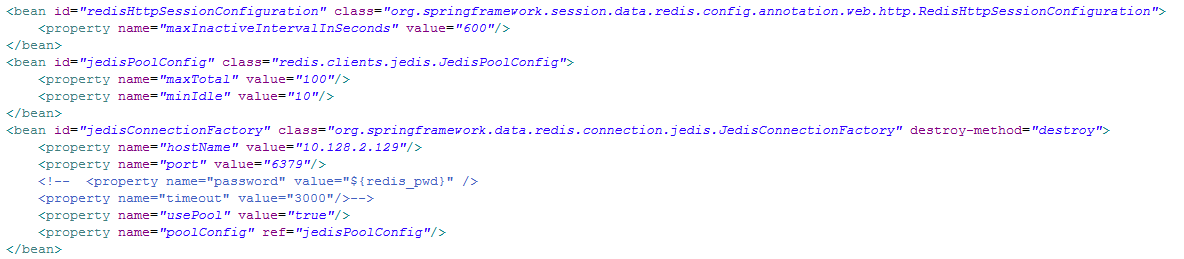
### 支持的功能

1）轻易把session存储到第三方存储容器，框架提供了redis、jvm的map、mongo、gemfire、hazelcast、jdbc等多种存储session的容器的方式。   
2）同一个浏览器同一个网站，支持多个session问题。   
3）Restful API，不依赖于cookie。可通过header来传递jessionID   
4）WebSocket和spring-session结合，同步生命周期管理。

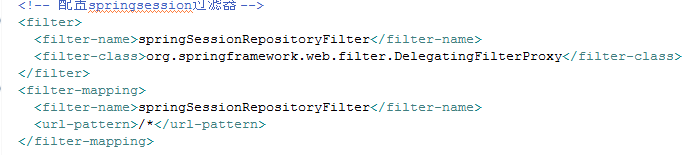
### 集成方式

引入jar包spring-session-data-redis

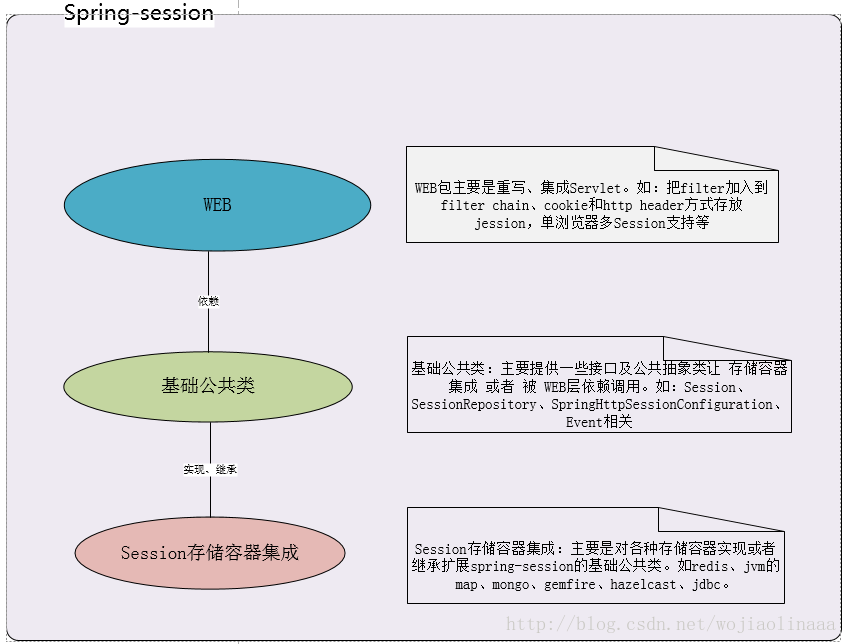
注解或者xml方式配置：

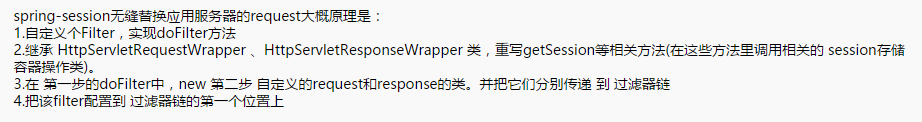


配置过滤器



## 原理分析





/\*\* 这个类是spring-session的1.30源码，也是实现上面第一到第三步的关键类 \*\*/

public class SessionRepositoryFilter<S extends ExpiringSession>

extends OncePerRequestFilter {

/\*\* session存储容器接口，redis、mongoDB、genfire等数据库都是实现该接口 \*\*/

private final SessionRepository<S> sessionRepository;

private ServletContext servletContext;

/\*\*

sessionID的传递方式接口。目前spring-session自带两个实现类

1.cookie方式 ：CookieHttpSessionStrategy

2.http header 方式：HeaderHttpSessionStrategy

当然，我们也可以自定义其他方式。

\*\*/

private MultiHttpSessionStrategy httpSessionStrategy = new CookieHttpSessionStrategy();

public SessionRepositoryFilter(SessionRepository<S> sessionRepository) {

if (sessionRepository == null) {

throw new IllegalArgumentException("sessionRepository cannot be null");

}

this.sessionRepository = sessionRepository;

}

public void setHttpSessionStrategy(HttpSessionStrategy httpSessionStrategy) {

if (httpSessionStrategy == null) {

throw new IllegalArgumentException("httpSessionStrategy cannot be null");

}

/\*\*

通过前面的spring-session功能介绍，我们知道spring-session可以支持单浏览器多

session， 就是通过MultiHttpSessionStrategyAdapter来实现的。

每个浏览器拥有一个sessionID，但是这个sessionID拥有多个别名（根据浏览器的tab）。如：

别名1 sessionID

别名2 sessionID

...

而这个别名通过url来传递，这就是单浏览器多session原理了

\*\*/

this.httpSessionStrategy = new MultiHttpSessionStrategyAdapter(

httpSessionStrategy);

}

public void setHttpSessionStrategy(MultiHttpSessionStrategy httpSessionStrategy) {

if (httpSessionStrategy == null) {

throw new IllegalArgumentException("httpSessionStrategy cannot be null");

}

this.httpSessionStrategy = httpSessionStrategy;

}

/\*\*

该方法相当于重写了doFilter，只是spring-session又做了多一层封装。

在这个方法里创建自定义的 request和response，然后传递到过滤器链filterChain

\*\*/

@Override

protected void doFilterInternal(HttpServletRequest request,

HttpServletResponse response, FilterChain filterChain)

throws ServletException, IOException {

request.setAttribute(SESSION\_REPOSITORY\_ATTR, this.sessionRepository);

/\*\*

spring-session重写的ServletRequest。这个类继承了HttpServletRequestWrapper

\*\*/

SessionRepositoryRequestWrapper wrappedRequest = new SessionRepositoryRequestWrapper(

request, response, this.servletContext);

SessionRepositoryResponseWrapper wrappedResponse = new SessionRepositoryResponseWrapper(

wrappedRequest, response);

HttpServletRequest strategyRequest = this.httpSessionStrategy

.wrapRequest(wrappedRequest, wrappedResponse);

HttpServletResponse strategyResponse = this.httpSessionStrategy

.wrapResponse(wrappedRequest, wrappedResponse);

try {

/\*\*

传递自定义 request和response到链中,想象下如果

该spring-sessionFilter位于过滤器链的第一个，那么后续的Filter，

以及到达最后的控制层所获取的 request和response，是不是就是我们自定义的了？

\*\*/

filterChain.doFilter(strategyRequest, strategyResponse);

}

finally {

wrappedRequest.commitSession();

}

}

public void setServletContext(ServletContext servletContext) {

this.servletContext = servletContext;

}

/\*\*

这个就是Servlet response的重写类了

\*/

private final class SessionRepositoryResponseWrapper

extends OnCommittedResponseWrapper {

private final SessionRepositoryRequestWrapper request;

SessionRepositoryResponseWrapper(SessionRepositoryRequestWrapper request,

HttpServletResponse response) {

super(response);

if (request == null) {

throw new IllegalArgumentException("request cannot be null");

}

this.request = request;

}

/\*\*

这步是持久化session到存储容器，我们可能会在一个控制层里多次调用session的操作方法

如果我们每次对session的操作都持久化到存储容器，必定会带来性能的影响。比如redis

所以我们可以在整个控制层执行完毕了，response返回信息到浏览器时，才持久化session

\*\*/

@Override

protected void onResponseCommitted() {

this.request.commitSession();

}

}

/\*\*

spring-session 的request重写类，这几乎是最重要的一个重写类。里面重写了获取getSession，Session等方法以及类

\*/

private final class SessionRepositoryRequestWrapper

extends HttpServletRequestWrapper {

private Boolean requestedSessionIdValid;

private boolean requestedSessionInvalidated;

private final HttpServletResponse response;

private final ServletContext servletContext;

private SessionRepositoryRequestWrapper(HttpServletRequest request,

HttpServletResponse response, ServletContext servletContext) {

super(request);

this.response = response;

this.servletContext = servletContext;

}

/\*\*

\* Uses the HttpSessionStrategy to write the session id to the response and

\* persist the Session.

\*/

private void commitSession() {

HttpSessionWrapper wrappedSession = getCurrentSession();

if (wrappedSession == null) {

// session失效，删除cookie或者header

if (isInvalidateClientSession()) {

SessionRepositoryFilter.this.httpSessionStrategy

.onInvalidateSession(this, this.response);

}

}

else {

S session = wrappedSession.getSession();

SessionRepositoryFilter.this.sessionRepository.save(session);

if (!isRequestedSessionIdValid()

|| !session.getId().equals(getRequestedSessionId())) {

// 把cookie或者header写回给浏览器保存

SessionRepositoryFilter.this.httpSessionStrategy.onNewSession(session,

this, this.response);

}

}

}

@SuppressWarnings("unchecked")

private HttpSessionWrapper getCurrentSession() {

return (HttpSessionWrapper) getAttribute(CURRENT\_SESSION\_ATTR);

}

private void setCurrentSession(HttpSessionWrapper currentSession) {

if (currentSession == null) {

removeAttribute(CURRENT\_SESSION\_ATTR);

}

else {

setAttribute(CURRENT\_SESSION\_ATTR, currentSession);

}

}

@SuppressWarnings("unused")

public String changeSessionId() {

HttpSession session = getSession(false);

if (session == null) {

throw new IllegalStateException(

"Cannot change session ID. There is no session associated with this request.");

}

// eagerly get session attributes in case implementation lazily loads them

Map<String, Object> attrs = new HashMap<String, Object>();

Enumeration<String> iAttrNames = session.getAttributeNames();

while (iAttrNames.hasMoreElements()) {

String attrName = iAttrNames.nextElement();

Object value = session.getAttribute(attrName);

attrs.put(attrName, value);

}

SessionRepositoryFilter.this.sessionRepository.delete(session.getId());

HttpSessionWrapper original = getCurrentSession();

setCurrentSession(null);

HttpSessionWrapper newSession = getSession();

original.setSession(newSession.getSession());

newSession.setMaxInactiveInterval(session.getMaxInactiveInterval());

for (Map.Entry<String, Object> attr : attrs.entrySet()) {

String attrName = attr.getKey();

Object attrValue = attr.getValue();

newSession.setAttribute(attrName, attrValue);

}

return newSession.getId();

}

// 判断session是否有效

@Override

public boolean isRequestedSessionIdValid() {

if (this.requestedSessionIdValid == null) {

String sessionId = getRequestedSessionId();

S session = sessionId == null ? null : getSession(sessionId);

return isRequestedSessionIdValid(session);

}

return this.requestedSessionIdValid;

}

private boolean isRequestedSessionIdValid(S session) {

if (this.requestedSessionIdValid == null) {

this.requestedSessionIdValid = session != null;

}

return this.requestedSessionIdValid;

}

private boolean isInvalidateClientSession() {

return getCurrentSession() == null && this.requestedSessionInvalidated;

}

private S getSession(String sessionId) {

// 从session存储容器中根据sessionID获取session

S session = SessionRepositoryFilter.this.sessionRepository

.getSession(sessionId);

if (session == null) {

return null;

}

// 设置sesison的最后访问时间，以防过期

session.setLastAccessedTime(System.currentTimeMillis());

return session;

}

/\*\*

这个方法是不是很熟悉，下面还有个getSession()才更加熟悉。没错，就是在这里重新获取session方法

\*\*/

@Override

public HttpSessionWrapper getSession(boolean create) {

//快速获取session，可以理解为一级缓存、二级缓存这种关系

HttpSessionWrapper currentSession = getCurrentSession();

if (currentSession != null) {

return currentSession;

}

//从httpSessionStratge里面根据cookie或者header获取sessionID

String requestedSessionId = getRequestedSessionId();

if (requestedSessionId != null

&& getAttribute(INVALID\_SESSION\_ID\_ATTR) == null) {

//从存储容器获取session以及设置当次初始化属性

S session = getSession(requestedSessionId);

if (session != null) {

this.requestedSessionIdValid = true;

currentSession = new HttpSessionWrapper(session, getServletContext());

currentSession.setNew(false);

setCurrentSession(currentSession);

return currentSession;

}

else {

if (SESSION\_LOGGER.isDebugEnabled()) {

SESSION\_LOGGER.debug(

"No session found by id: Caching result for getSession(false) for this HttpServletRequest.");

}

setAttribute(INVALID\_SESSION\_ID\_ATTR, "true");

}

}

if (!create) {

return null;

}

if (SESSION\_LOGGER.isDebugEnabled()) {

SESSION\_LOGGER.debug(

"A new session was created. To help you troubleshoot where the session was created we provided a StackTrace (this is not an error). You can prevent this from appearing by disabling DEBUG logging for "

+ SESSION\_LOGGER\_NAME,

new RuntimeException(

"For debugging purposes only (not an error)"));

}

// 如果该浏览器或者其他http访问者是初次访问服务器，则为他创建个新的session

S session = SessionRepositoryFilter.this.sessionRepository.createSession();

session.setLastAccessedTime(System.currentTimeMillis());

currentSession = new HttpSessionWrapper(session, getServletContext());

setCurrentSession(currentSession);

return currentSession;

}

@Override

public ServletContext getServletContext() {

if (this.servletContext != null) {

return this.servletContext;

}

// Servlet 3.0+

return super.getServletContext();

}

@Override

public HttpSessionWrapper getSession() {

return getSession(true);

}

@Override

public String getRequestedSessionId() {

return SessionRepositoryFilter.this.httpSessionStrategy

.getRequestedSessionId(this);

}

/\*\*

HttpSession的重写类

\*/

private final class HttpSessionWrapper extends ExpiringSessionHttpSession<S> {

HttpSessionWrapper(S session, ServletContext servletContext) {

super(session, servletContext);

}

@Override

public void invalidate() {

super.invalidate();

SessionRepositoryRequestWrapper.this.requestedSessionInvalidated = true;

setCurrentSession(null);

SessionRepositoryFilter.this.sessionRepository.delete(getId());

}

}

}

}