스프링 어플리케이션의 문제해결 사례 & 안티 패턴

2014-06-08 Spring Camp 2014

정상혁

발표자

정 상 혁

2004 ~ 2008 : 삼성SDS

S/W엔지니어링팀에서 공공프로젝트 수행

2008 ~ 현재 : NHN, NHN Technology Services, NBP/Naver Labs

생산성혁신랩에서 신규 프로젝트 개발 지원

웹플랫폼개발랩에서 프레임워크개발/기술지원

발표내용

스프링으로 만든 웹어플리케이션에서

- 치명적인 문제를 유발하는 사용방식
- 프레임워크의 장점을 못 살리는 불편한 사용방식

치명적인 사용방식

관례를 고려하지 않은 컴파일 옵션 변경

@PathVariable, @RequestParam의 속성을 생략했을 때

```
@RequestMapping(value="/user/{id}")
public String user(@PathVariable String id){
}
```

```
@RequestMapping(value="/userList/ ")
public String user(@RequestParam String name){
}
```

어떤 프로젝트에서는 서버에 올리니 에러가 난다 (IllegalArgumentException)

관례를 고려하지 않은 컴파일 옵션 변경

@PathVariable, @RequestParam의 속성을 생략했을 때

컴파일 옵션을 확인

Debug가 false라면 제대로 동작하지 않는다

관례를 고려하지 않은 컴파일 옵션 변경

```
<plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.6</source>
        <target>1.6</target>
        <debug>false</debug>
        <optimize>true</optimize>
        <encoding>utf-8</encoding>
      </configuration>
    </plugin>
</plugins>
```

관례를 고려하지 않은 컴파일 옵션 변경

원인

명시된 속성값이 없으면 Spring에서 Debug정보를 참고하기 때문

해결방법

컴파일옵션에 debug=false를 명시하지 않는다. (디폴트는 true) 또는

@PathVariable 등을 쓸 때 속성을 명시한다

```
@RequestMapping(value="/user/{id}")
public String user(@PathVariable("id") String id){
}
```

관례를 고려하지 않은 컴파일 옵션 변경

참고자료

@PathVariable을 사용할 때 주의할점. 컴파일러 상태에 따라 오류가 날수도… :
http://gubok.tistory.com/382

Spring Framework 2.5의 Annotation based Controller의 메서드 파라미터에서 주의점 :
http://corund.net/blog/entry/Spring-Framework-2.5%EC%9D%98-Annotation-based-Controller%EC%9D%98-%EB%A9%94%EC%84%9C%EB%93%9C-%ED%8C%8C%EB%9D%BC%EB%AF%B8%ED%84%B0%EC%97%90%EC%84%9C-%EC%A3%BC%EC%9D%98%EC%A0%90

View의 Cache를 의식하지 않은 ViewName

Redirect url에 변수를 더하는 경우

매번 변하는 ViewName이 Controller에서 String return type이나 ModelAndView의 ViewName으로 지정될 때

```
return "redirect:form.html?entityId=" + entityId;
```

```
modelAndView.setViewName(
    "redirect:form.html?entityId=" + entityId
);
```

서버에 올린 뒤 오래되면 OOM이 발생

View의 Cache를 의식하지 않은 ViewName

원인

ViewResolver가 viewName으로 View를 캐쉬한다. (AbstractCachingViewResolver의 구현 방식)

해결방법

버전업 : 스프링 3.1.4와 3.2.GA버전에 OOM이 방어됨

OOM이 안 나는 버전을 쓰더라도 Cache 효율성을 감안하여 사용

View의 Cache를 의식하지 않은 ViewName

해결방법 (OOM 방어가 안 된 버전에서도)

View를 직접 Return

```
return new RedirectView("form.html?entityId="+entityId);
```

URI template 활용 (Spring 3.1 이상)

```
return "redirect:form.html?entityId={entityId}";
```

View의 Cache를 의식하지 않은 ViewName

RedirectAttributes (Spring 3.1 이상)

```
@RequestMapping(method = RequestMethod.POST)
public String onPost(RedirectAttributes attrs) {
    ...
    attrs.addAttribute(entityId, 123);
    return "redirect:form.html";
}
```

View의 Cache를 의식하지 않은 ViewName

참고자료 : 이슈 트래커의 SPR-10065 (View 캐쉬의 OOM 방어)

2012년 12월 03일 : 이슈 올라옴 (https://jira.spring.io/browse/SPR-10065)
AbstractCachingViewResolver - caching redirect views leads to memory leak

2012년 12월 11일 : Commit by Juergen Hoeller https://github.com/spring-projects/spring-framework/commit/9deaefe74d 오래된 View를 지우는 구현을 추가
LinkedHashMap.removeEldestEntry() override해서 활용

2012년 12월 13일 : 커밋이 반영된 3.2 GA 버전 릴리즈 2013년 01월 23일 : 커밋이 반영된 3.1.4 버전 릴리즈

Redirect url에 변수를 더하기

참고자료 : 이슈 트래커의 SPR-3145 (View 캐쉬의 성능개선)

2006년 12월 06일 : 이슈 올라옴 (https://jira.spring.io/browse/SPR-3145) Performance improvement on AbstractCachingViewResolver 당시는 Java 5 이전 버전도 지원해야 했기 때문에 ConcurrentHashMap을 도입 못함

2013년 2월 06일 : Commit by Juergen Hoeller https://github.com/SpringSource/spring-framework/commit/06c6cbb6b92 앞에 나온 OOM방어 때문에 LinkedHashMap.removeEldestEntry(…)를 계속 유지. ConcurrentHashMap과 LinkedHashMap을 동시에 사용하고, 새로 View를 생성할 때만 LinkedHashMap을 synchronized 로 잡는 방식을 선택

2013년 3월 14일 : 커밋이 반영된 Spring 3.2.2 릴리즈

매번 생성되는 객체에 @Async 적용

`<task:annotation-driven/>` + Prototype bean

@Async, @Scheduled를 쓰기 위해 쓰면서 `<task:annotation-driven/>` 을 추가

Scope=prototype 혹은 @Configurable 선언으로 Spring에서 관리하는 객체가 자주 생성될 때

```
<task:annotation-driven executor="asyncExecutor"/>

<bean id="myService" class="...Service" scope="prototype"/>
```

CPU 사용률이 비정상적으로 올라감

매번 생성되는 객체에 @Async 적용

원인

AOP 대상 여부를 검사하는 코드 때문에 모든 Spring Bean의 생성비용이 올라감. 내부에서 호출되는 AopUtils.canApply 메서드가 Spring 3.1까지는 성능저하가 심했음

```
- locked <0x00002aaabb154148> (a java.lang.reflect.Method)
at java.lang.reflect.Method.getAnnotation(Method.java:679)
at java.lang.reflect.AccessibleObject.isAnnotationPresent(AccessibleObject.java:168)
at org.springframework.aop.support.annotation.AnnotationMethodMatcher.matches(AnnotationMethodMatcher.java:56)
at org.springframework.aop.support.MethodMatchers$UnionMethodMatcher.matches(MethodMatchers.java:121)
at org.springframework.aop.support.AopUtils.canApply(AopUtils.java:226)
at org.springframework.aop.support.AopUtils.canApply(AopUtils.java:263)
at org.springframework.aop.support.AopUtils.canApply(AopUtils.java:244)
at org.springframework.scheduling.annotation.AsyncAnnotationBeanPostProcessor.postProcessAfterInitialization(AsyncAnnotationBeanPostProcessor.java:125)
```

매번 생성되는 객체에 @Async 적용

해결 방법

Spring 3.2 이상 업그레이드 또는

`<task:annotation-driven/>` 이 적용되는 ApplicationContext에는 singleton bean만 등록되도록 설정 정리 또는

`<task:annotation-driven/>` 선언을 사용하지 않고 직접 Executor사용

매번 생성되는 객체에 @Async 적용

참고 자료

AopUtils.canApply의 성능개선 논의 이슈

https://jira.springsource.org/browse/SPR-8065

https://jira.springsource.org/browse/SPR-7328

유사 문제 사례

http://stackoverflow.com/questions/6729860/slow-down-with-combining-scheduled-and-configurable

http://www.solutionoferror.com/java/slow-down-with-combining-scheduled-and-configurable-288213.asp

생성자에서 Lock을 잡는 객체를 매번 생성

StringHttpMessageConverter를 매 요청마다 생성

```
public void handleRequest(HttpServletRequest request) {
    HttpMessageConverter<String> converter =
                        new StringHttpMessageConverter();
}
```

고부하 상황에서 CPU는 다 쓰지 않으면서 TPS가 더 이상 올라가지 않는다.

생성자에서 Lock을 잡는 객체를 매번 생성

원인

생성시에 encoding을 위해 시스템이 지원하는 character set을 확인하게 됨

charsets.jar 파일 안의 객체를 동적 로딩하게 되는데, 동적 로딩을 하는 jdk 코드 내 synchronized로 감싼 코드로 인해 locking

길지 않은 Lock구간이지만 대량 요청 시에는 문제가 됨

```
at java.nio.charset.Charset$1.getNext(Charset.java:317)
at java.nio.charset.Charset$1.hasNext(Charset.java:332)
at java.nio.charset.Charset$4.run(Charset.java:551)
at java.security.AccessController.doPrivileged(Native Method)
at java.nio.charset.Charset.availableCharsets(Charset.java:546)
at org.springframework.http.converter.StringHttpMessageConverter.(StringHttpMessageConverter.java:52)
…
```

생성자에서 Lock을 잡는 객체를 매번 생성

해결방법

이 클래스는 Thread-safe하므로 매번 생성할 필요 없었음

어플리케이션 초기화 시에 한 번만 생성되도록

ApplicationContext에 Singleton Bean 등록 혹은 직접 생성하더라도 멤버변수로

XXE Injection 취약점 노출

Spring-OXM으로 신뢰할 수 없는 출처의 XML을 파싱할 때

XXE = XML External Entity

아래 조건을 충족시킬 때

- 외부에서 생성한 XML을 파싱
- Spring-OXM 사용

(Spring MVC에서 @RequestBody로 자동 파싱하는 경우도 포함)

XXE Injection 취약점 노출

```
@RequestMapping("/update")
@ResponseBody
public Group update(@RequestBody Person person) {
 …
}
```

서버의 파일 노출 가능

XXE Injection 취약점 노출

원인

SAX, DOM, StAX 등 다양한 근본 구현 기술에서 가진 문제

PHP, C/C++, 닷넷, iOS 등 다른 플랫폼에서도 존재

공격 XML 사례

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE person [
<!ELEMENT person ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
<person>
    <name>&xxe;</name>
</person>
```

XXE Injection 취약점 노출

해결방안

Spring 3.2.5 업그레이드

참고자료

http://www.gopivotal.com/security/cve-2013-4152

http://www.gopivotal.com/security/cve-2013-6429

ClassLoader 노출

class.classLoader 접근식으로 속성 조작이 가능

HttpRequest -> Bean 매핑을 하는 URL에서

```
@RequestMapping("/saveUser")
public String saveUser(User user) {
    ….
    return "index";
}
```

서버의 파일 노출, Remote code execution 가능

ClassLoader 노출

원인

bean의 getter/setter 호출 관례

```
class.classLoader.resource.home =/etc
```

⬇

```
getClass().getClassLoader().getResource().
.setHome("/etc");
```

몇 년 전에는 TLD 파일을 업로드해서 커스텀 태그를 Injection 하는 경로만 알려졌으나 최근 Tomcat의 classLoader 접근 방식이 공개되어 더욱 치명적

ClassLoader 노출

해결방안

Spring 3.0.3 이상 업그레이드

해당 버전에서 패치된 부분 (CachedIntrospectionResults.java의 245~246행)

```
if (Class.class.equals(beanClass) && "classLoader".equals(pd.getName())) {
 // Ignore Class.getClassLoader() method - nobody needs to bind to that
 continue;
}
```

ClassLoader 노출

참고자료

http://support.springsource.com/security/cve-2010-1622

2010년 5월 17일 : Commit by Juergen Hoeller : https://github.com/spring-projects/spring-framework/commit/3a5af35d37

2010년 6월 16일 : 커밋이 반영된 3.0.3 버전 릴리즈

TLD업로드 공격방법에 대한 설명 : https://www.troopers.de/wp-content/uploads/2010/12/TR11_Meder_Milking_a_horse.pdf 의 50페이지

Struts2의 유사사례 : http://hacksum.net/?p=2103

EL Injection 취약점 노출

Tomcat 7 + Spring의 커스텀 태그를 사용할 때

아래 조건을 모두 충족할 때

- EL 2.2를 지원하는 서블릿 컨테이너를 쓰거나 EL 2.2 라이브러리를 직접 jar파일로 참조해서 쓰고 있다. (대표적으로 Tomcat 7.x혹은 Glassfish 2.2.x)
- Spring 3.1.x 미만 버전을 쓰고 있다.
- Spring의 JSP Tag(`<spring:message..` 등)을 쓰고 있다.
- Spring의 JSP Tag에서 EL을 지원하는 속성에 사용자가 입력한 값이 들어갈 수 있다.

Remote code execution 가능

EL Injection 취약점 노출

해결방법

Spring 3.0.6 혹은 2.5.6.SEC03버전 이상 사용 + web.xml에 추가선언 또는 Spring 3.1.x 버전 이상 사용

참고자료

http://support.springsource.com/security/cve-2011-2730

https://gist.github.com/benelog/4582041

불편한 사용방식

HttpServletRequest, Response 애착

ResponseEntity 등 Spring의 API를 활용하지 않는다.

```
@RequestMapping(value="/product1")
public void product1(HttpServletResponse res) throws IOException {
    // IE에서 이상동작 때문에 TEXT_PLAIN으로 해달라고 Ajax담당자 요청이 있었음
    res.setHeader("Content-Type", "text/plain");

    Product product = newProduct();
    ServletOutputStream output = res.getOutputStream();
    mapper.writeValue(output, product);
}
```

HttpServletRequest, Response 애착

헤더를 조작해야 할 때도 ResponseEntity는 Type-safe한 API를 제공한다.

⬇

```
@RequestMapping("/product2")
public ResponseEntity<Product> product2() throws IOException {
    HttpHeaders headers = new HttpHeaders();
    // IE에서 이상동작 때문에 TEXT_PLAIN으로 해달라고 Ajax담당자 요청이 있었음
    headers.setContentType(MediaType.TEXT_PLAIN);

    Product product = newProduct();
    return new ResponseEntity<Product>(product, headers, HttpStatus.OK);
}
```

Annotation의 속성선언을 매번 반복

@Transactional을 쓸 때

Timeout 등의 속성을 모든 메서드에 지정하는 사례

```
@Transactional(value="account",
               propagation = Propagation.REQUIRED,
               readOnly=false,
               timeout = 3,
               rollbackFor=Exception.class)
public void deleteUser(String id) {
…
}
```

Annotation의 속성선언을 매번 반복

@Transactional을 쓸 때 : 개선

공통 Annotation 정의 가능

```
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional(
    value="order",
    propagation = Propagation.REQUIRES_NEW,
    rollbackFor=Exception.class )
public @interface OrderTx {

}
```

Annotation의 속성선언을 매번 반복

```
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Transactional("account")
public @interface AccountTx {

}
```

Annotation의 속성선언을 매번 반복

@Transactional을 쓸 때 : 개선

공통 Annotation 정의 가능

```
public class CrmService {
  @OrderTx
  public void orderItems(List<Item> items) {
    ...
  }

  @AccountTx
  public void deleteUser(String id) {
    ...
  }
}
```

Custom namespace의 미흡한 활용

ArgumentResolver를 등록할 때

별도로 AnnotationMethodHandlerAdapter를 Bean 등록하는 사례

```
<mvc:annotation-driven/>
<bean id="handlerAdapter"
class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
  <property name="customArgumentResolvers">
    <array>
      <bean class="...MyArgumentResolver"/>
    </array>
  </property>
  <property name="order" value="-1"/>
</bean>
```

Custom namespace의 미흡한 활용

ArgumentResolver를 등록할 때 : 개선

3.1부터는 `<mvc:annotation-driven/>` 내부에서 가능

```
<mvc:annotation-driven>
    <mvc:argument-resolvers>
      <bean class="...MyArgumentResolver"/>
    </mvc:argument-resolvers>
</mvc:annotation-driven>
```

Custom namespace의 미흡한 활용

viewName만 리턴하는 Controller

필요한 정보는 "/" -> "home" 인데 긴 파일을 작성

```
package com.nhncorp.edu.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {

  @RequestMapping("/")
  public String home(){
      return "home";
  }
}
```

Custom namespace의 미흡한 활용

viewName만 리턴하는 Controller : 개선

3.0부터 `<mvc:view-controller/>` 활용

```
<mvc:view-controller path="/" view-name="home"/>
```

정리

버전 확인

가급적 OOM 방어, 보안 취약점 방어, 성능 개선이 된 버전 사용

3.2.5 이상 : XXE Injection 방어

3.2.RC1 이상 : AopUtils.canApply(…)의 성능 개선

3.1.4 이상 : View Cache의 OOM 방어

3.0.6 이상 : EL Injection 방어

3.0.3 이상 : ClassLoader 접근 방어

특히 보안 패치는 이슈별로 확인

http://www.gopivotal.com/security/

http://support.springsource.com/security/springsource-all

버전 확인

업그레이드 시 주의할 점

Spring 3.0 -> 3.1 -> 3.2 따라잡기 참고

https://benelog.github.io/presentations/20130713-spring-upgrade/

사용 관례 정의

사용방식을 합의하고 배경을 공유한다

Annotation의 디폴트 속성 생략 여부

커스텀 네임스페이스 선언(`<mvc:../>`) 활용 규칙

Controller의 Return type 규칙

예) Model이 없을 때는 String, 있을 때는 ModelAndView,

예) Redirect URL은 문자열 더하기 금지

@PathVariable, @RequestParam은 컴파일옵션에 영향받는 속성의 사용 정책

HttpServletRequest, Response 사용 규칙

예: 쿠키를 새로 만들 때만 사용