

# Spring Boot Embedded Web Servers 그리고 Docbase

Spring Boot의 Executable jar를 통해 Embedded Web Server를 기반으로 서비스 할 때 `servletContext.getRealPath("/")` 코드에서 문제가 생기는 경우가 발생할 수 있다.

<https://github.com/spring-projects/spring-boot/issues/5009> 이슈에서 2016년도부터 현재까지 발생하고 있으며 마주할 수 있는 예외 상황은 아래와 같다.

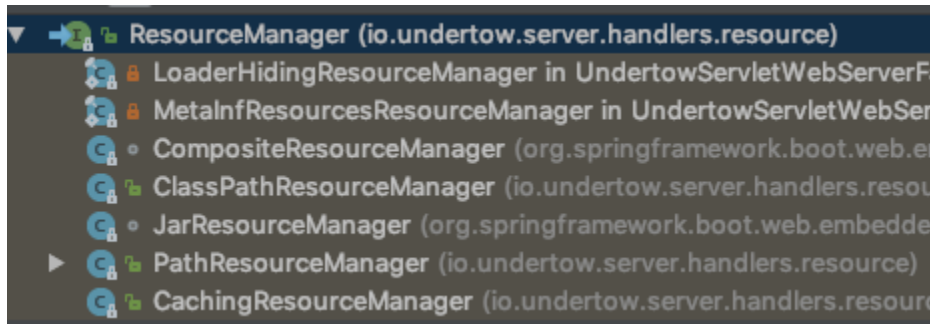
- `java.lang.ClassNotFoundException`
- The temporary upload location [/tmp/tomcat.8614765871571973022.8080/work/Tomcat/localhost/ROOT] is not valid
- Could not parse multipart servlet request; nested exception is `java.io.IOException`: The temporary upload location [/tmp/tomcat.3709455009677465432.8085/work/Tomcat/localhost/ROOT] is not valid
- `java.lang.NullPointerException`: null (`servletContext.getRealPath("/")`를 호출하는 시점에서)

대부분 `servletContext`로부터 root docbase를 가져오는 부분과 관련되어있다.

관련 문제의 원인과 해결법을 찾기 위해 Spring Boot Embedded Web Server의 Docbase는 어떻게 정해지는지 확인이 필요하다.

`servletContext.getRealPath("/")`를 호출하면 `ServletContextImpl`의 `getRealPath()`가 호출되고 `deploymentInfo`의 `getResourceManager().getResource()` 메소드를 통해 처리가 된다.

Undertow의 경우 `ResourceManager`는 `PathResourceManager`가 런타임 의존성이며 `PathResourceManager`에선 `fileSystem`으로부터 base 프로퍼티를 기반으로 하는 디렉토리가 Root가 된다.



```
public Resource getResource(final String p) {
    String path;
    //base always ends with a /
    if (p.startsWith("/")) {
        path = p.substring(1);
    } else {
        path = p;
    }
    try {
        Path file = fileSystem.getPath(base, path);
        ...
    }
}
```

여기서 base는 linux 계열 기준 `/tmp/undertow-docbase.4487171275524936477.9092/` 디렉토리이며 매번 WebServer가 실행 될 때마다 다른 값을 가진다.

정리하자면 `servletContext.getRealPath("/")`를 호출했을 때 대상 디렉토리는 `/tmp/undertow-docbase.4487171275524936477.9092/`와 같은 임시디렉토리이며 null이 반환된다는 것은 해당 디렉토리가 존재하지 않는 것이다.

실제로 문제가 발생했을 때 `/tmp` 디렉토리를 접근하면 `undertow-docbase`로 시작하는 디렉토리가 존재하지 않음을 확인하였고 여러 서버에서 나타나 우연이 아닌 것으로 파악 하였다.

Undertow만의 문제일까 싶어 Embedded WebServer들의 docbase는 어떻게 설정되는 지, 언제 삭제 되는 지 확인이 필요하다.

## Spring Boot Embedded WebServers

<https://docs.spring.io/spring-boot/docs/2.1.9.RELEASE/reference/html/howto-embedded-web-servers.html>

Spring Boot 기반의 Web Application은 기본적으로 Embedded Web Server (더 이상 WAS가 아닌 Web Server로 표기 함)들이 포함되어있다. Spring 5 부터 Reactor의 등장으로 Servlet stack 외에도 Reactive stack이 분리되어 제공되며 각각의 Web Server는 아래와 같다.

### For Servlet Stack

`spring-boot-starter-web` Artifact의 기본 Web Server는 `spring-boot-starter-tomcat`이며 `spring-boot-starter-jetty`, `spring-boot-starter-undertow` 등이 지원된다.

### For Reactive Stack

`spring-boot-starter-webflux` Artifact의 기본 Web Server는 `spring-boot-starter-reactor-netty` (Reactor Netty) 이며 `spring-boot-starter-tomcat`, `spring-boot-starter-jetty`, `spring-boot-starter-undertow` 등이 지원된다.



만약 Servlet stack에서 기본 의존성인 Tomcat 대신 Undertow를 사용하고 싶다면 아래와 같이 Maven에서 의존성을 제거하고 Undertow를 추가하면 된다.

```

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <exclusions>
        <exclusion>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-tomcat</artifactId>
        </exclusion>
    </exclusions>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-undertow</artifactId>
</dependency>

```

각각의 Embedded Web Server의 구조는 아래와 같다.

| 계층구조  | 클래스다이어:  |
|---|--|
|  |  |

Servlet Stack에서 Spring Boot Application이 실행 될 때 ServletWebServerApplicationContext가 초기화 되고 해당 클래스의 createWebServer 메소드에서 Embedded Web Server가 실행된다.

```

private void createWebServer() {
    WebServer webServer = this.webServer;
    ServletContext servletContext = getServletContext();
    if (webServer == null && servletContext == null) {
        ServletWebServerFactory factory = getWebServerFactory();
        this.webServer = factory.getWebServer(getSelfInitializer());
    }
    else if (servletContext != null) {
        try {
            getSelfInitializer().onStartup(servletContext);
        }
        catch (ServletException ex) {
            throw new ApplicationContextException("Cannot initialize servlet context", ex);
        }
    }
    initPropertySources();
}

/**
 * Returns the {@link ServletWebServerFactory} that should be used to create the
 * embedded {@link WebServer}. By default this method searches for a suitable bean in
 * the context itself.
 * @return a {@link ServletWebServerFactory} (never {@code null})
 */
protected ServletWebServerFactory getWebServerFactory() {
    // Use bean names so that we don't consider the hierarchy
    String[] beanNames = getBeanFactory().getBeanNamesForType(ServletWebServerFactory.class);
    if (beanNames.length == 0) {
        throw new ApplicationContextException("Unable to start
ServletWebServerApplicationContext due to missing "
            + "ServletWebServerFactory bean.");
    }
    if (beanNames.length > 1) {
        throw new ApplicationContextException("Unable to start
ServletWebServerApplicationContext due to multiple "
            + "ServletWebServerFactory beans : " + StringUtils.
arrayToCommaDelimitedString(beanNames));
    }
    return getBeanFactory().getBean(beanNames[0], ServletWebServerFactory.class);
}

```

여기서는 Web Server로 Undertow를 설정하였으므로 UndertowServletWebSerFactory의 getWebServer 메소드가 호출된다.

메소드 호출 체인을 따라가다 보면 getWebServer() → createDeploymentManager() → getDocumentRootResourceManager() → getCanonicalDocumentRoot()에서 docbase를 생성하는 것을 볼 수 있다.

```

private ResourceManager getDocumentRootResourceManager() {
    File root = getValidDocumentRoot();
    File docBase = getCanonicalDocumentRoot(root);
    List<URL> metaInfResourceUrls = getUrlsOfJarsWithMetaInfResources();
    List<URL> resourceJarUrls = new ArrayList<>();
    List<ResourceManager> managers = new ArrayList<>();
    ResourceManager rootManager = (docBase.isDirectory() ? new FileResourceManager(docBase, 0)
        : new JarResourceManager(docBase));
    if (root != null) {
        rootManager = new LoaderHidingResourceManager(rootManager);
    }
    managers.add(rootManager);
    ...
}

private File getCanonicalDocumentRoot(File docBase) {
    try {
        File root = (docBase != null) ? docBase : createTmpDir("undertow-docbase");
        return root.getCanonicalFile();
    }
    catch (IOException ex) {
        throw new IllegalStateException("Cannot get canonical document root", ex);
    }
}

```

createTmpDir 메소드는 어떤 WebServerFactory 구현체들도 오버라이딩 하고 있지 않고 AbstractConfigurableWebServerFactory의 구현코드를 상속받아 사용하고 있다. 여기서 prefix로 넘어가는 인자는 "undertow-docbase"이다.

```

/**
 * Return the absolute temp dir for given web server.
 * @param prefix server name
 * @return the temp dir for given server.
 */
protected final File createTmpDir(String prefix) {
    try {
        File tempDir = File.createTempFile(prefix + ".", "." + getPort());
        tempDir.delete();
        tempDir.mkdir();
        tempDir.deleteOnExit();
        return tempDir;
    }
    catch (IOException ex) {
        throw new WebServerException(
            "Unable to create tempDir. java.io.tmpdir is set to " + System.
            getProperty("java.io.tmpdir"), ex);
    }
}

```

로직을 보면 docbase에 해당하는 임시파일을 생성하고 삭제 후 디렉토리를 생성한 후 deleteOnExit() API를 통해 JVM 종료 시 삭제되도록 하고 있다. (파일을 생성하고 삭제하는 것은 디렉토리 생성 시 해당 경로에 파일이 존재할 경우를 대비한 방어코드 같다)

File.createTempFile API의 javadoc을 보면 Linux, Window의 임시디렉토리에 생성한다고 되어있으며 이너클래스인 TempDirectory에선 아래와 같이 java.io.tmpdir 프로퍼티로부터 위치를 얻어오는 것을 볼 수 있다.

```

private static class TempDirectory {
    private TempDirectory() { }

    // temporary directory location
    private static final File tmpdir = new File(AccessController
        .doPrivileged(new GetPropertyAction("java.io.tmpdir")));
    static File location() {
        return tmpdir;
    }
    ...
}

```

아래 Github에서 Linux 계열의 임시디렉토리는 주기적으로 삭제될 가능성이 있어 Embedded tomcat의 work directory를 임시 디렉토리에 구성하는 것은 위험하다는 이슈를 볼 수 있다.

<https://github.com/spring-projects/spring-boot/issues/5009>

위 과정에서 Embedded Web Server들은 모두 동일하게 File.createTempFile API를 통해 docbase, work directory를 구성하는 것을 확인 하였으니 Tomcat 뿐 아니라 우리의 환경인 Undertow에서도 문제의 원인이 됨을 확인할 수 있다.

해결법으로 java.io.tmpdir 프로퍼티에 /var/tmp를 설정하라는 것도 있었는데 <https://unix.stackexchange.com/questions/30489/what-is-the-difference-between-tmp-and-var-tmp>를 확인하면 /tmp는 수명이 짧고 작은 저장소이고 /var/tmp는 임시파일을 더 오래 보관할 수 있는 저장소로 소개하고 있다.

따라서 언젠간 디렉토리 내부가 청소될 수 있는 위험이 있어 최종적인 해결법으로는 java.io.tmpdir을 임시디렉토리가 아닌 위치로 설정하는 것으로 정하였다.

임시파일 생성 시 deleteOnExit 메소드를 실행하고 있기 때문에 비정상 종료 상황이 아닌 이상 임시파일이 많이 쌓이는 상황은 거의 없을 것으로 파악한다.

## 참고자료

<https://github.com/spring-projects/spring-boot/issues/5009>

<https://www.thegeekdiary.com/centos-rhel-67-why-the-files-in-tmp-directory-gets-deleted-periodically/>

<https://unix.stackexchange.com/questions/30489/what-is-the-difference-between-tmp-and-var-tmp>