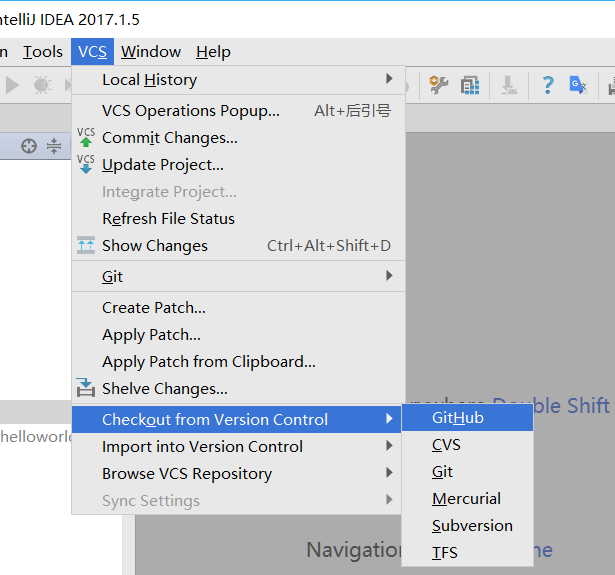
Helloworld

# **导入项目**

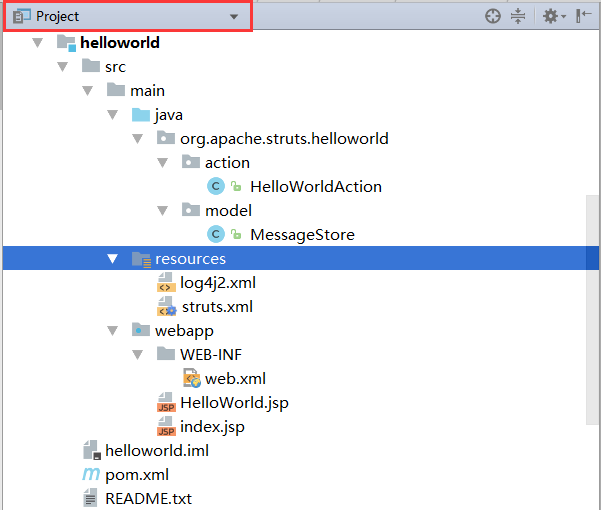
使用开发工具Intellij Idea将struts-example项目导入到本地中，打开helloworld模块；

地址：<https://github.com/guideself/struts-examples.git>，这个是我从官网github中fork了一份到自己的github账号中的



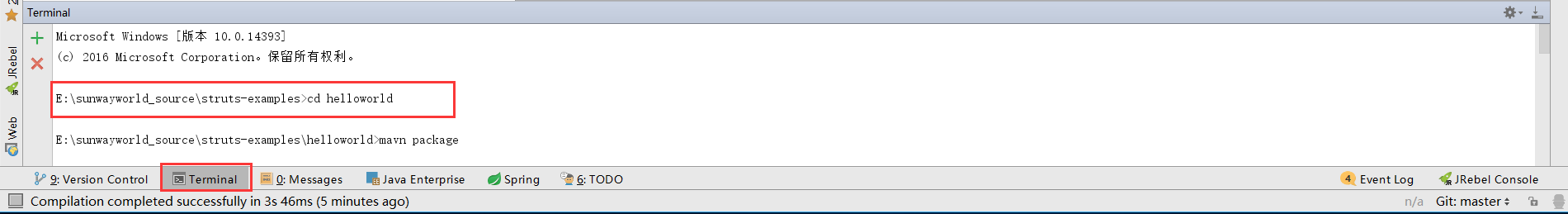
# 项目结构

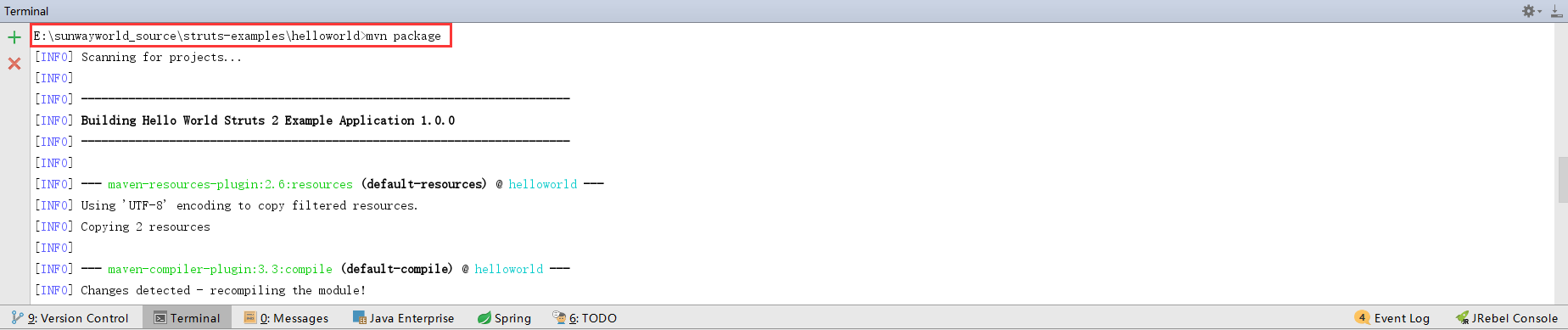
这里是maven管理,以project模式展开项目

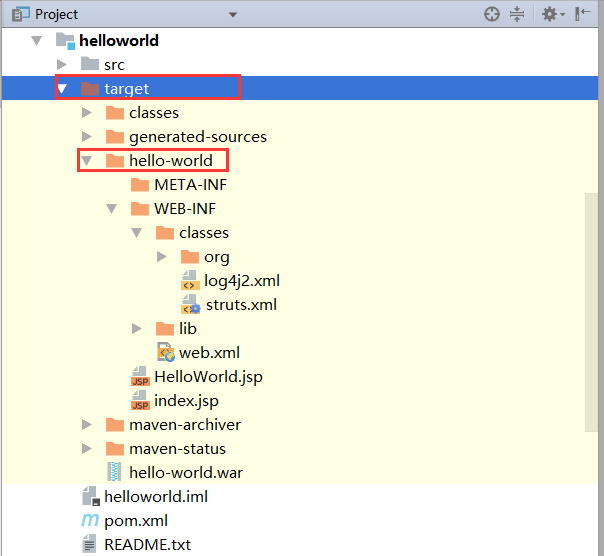


# **编译打包问**部署tomat

进入控制台，执行如下命令：

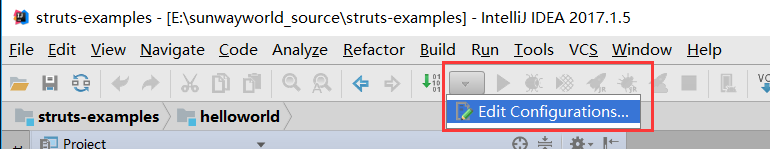


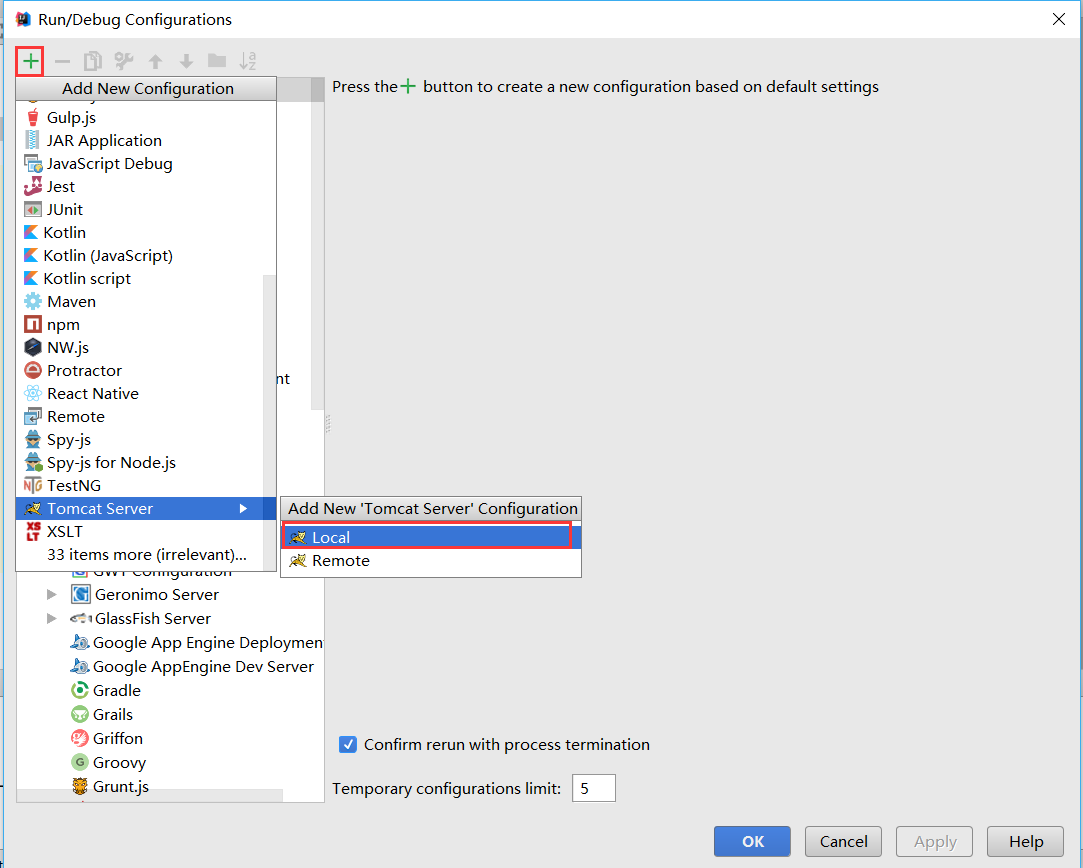


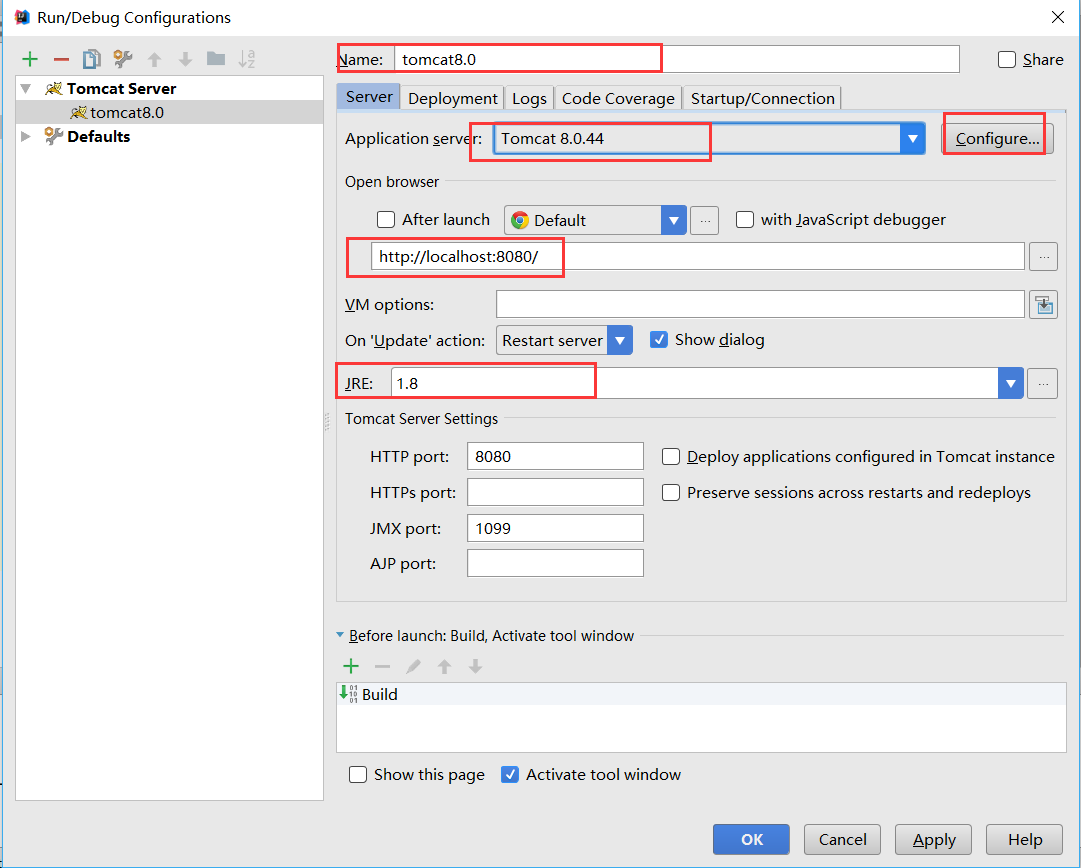


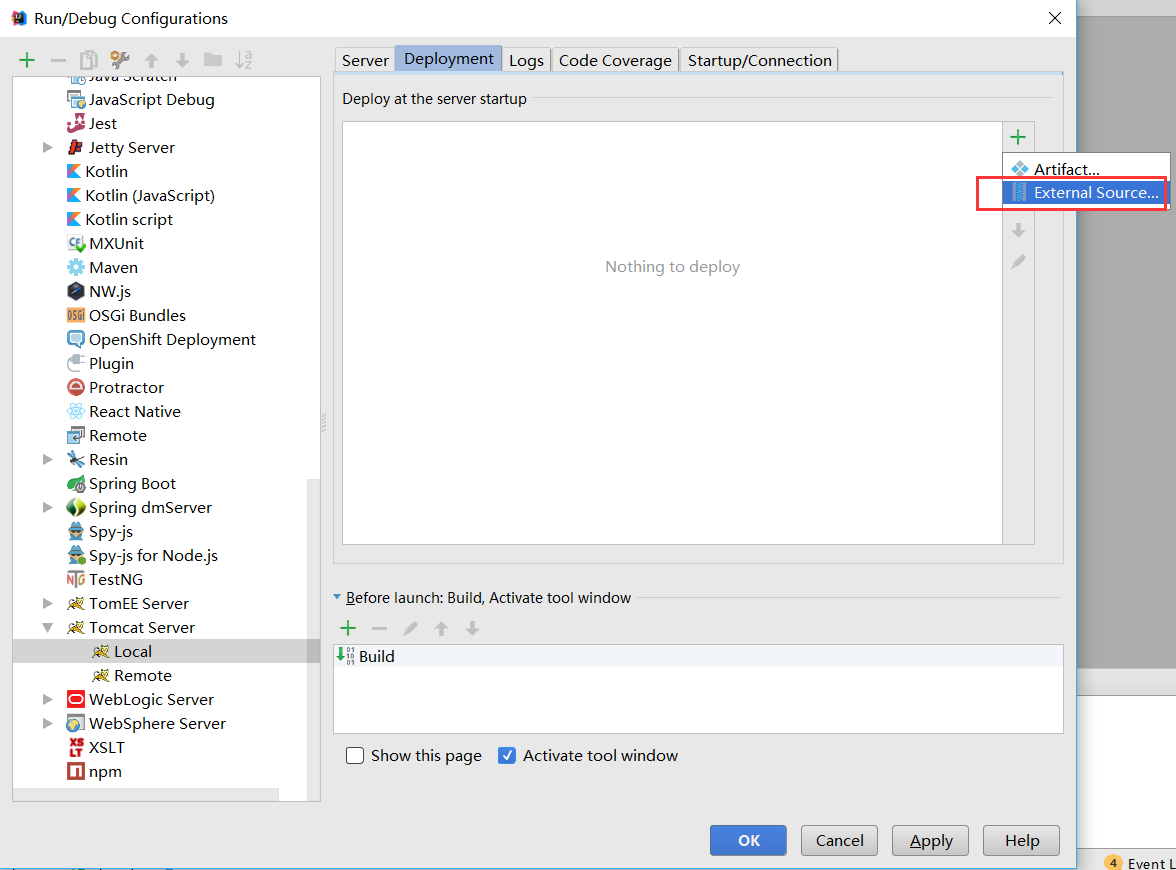
这里生成的target/hello-world文件夹就相当于我们要部署到tomcat的webapp；

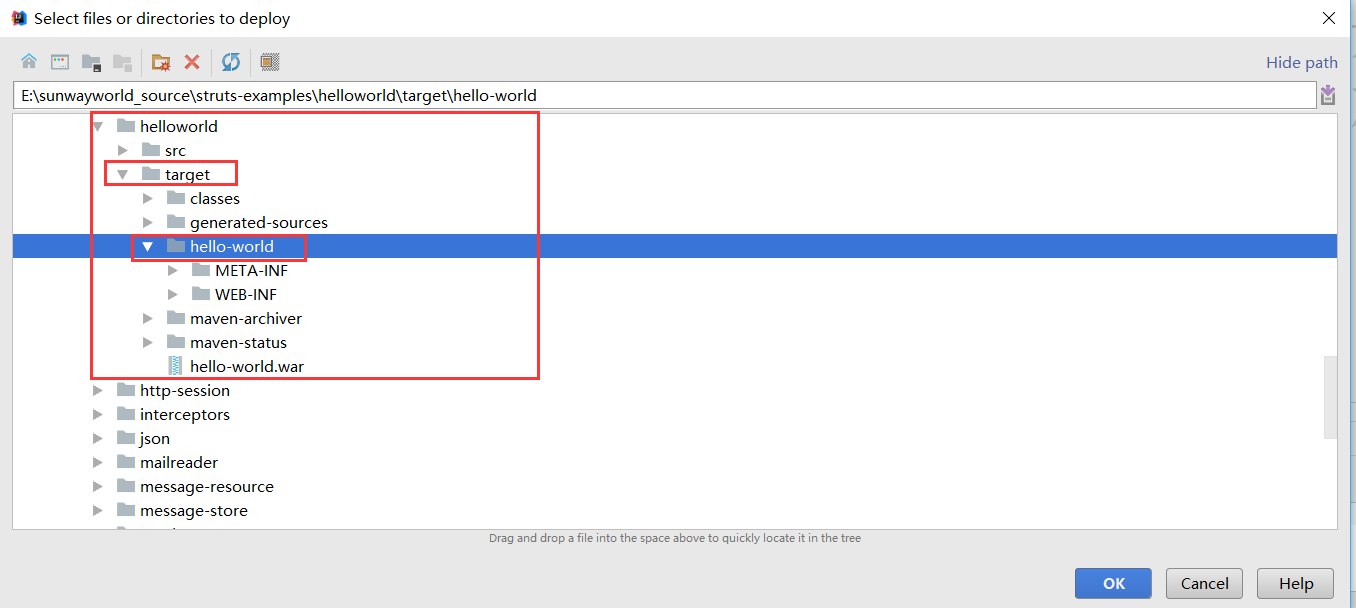
1. 新建tomcat

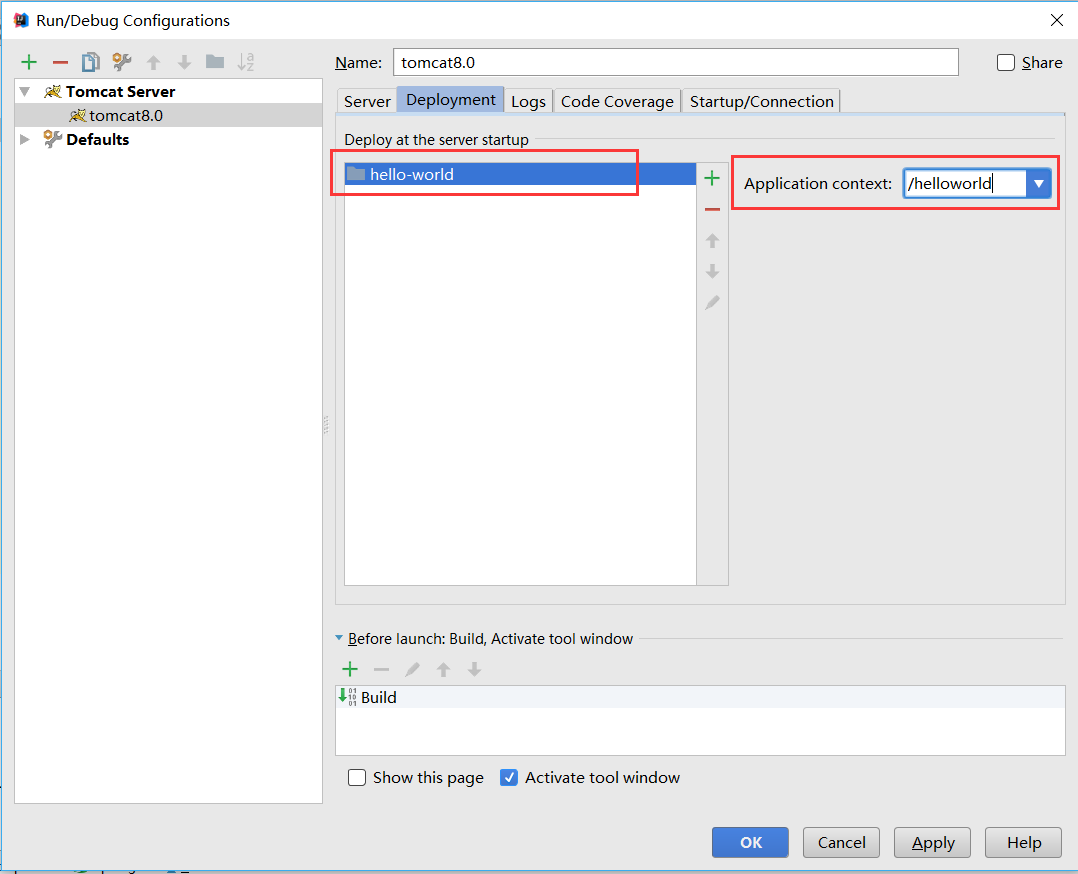




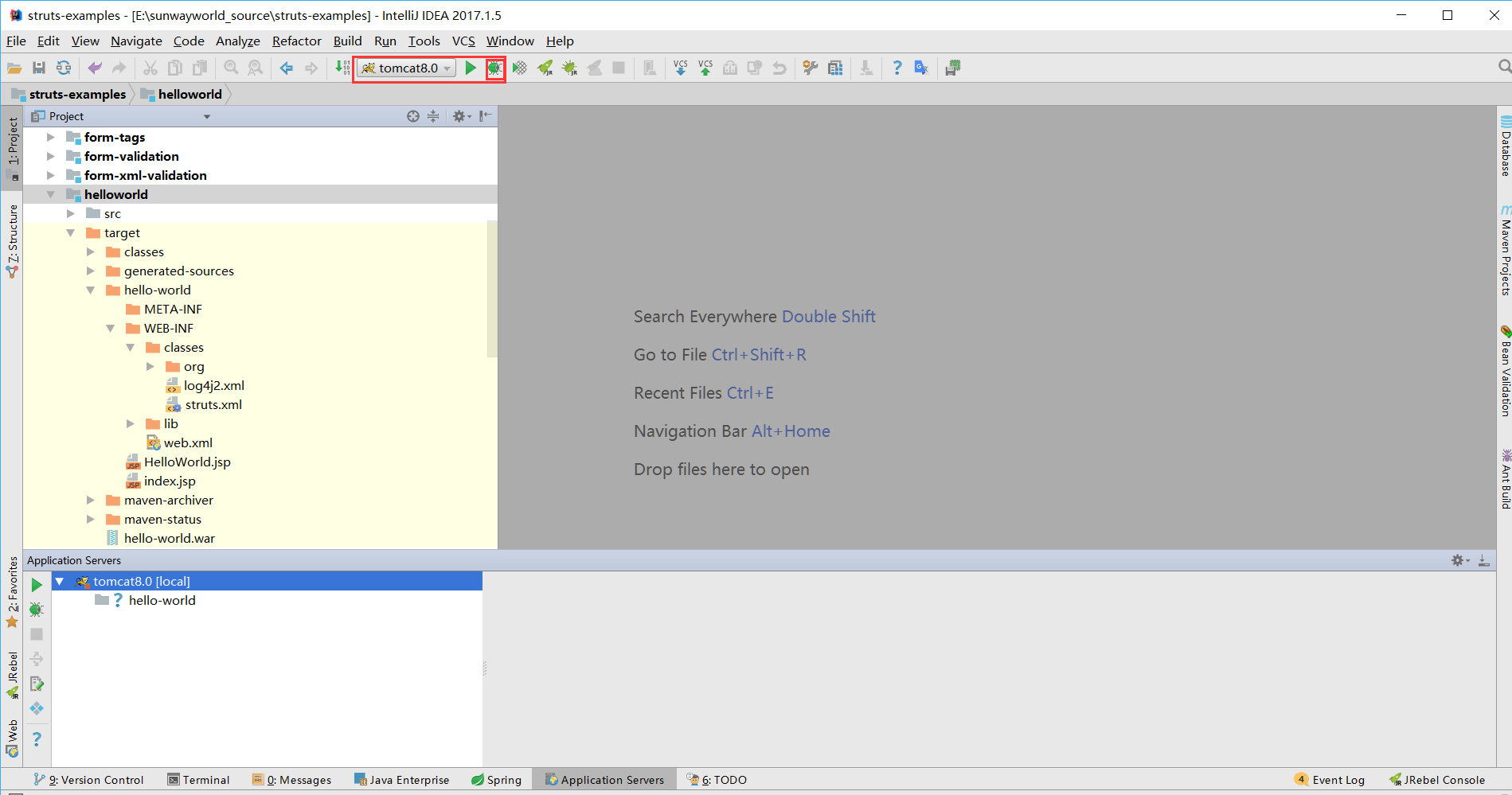




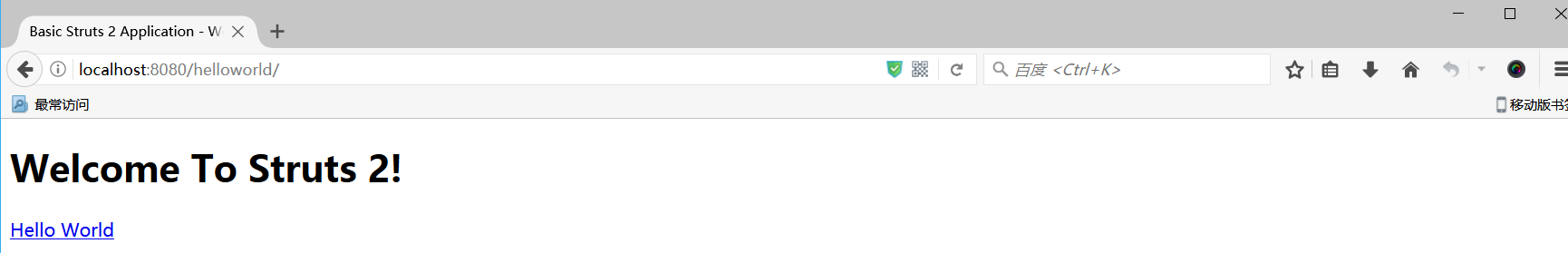




点击Apply



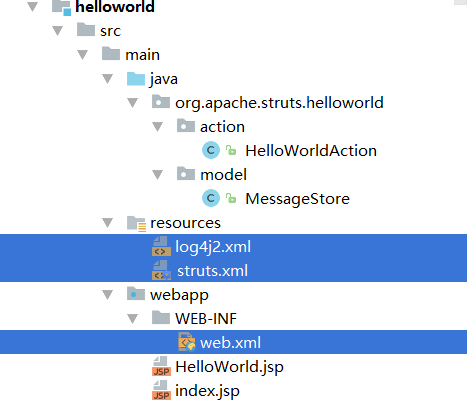
点击其他tomcat调试；



# 代码分析

代码分析会依据功能的实现为目的，尽量着力于了解每一个细节的实现。依照这个模式学习struts-examples中的所有例子。

## 配置文件分析



3个配置文件：

### log4j2.xml

日志配置文件，可以对系统中的日志进行管理

配置可以参考：

<?xml version="1.0" encoding="UTF-8"?>

<!--

status : 这个用于设置log4j2自身内部的信息输出,可以不设置,当设置成trace时,会看到log4j2内部各种详细输出

monitorInterval : Log4j能够自动检测修改配置文件和重新配置本身, 设置间隔秒数。

-->

<Configuration status="WARN" monitorInterval="600">

<Properties>

<!-- 配置日志文件输出目录 -->

<Property name="LOG\_HOME">/Users/hanhan.zhang/logs</Property>

</Properties>

<Appenders>

<!--这个输出控制台的配置-->

<Console name="Console" target="SYSTEM\_OUT">

<!-- 控制台只输出level及以上级别的信息(onMatch),其他的直接拒绝(onMismatch) -->

<ThresholdFilter level="trace" onMatch="ACCEPT" onMismatch="DENY"/>

<!-- 输出日志的格式 -->

<PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level %class{36} %L %M - %msg%xEx%n"/>

</Console>

<!-- 设置日志格式并配置日志压缩格式(service.log.年份.gz) -->

<RollingRandomAccessFile name="service\_appender"

immediateFlush="false" fileName="${LOG\_HOME}/service.log"

filePattern="${LOG\_HOME}/service.log.%d{yyyy-MM-dd}.log.gz">

<!--

%d{yyyy-MM-dd HH:mm:ss, SSS} : 日志生产时间

%p : 日志输出格式

%c : logger的名称

%m : 日志内容，即 logger.info("message")

%n : 换行符

%C : Java类名

%L : 日志输出所在行数

%M : 日志输出所在方法名

hostName : 本地机器名

hostAddress : 本地ip地址

-->

<PatternLayout>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level %class{36} %L %M -- %msg%xEx%n</pattern>

</PatternLayout>

<Policies>

<TimeBasedTriggeringPolicy interval="1" modulate="true" />

</Policies>

</RollingRandomAccessFile>

<!-- DEBUG日志格式 -->

<RollingRandomAccessFile name="service\_debug\_appender"

immediateFlush="false" fileName="${LOG\_HOME}/service.log"

filePattern="${LOG\_HOME}/service.log.%d{yyyy-MM-dd}.debug.gz">

<PatternLayout>

<pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level %class{36} %L %M -- %msg%xEx%n</pattern>

</PatternLayout>

<Policies>

<TimeBasedTriggeringPolicy interval="1" modulate="true" />

</Policies>

</RollingRandomAccessFile>

</Appenders>

<Loggers>

<!-- 配置日志的根节点 -->

<root level="debug">

<appender-ref ref="Console"/>

</root>

<!-- 第三方日志系统 -->

<logger name="org.springframework.core" level="info"/>

<logger name="org.springframework.beans" level="info"/>

<logger name="org.springframework.context" level="info"/>

<logger name="org.springframework.web" level="info"/>

<logger name="org.jboss.netty" level="warn"/>

<logger name="org.apache.http" level="warn"/>

<!-- 日志实例(info),其中'service-log'继承root,但是root将日志输出控制台,而'service-log'将日志输出到文件,通过属性'additivity="false"'将'service-log'的

的日志不再输出到控制台 -->

<logger name="service\_log" level="info" includeLocation="true" additivity="true">

<appender-ref ref="service\_appender"/>

</logger>

<!-- 日志实例(debug) -->

<logger name="service\_log" level="debug" includeLocation="true" additivity="false">

<appender-ref ref="service\_debug\_appender"/>

</logger>

</Loggers>

</Configuration>

### struts.xml

struts框架的核心配置文件，如果用到了该框架，则可以配置该文件

<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-2.0.dtd" >  
<struts>  
  
    <!-- include节点是struts2中组件化的方式 可以将每个功能模块独立到一个xml配置文件中 然后用include节点引用 -->  
    <include file="struts-default.xml"></include>  
      
      
    <!-- package提供了将多个Action组织为一个模块的方式  
        package的名字必须是唯一的 package可以扩展 当一个package扩展自  
        另一个package时该package会在本身配置的基础上加入扩展的package  
        的配置 父package必须在子package前配置   
        name：package名称  
        extends:继承的父package名称  
        abstract:设置package的属性为抽象的 抽象的package不能定义action 值true:false  
        namespace:定义package命名空间 该命名空间影响到url的地址，例如此命名空间为/test那么访问是的地址为http://localhost:8080/struts2/test/XX.action  
     -->  
    <package name="com.kay.struts2" extends="struts-default" namespace="/test">  
        <interceptors>  
            <!-- 定义拦截器   
                name:拦截器名称  
                class:拦截器类路径  
             -->  
            <interceptor name="timer" class="com.kay.timer"></interceptor>  
            <interceptor name="logger" class="com.kay.logger"></interceptor>  
            <!-- 定义拦截器栈 -->  
            <interceptor-stack name="mystack">  
                <interceptor-ref name="timer"></interceptor-ref>  
                <interceptor-ref name="logger"></interceptor-ref>  
            </interceptor-stack>  
        </interceptors>  
          
        <!-- 定义默认的拦截器 每个Action都会自动引用  
         如果Action中引用了其它的拦截器 默认的拦截器将无效 -->  
        <default-interceptor-ref name="mystack"></default-interceptor-ref>  
          
          
        <!-- 全局results配置 -->  
        <global-results>  
            <result name="input">/error.jsp</result>  
        </global-results>  
          
        <!-- Action配置 一个Action可以被多次映射(只要action配置中的name不同)  
             name：action名称  
             class: 对应的类的路径  
             method: 调用Action中的方法名  
        -->  
        <action name="hello" class="com.kay.struts2.Action.LoginAction">  
            <!-- 引用拦截器  
                name:拦截器名称或拦截器栈名称  
             -->  
            <interceptor-ref name="timer"></interceptor-ref>  
          
            <!-- 节点配置  
                name : result名称 和Action中返回的值相同  
                type : result类型 不写则选用superpackage的type struts-default.xml中的默认为dispatcher  
             -->  
         <result name="success" type="dispatcher">/talk.jsp</result>  
         <!-- 参数设置   
             name：对应Action中的get/set方法   
         -->  
         <param name="url">http://www.sina.com</param>  
        </action>  
    </package>  
</struts>

### web.xml

是web容器启动时会加载时的文件

 WEB容器的加载顺序是：ServletContext -> context-param -> listener -> filter -> servlet。在web.xml文件中最好按照这种顺序配置这些元素，以兼容较低版本的Tomcat。

相关参数说明：

*<?***xml version="1.0" encoding="UTF-8"***?>*<**web-app id="WebApp\_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd"**>  
<**display-name**>Hello World Struts 2</**display-name**>  
 <**welcome-file-list**>  
 <**welcome-file**>index.jsp</**welcome-file**>  
 </**welcome-file-list**>  
   
   
 <**filter**>  
 <**filter-name**>struts2</**filter-name**>  
 <**filter-class**>org.apache.struts2.dispatcher.filter.StrutsPrepareAndExecuteFilter</**filter-class**>  
 </**filter**>  
  
 <**filter-mapping**>  
 <**filter-name**>struts2</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
 </**filter-mapping**>  
   
</**web-app**>

**welcome-file-list，**用来配置首页用的：

<welcome-file-list>

<welcome-file>index1.jsp</welcome-file>

<welcome-file>index2.jsp</welcome-file>

<welcome-file>index3.jsp</welcome-file>

<welcome-file>index4.jsp</welcome-file>

<welcome-file>/target/redirectAndFoward.jsp</welcome-file>

</welcome-file-list>

**error-page，**当HTTP返回指定状态码的时候，容器将此次请求转发到配置的指定页面

<error-page>

<error-code>400</error-code>

<location>/filter/error.jsp</location>

</error-page>

<error-page>

<error-code>404</error-code>

<location>/filter/error.jsp</location>

</error-page>

<error-page>

<error-code>500</error-code>

<location>/filter/error.jsp</location>

</error-page>

**Filter，过滤器，执行的顺序就是filter定义的顺序**

**Servlet，**

<servlet>

<servlet-name>startUpServlet</servlet-name>

<servlet-class>com.xrq.servlet.StartUpServlet</servlet-class>

<init-param>

<param-name>Name</param-name>

<param-value>123</param-value>

</init-param>

<init-param>

<param-name>Age</param-name>

<param-value>456</param-value>

</init-param>

<load-on-startup>8</load-on-startup>

</servlet>

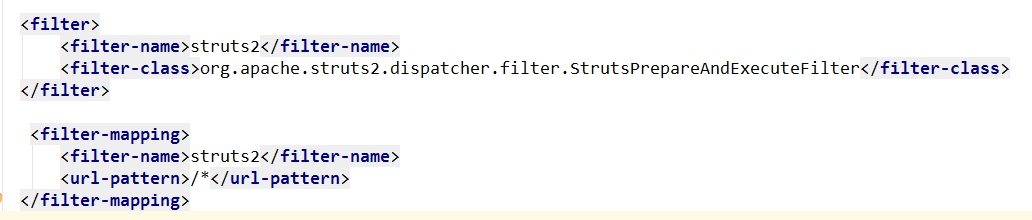
load-on-startup，当值为0或者大于0时，表示容器在应用启动时就加载并初始化这个servlet，正数值越小，该servlet的优先级就越高，应用启动时就越先加载

**listener，监听器**

**context-param，全局上下文参数**

**session-config，**配置session失效时间的

## Struts的StrutsPrepareAndExecuteFilter



Struts的核心过滤器，里面首先定义了一个

初始化读取配置项方法：

**public void** init(FilterConfig filterConfig) **throws** ServletException {  
 InitOperations init = createInitOperations();  
 Dispatcher dispatcher = **null**;  
 **try** {  
 FilterHostConfig config = **new** FilterHostConfig(filterConfig);  
 init.initLogging(config);  
 dispatcher = init.initDispatcher(config);  
 init.initStaticContentLoader(config, dispatcher);  
  
 **prepare** = createPrepareOperations(dispatcher);  
 **execute** = createExecuteOperations(dispatcher);  
 **this**.**excludedPatterns** = init.buildExcludedPatternsList(dispatcher);  
  
 postInit(dispatcher, filterConfig);  
 } **finally** {  
 **if** (dispatcher != **null**) {  
 dispatcher.cleanUpAfterInit();  
 }  
 init.cleanup();  
 }  
}

其中的filterConfig就是对应加载配置项

，根据这个配置项创建了dispatcher工具类，再根据dispatcher创建了**prepare，execute；**

**过滤方法：**

**public void** doFilter(ServletRequest req, ServletResponse res, FilterChain chain) **throws** IOException, ServletException {  
  
 HttpServletRequest request = (HttpServletRequest) req;  
 HttpServletResponse response = (HttpServletResponse) res;  
  
 **try** {  
 String uri = RequestUtils.*getUri*(request);  
 **if** (**excludedPatterns** != **null** && **prepare**.isUrlExcluded(request, **excludedPatterns**)) {  
 ***LOG***.trace(**"Request {} is excluded from handling by Struts, passing request to other filters"**, uri);  
 chain.doFilter(request, response);  
 } **else** {  
 ***LOG***.trace(**"Checking if {} is a static resource"**, uri);  
 **boolean** handled = **execute**.executeStaticResourceRequest(request, response);  
 **if** (!handled) {  
 ***LOG***.trace(**"Assuming uri {} as a normal action"**, uri);  
 **prepare**.setEncodingAndLocale(request, response);  
 **prepare**.createActionContext(request, response);  
 **prepare**.assignDispatcherToThread();  
 request = **prepare**.wrapRequest(request);  
 ActionMapping mapping = **prepare**.findActionMapping(request, response, **true**);  
 **if** (mapping == **null**) {  
 ***LOG***.trace(**"Cannot find mapping for {}, passing to other filters"**, uri);  
 chain.doFilter(request, response);  
 } **else** {  
 ***LOG***.trace(**"Found mapping {} for {}"**, mapping, uri);  
 **execute**.executeAction(request, response, mapping);  
 }  
 }  
 }  
 } **finally** {  
 **prepare**.cleanupRequest(request);  
 }  
}

**使用prepare设置了编码和语言，设置了上下文，并且将每一个请求都保存到持有一个，根据ActionMapping匹配action**

## Action介绍

类似于springmvc的controller,控制器