课程目标

1、高仿真手写Spring AOP。
2、用30个类搭建基本框架，满足核心功能。

内容定位

在完全掌握Spring系统结构、实现原理，在理解设计模式的基础上，自己动手写一个高仿真版本的Spring框架，以达到透彻理解Spring的目的，感受作者创作意图。

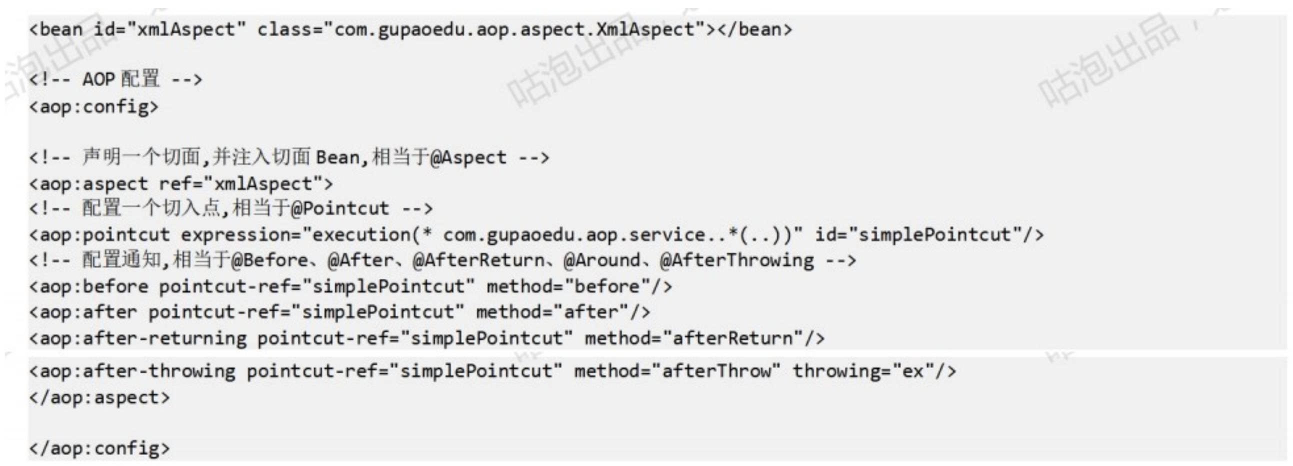
1. 基础配置

在application.properties中增加如下自定义配置：

```
# 多切面配置可以在key前面加前缀
# 例如: aspect.logAspect.

# 切面表达式, expression#
pointCut=public .* cn.sitedev.demo.service..*Service..*(.*)
```

```
# 切面类#
aspectClass=cn.sitedev.demo.aspect.LogAspect
# 切面前置通知#
aspectBefore=before
# 切面后置通知#
aspectAfter=after
# 切面异常通知#
aspectAfterThrow=afterThrowing
# 切面异常类型#
aspectAfterThrowingName=java.lang.Exception
```

下面是Spring AOP的原生配置，为了方便操作，用properties文件来代替xml，以简化操作：



```
<bean id="xmlAspect" class="com.gupaoedu.aop.aspect.XmlAspect"></bean>

<!-- AOP 配置 -->
<aop:config>

<!-- 声明一个切面,并注入切面 Bean,相当于@Aspect -->
<aop:aspect ref="xmlAspect">
<!-- 配置一个切入点,相当于@Pointcut -->
<aop:pointcut expression="execution(* com.gupaoedu.aop.service..*(..))" id="simplePointcut"/>
<!-- 配置通知,相当于@Before、@After、@AfterReturn、@Around、@AfterThrowing -->
<aop:before pointcut-ref="simplePointcut" method="before"/>
<aop:after pointcut-ref="simplePointcut" method="after"/>
<aop:after-returning pointcut-ref="simplePointcut" method="afterReturn"/>
<aop:after-throwing pointcut-ref="simplePointcut" method="afterThrow" throwing="ex"/>
</aop:aspect>

</aop:config>
```

2. 完成AOP顶层设计

2.1. MyAopProxy代理顶层接口定义

```java
package cn.sitedev.spring.framework.aop;

public interface MyAopProxy {
    Object getProxy();

    Object getProxy(ClassLoader classLoader);
}
```

2.2. MyCglibAopProxy基于Cglib的动态代理实现

2.3. MyJdkDynamicAopProxy基于JDK动态代理实现

```java
package cn.sitedev.spring.framework.aop;

import cn.sitedev.spring.framework.aop.aspect.MyAdvice;
import cn.sitedev.spring.framework.aop.support.MyAdvisedSupport;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;
import java.util.Map;

public class MyJdkDynamicAopProxy implements MyAopProxy, InvocationHandler {
    private MyAdvisedSupport advised;

    public MyJdkDynamicAopProxy(MyAdvisedSupport config) {
        this.advised = config;
    }

    @Override
    public Object getProxy() {
        return getProxy(this.advised.getTargetClass().getClassLoader());
    }

    @Override
    public Object getProxy(ClassLoader classLoader) {
        return Proxy.newProxyInstance(classLoader, this.advised.getTargetClass().getInt
                this);
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        Map<String, MyAdvice> advices =
                this.advised.getInterceptorsAndDynamicInterceptionAdvice(method,
                        this.advised.getTargetClass());

        Object returnValue = null;
        try {
            invokeAdvice(advices.get("before"));
            returnValue = method.invoke(this.advised.getTarget(), args);
            invokeAdvice(advices.get("after"));
        } catch (Exception e) {
            invokeAdvice(advices.get("afterThrow"));
```

```java
                throw e;
            }
            return returnValue;
        }

        private void invokeAdvice(MyAdvice advice) {
            try {
                advice.getAdviceMethod().invoke(advice.getAspect());

            } catch (IllegalAccessException e) {
                e.printStackTrace();
            } catch (InvocationTargetException e) {
                e.printStackTrace();
            }

        }
}
```

2.4. MyAdvisedSupport配置解析

```java
package cn.sitedev.spring.framework.aop.support;

import cn.sitedev.spring.framework.aop.aspect.MyAdvice;
import cn.sitedev.spring.framework.aop.config.MyAopConfig;
import lombok.Data;

import java.lang.reflect.Method;
import java.util.HashMap;
import java.util.Map;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

/**
 * 解析AOP配置的工具类
 */
@Data
public class MyAdvisedSupport {
    private Class<?> targetClass;
    private Object target;
    private MyAopConfig config;
    private Pattern pointCutClassPattern;
    private transient Map<Method, Map<String, MyAdvice>> methodCache;

    public MyAdvisedSupport(MyAopConfig config) {
        this.config = config;
    }

    public Map<String, MyAdvice> getInterceptorsAndDynamicInterceptionAdvice(Method met
                                                                             Class<?> t
        Map<String, MyAdvice> cached = methodCache.get(method);
        if (cached == null) {
            Method m = targetClass.getMethod(method.getName(), method.getParameterTypes
            cached = methodCache.get(m);

            // 底层逻辑，对代理方法进行一个兼容处理
            this.methodCache.put(m, cached);
        }
        return cached;
    }

    public void setTargetClass(Class<?> targetClass) {
        this.targetClass = targetClass;
        parse();
    }

    // 解析配置文件中的方法
    private void parse() {
        String pointCut = config.getPointCut().replaceAll("\\\\.", "\\\\.").replaceAll(
                ".*").replaceAll("\\(", "\\\\(").replaceAll("\\)", "\\\\)");

        // 保存专门匹配Class的正则
        String pointCutForClassRegex = pointCut.substring(0, pointCut.lastIndexOf("\\("
        pointCutClassPattern =
                Pattern.compile("class " + pointCutForClassRegex.substring(pointCutForC

        // 享元的共享池
        methodCache = new HashMap<>();
        // 保存专门匹配方法的正则
        Pattern pointCutPattern = Pattern.compile(pointCut);
        try {
            Class aspectClass = Class.forName(this.config.getAspectClass());
            Map<String, Method> aspectMethods = new HashMap<>();
            for (Method method : aspectClass.getMethods()) {
                aspectMethods.put(method.getName(), method);
            }

            for (Method method : this.targetClass.getMethods()) {
                String methodString = method.toString();
                if (methodString.contains("throws")) {
                    methodString =
                            methodString.substring(0, methodString.lastIndexOf("throws"
                }

                Matcher matcher = pointCutPattern.matcher(methodString);
                if (matcher.matches()) {
                    // 执行器链
                    Map<String, MyAdvice> advices = new HashMap<>();
                    // 把每个方法包装成MethodInterceptor
                    if (config.getAspectBefore() != null && !"".equals(config.getAspect
                        advices.put("before", new MyAdvice(aspectClass.newInstance(),
                                aspectMethods.get(config.getAspectBefore())));
                    }
                    if (config.getAspectAfter() != null && !"".equals(config.getAspectA
                        advices.put("after", new MyAdvice(aspectClass.newInstance(),
                                aspectMethods.get(config.getAspectAfter())));
                    }
                    if (config.getAspectAfterThrow() != null && !"".equals(config.getAs
                        MyAdvice advice = new MyAdvice(aspectClass.newInstance(),
                                aspectMethods.get(config.getAspectAfterThrow()));
                        advice.setThrowName(config.getAspectAfterThrowingName());
                        advices.put("afterThrow", advice);
                    }

                    // 跟目标代理类的业务方法和Advices建立一对多关联关系，以便在Proxy类
                    methodCache.put(method, advices);
                }
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (InstantiationException e) {
            e.printStackTrace();
        }
    }

    // 根据一个目标代理类的方法，获得其对应的通知
    public Map<String, MyAdvice> getAdvices(Method method, Object o) throws Exception {
        // 享元模式的应用
        Map<String, MyAdvice> cache = methodCache.get(method);
        if (cache == null) {
            Method m = targetClass.getMethod(method.getName(), method.getParameterTypes
            cache = methodCache.get(method);
            this.methodCache.put(m, cache);
        }
        return cache;
    }

    // 首先IOC中的对象对应初始化时调用，决定要不要生成代理类的逻辑
    public boolean pointCutMatch() {
        return pointCutClassPattern.matcher(this.targetClass.toString()).matches();
    }

}
```

2.5. MyAdvice通知接口定义

```java
package cn.sitedev.spring.framework.aop.aspect;

import lombok.Data;

import java.lang.reflect.Method;

/**
 * 用于通知回调
 */
@Data
public class MyAdvice {
    private Object aspect;
    private Method adviceMethod;
    private String throwName;

    public MyAdvice(Object aspect, Method adviceMethod) {
        this.aspect = aspect;
        this.adviceMethod = adviceMethod;
    }
}
```

2.6. MyAopConfig封装配置

```java
package cn.sitedev.spring.framework.aop.config;

import lombok.Data;

@Data
public class MyAopConfig {
    private String pointCut;
    private String aspectClass;
    private String aspectBefore;
    private String aspectAfter;
    private String aspectAfterThrow;
    private String aspectAfterThrowingName;
}
```

3. 设计AOP基础实现

3.1. 接入getBean()方法与IOC容器衔接

找到MyApplicationContext的getBean()方法，我们知道 getBean()中负责Bean初始化的方法其实就是instantiateBean()，我们在初始化时就可以确定是否返回原生Bean 还是Proxy Bean。代码实现如下：

```java
    // 创建真正的实例对象
    private Object instantiateBean(String beanName, MyBeanDefinition beanDefinition) {
        String className = beanDefinition.getBeanClassName();
        Object instance = null;
        try {
            if (this.factoryBeanObjectCache.containsKey(beanName)) {
                instance = this.factoryBeanObjectCache.get(beanName);
            } else {
                Class<?> clazz = Class.forName(className);
                // 2. 默认的类名首字母小写
                instance = clazz.newInstance();

                //-------------------AOP开始--------------------
                MyAdvisedSupport config = instantionAopConfig(beanDefinition);
                config.setTargetClass(clazz);
                config.setTarget(instance);

                // 判断规则，要不要生成代理类，如果要就覆盖原生对象
                // 如果不要就不做任何处理，返回原生对象
                if (config.pointCutMatch()) {
                    instance = new MyJdkDynamicAopProxy(config).getProxy();
                }

                //-----------------AOP结束---------------------
                // 符合PointCut的规则的话,将会使用代理对象
                this.factoryBeanObjectCache.put(beanName, instance);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return instance;
    }

    private MyAdvisedSupport instantionAopConfig(MyBeanDefinition beanDefinition) {
        MyAopConfig config = new MyAopConfig();
        config.setPointCut(this.reader.getConfig().getProperty("pointCut"));
        config.setAspectClass(this.reader.getConfig().getProperty("aspectClass"));
        config.setAspectBefore(this.reader.getConfig().getProperty("aspectBefore"));
        config.setAspectAfter(this.reader.getConfig().getProperty("aspectAfter"));
        config.setAspectAfterThrow(this.reader.getConfig().getProperty("aspectAfterThro
        config.setAspectAfterThrowingName(this.reader.getConfig().getProperty(
                "aspectAfterThrowingName"));
        return new MyAdvisedSupport(config);
    }
```

4. 织入业务代码

4.1. LogAspect自定义切面配置

```java
package cn.sitedev.demo.aspect;

```

```java
public class LogAspect {
    // 在调用一个方法前，执行before方法
    public void before() {
        System.out.println("Invoker Before Method...");
    }

    // 在调用一个方法后，执行after方法
    public void after() {
        System.out.println("Invoker After Method...");
    }

    // 在调用一个方法发生异常时，执行afterThrowing方法
    public void afterThrowing() {
        System.out.println("Invoker AfterThowing Method...");
    }
}
```

4.2. ModifyService切面业务逻辑实现

IModifyService 业务接口定义

```java
package cn.sitedev.demo.service;

/**
 * 增删改业务
 */
public interface IModifyService {
    // 增加
    String add(String name, String addr) throws Exception;

    // 修改
    String edit(Integer id, String name);

    // 删除
    String remove(Integer id);
}
```

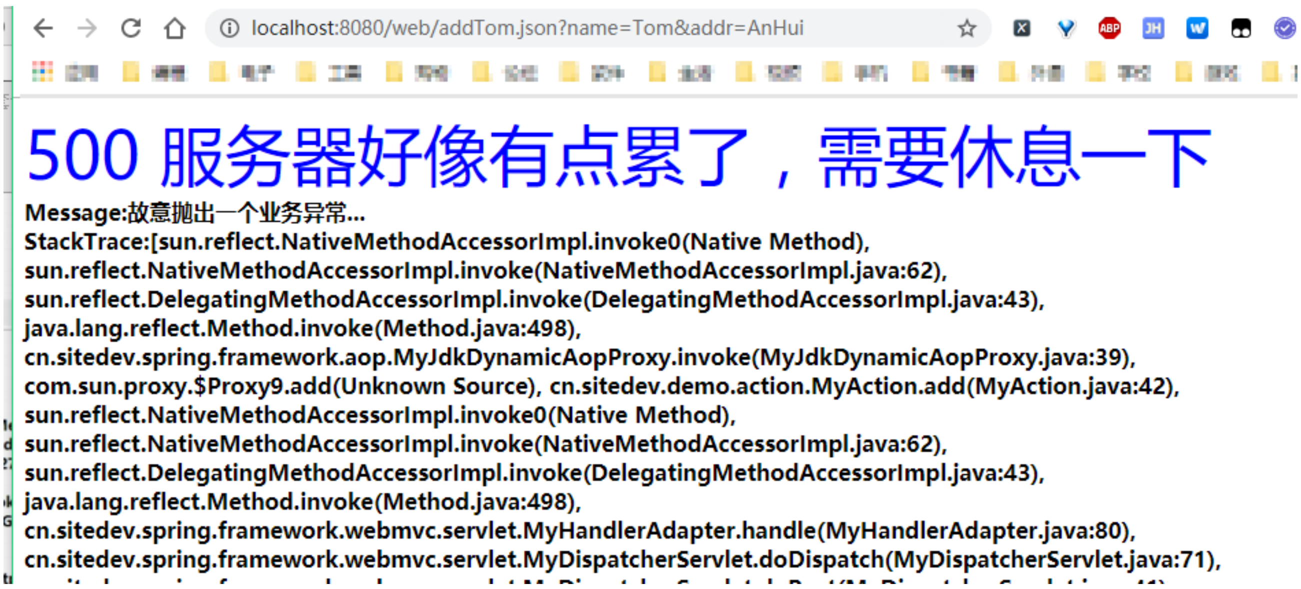
ModifyService 切面业务逻辑实现

```
```

```java
package cn.sitedev.demo.service.impl;

import cn.sitedev.demo.service.IModifyService;

/**
 * 增删改业务
 */
public class ModifyService implements IModifyService {
    @Override
    public String add(String name, String addr) throws Exception {
        throw new Exception("故意抛出一个业务异常...");
//        return "ModifyService add: name = " + name + ", addr = " + addr;
    }

    @Override
    public String edit(Integer id, String name) {
        return "ModifyService edit: id = " + id + ", name = " + name;
    }

    @Override
    public String remove(Integer id) {
        return "ModifyService remove: id = " + id;
    }
}
```

5. 最终效果演示与总结

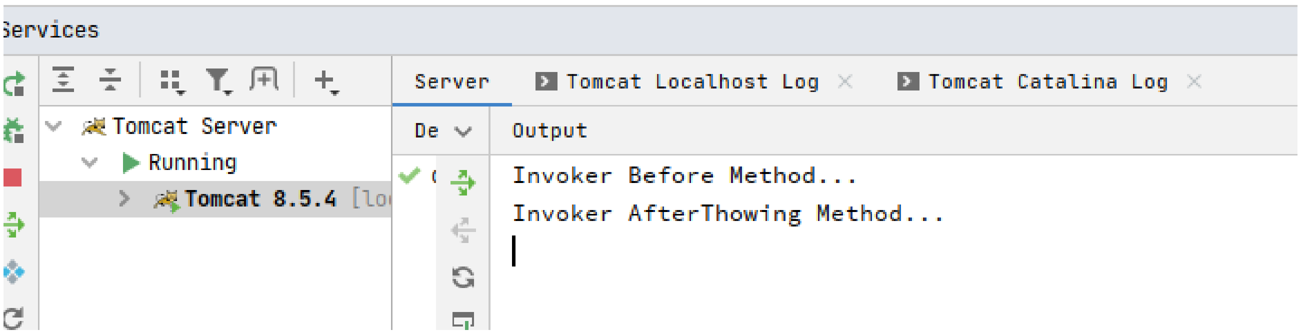
输入: http://localhost:8080/web/addTom.json?name=Tom&addr=AnHui



500 服务器好像有点累了，需要休息一下

```
Message:故意抛出一个业务异常...
StackTrace:[sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method),
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62),
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43),
java.lang.reflect.Method.invoke(Method.java:498),
cn.sitedev.spring.framework.aop.MyJdkDynamicAopProxy.invoke(MyJdkDynamicAopProxy.java:39),
com.sun.proxy.$Proxy9.add(Unknown Source), cn.sitedev.demo.action.MyAction.add(MyAction.java:42),
sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method),
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62),
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43),
java.lang.reflect.Method.invoke(Method.java:498),
cn.sitedev.spring.framework.webmvc.servlet.MyHandlerAdapter.handle(MyHandlerAdapter.java:80),
cn.sitedev.spring.framework.webmvc.servlet.MyDispatcherServlet.doDispatch(MyDispatcherServlet.java:71),
```

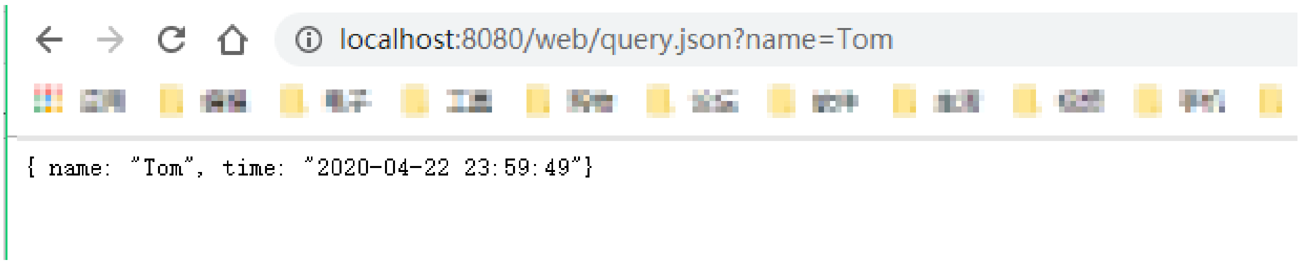
控制台输出:



```
Invoker Before Method...
Invoker AfterThowing Method...
```

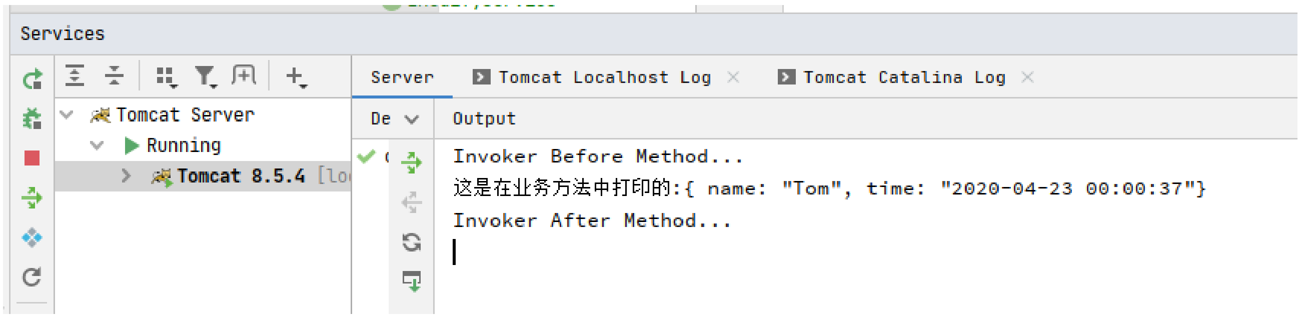
从控制台输出可以看到切面已经生效了。

下面再做一个测试，输入：http://localhost:8080/web/query.json?name=Tom



```
{ name: "Tom", time: "2020-04-22 23:59:49"}
```

控制台输出:

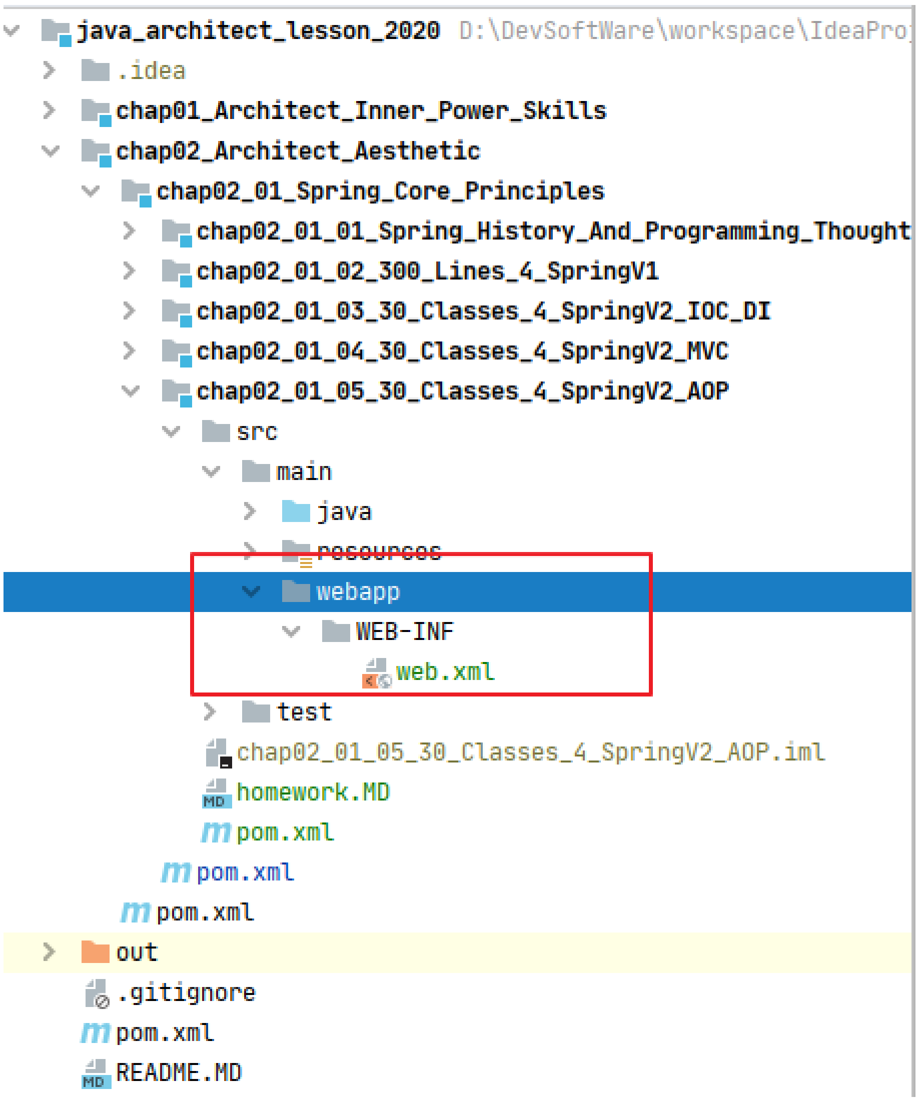


```
Invoker Before Method...
这是在业务方法中打印的:{ name: "Tom", time: "2020-04-23 00:00:37"}
Invoker After Method...
```

至此AOP模块就大功告成。

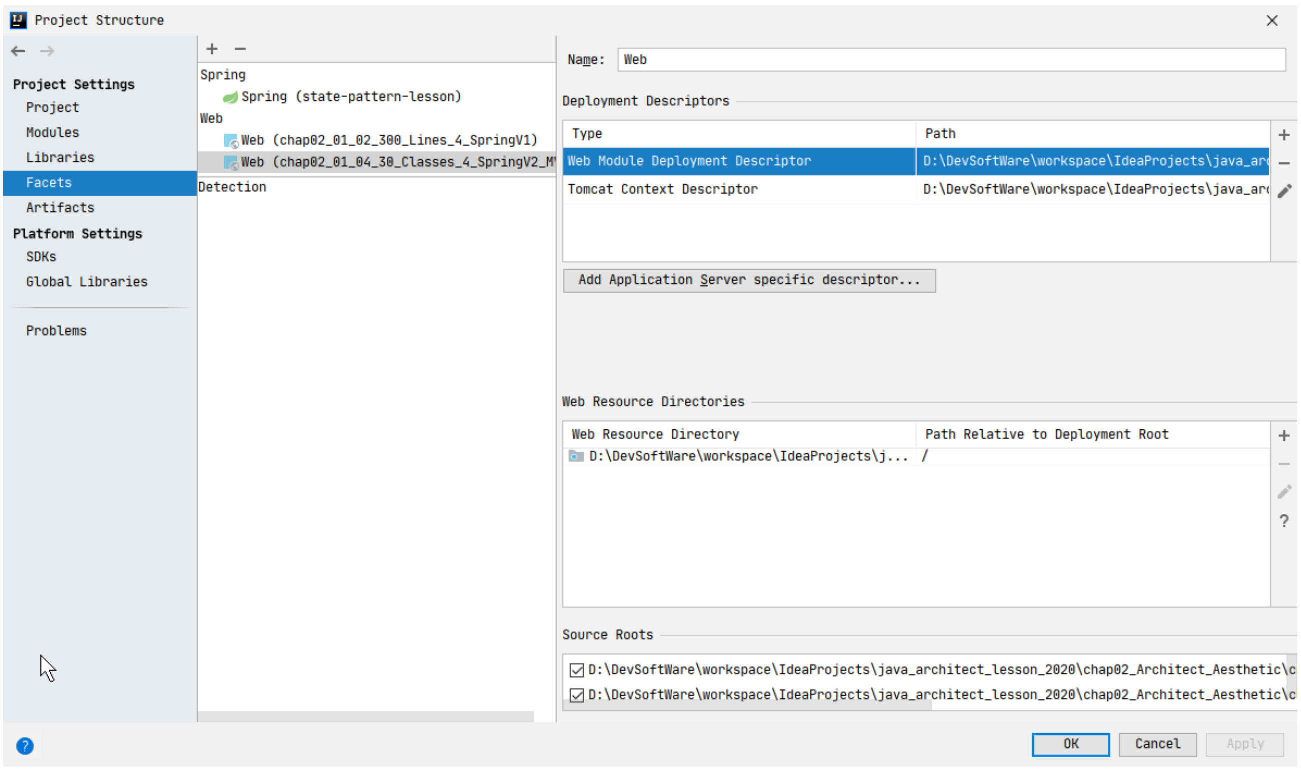
6. 补充:在tomcat中启动web应用

项目结构



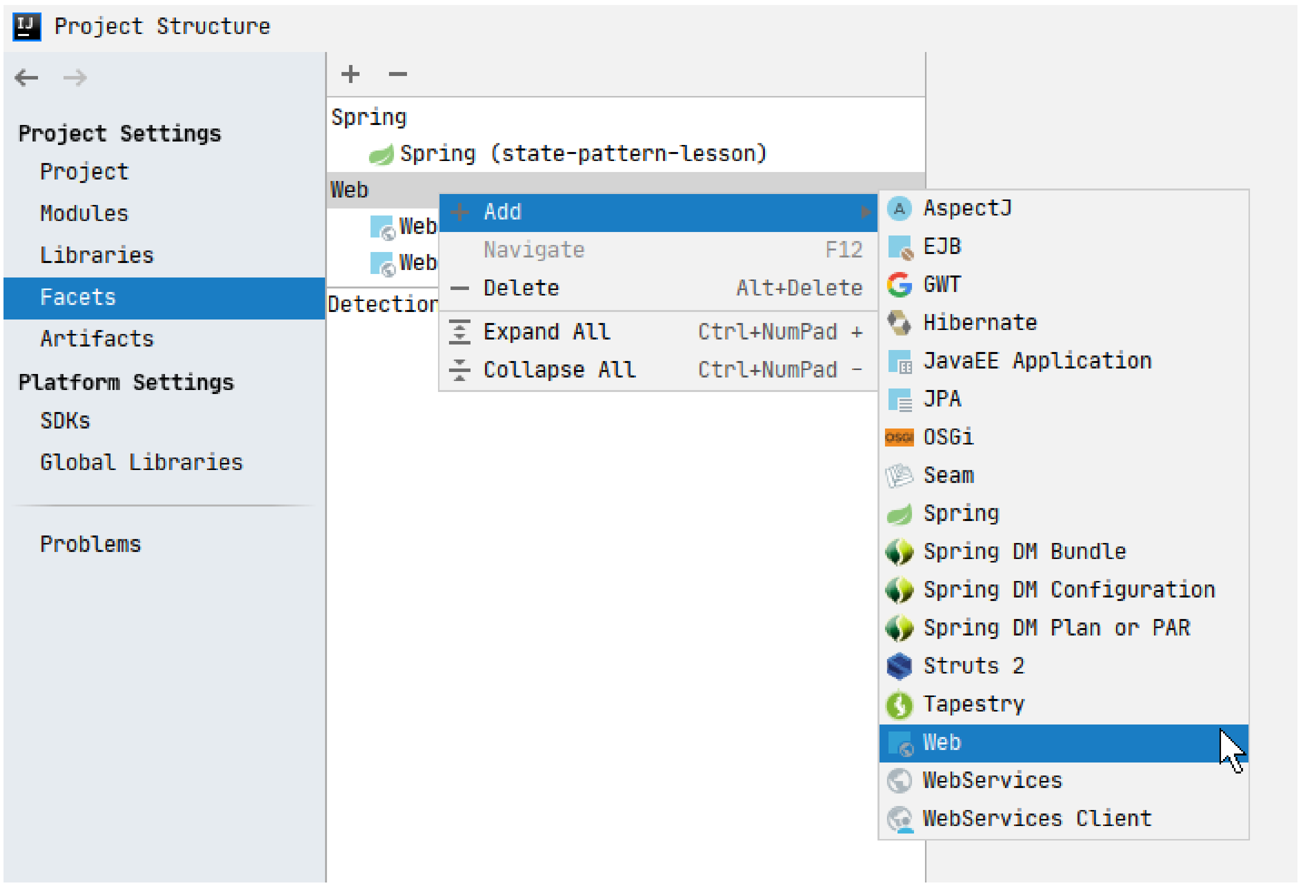
```
java_architect_lesson_2020  D:\DevSoftWare\workspace\IdeaPro
  .idea
  chap01_Architect_Inner_Power_Skills
  chap02_Architect_Aesthetic
    chap02_01_Spring_Core_Principles
      chap02_01_01_Spring_History_And_Programming_Thought
      chap02_01_02_300_Lines_4_SpringV1
      chap02_01_03_30_Classes_4_SpringV2_IOC_DI
      chap02_01_04_30_Classes_4_SpringV2_MVC
      chap02_01_05_30_Classes_4_SpringV2_AOP
        src
          main
            java
            resources
            webapp
              WEB-INF
                web.xml
          test
        chap02_01_05_30_Classes_4_SpringV2_AOP.iml
        homework.MD
        pom.xml
      pom.xml
    pom.xml
  out
  .gitignore
  pom.xml
  README.MD
```

File -> Project Structure -> Facets

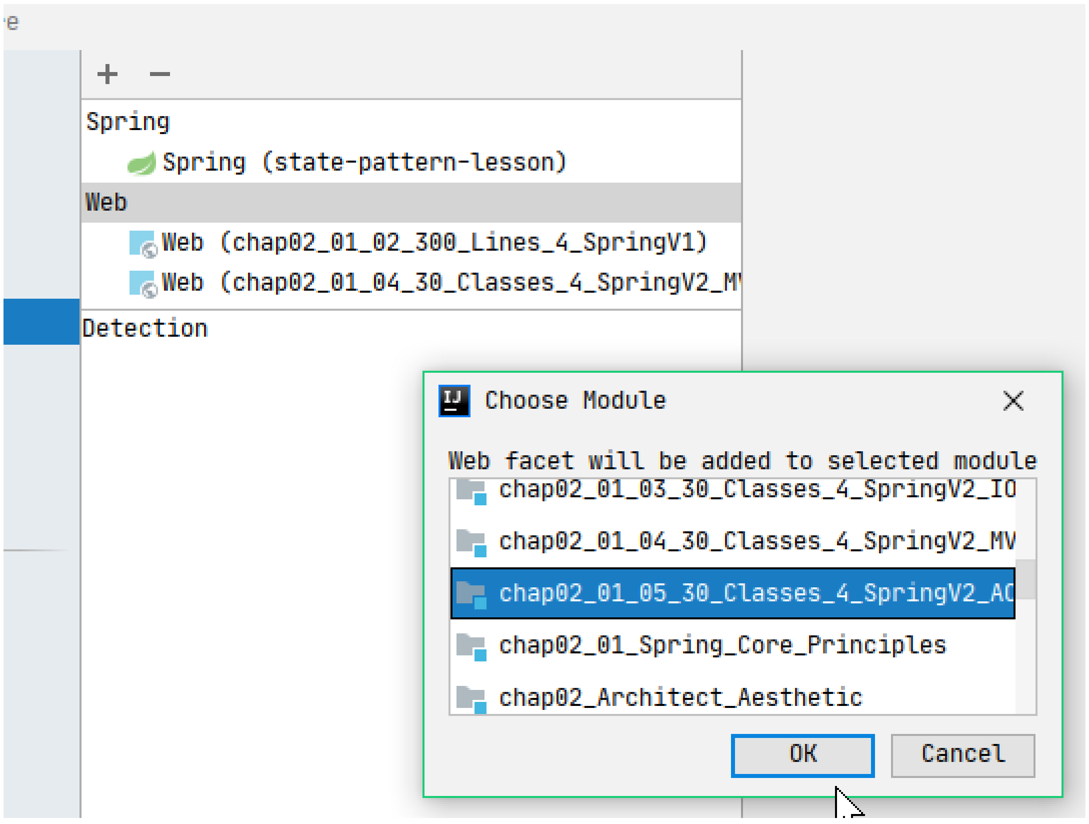


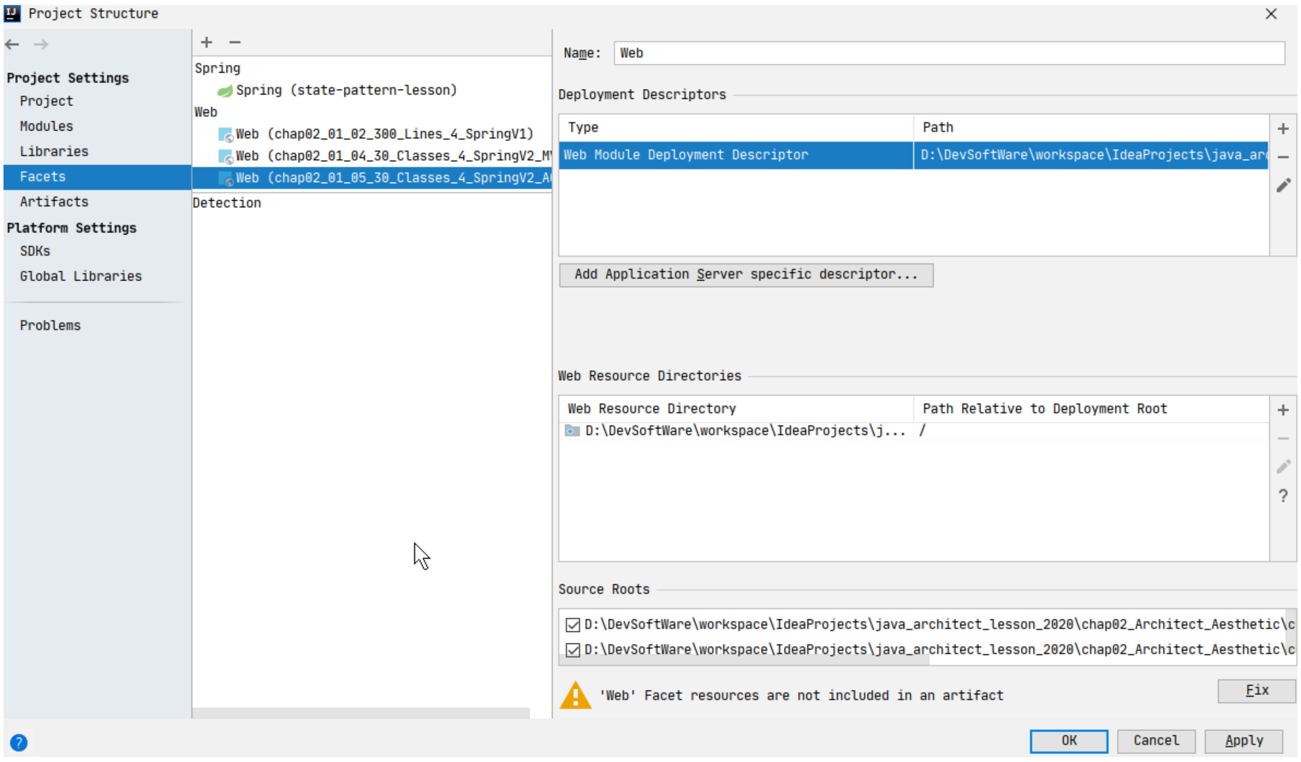
```
Project Structure

Project Settings
  Project
  Modules
  Libraries
  Facets
  Artifacts
Platform Settings
  SDKs
  Global Libraries

  Problems

Spring
  Spring (state-pattern-lesson)
Web
  Web (chap02_01_02_300_Lines_4_SpringV1)
  Web (chap02_01_04_30_Classes_4_SpringV2_M
Detection

Name: Web

Deployment Descriptors
Type                                Path
Web Module Deployment Descriptor    D:\DevSoftWare\workspace\IdeaProjects\java_ar
Tomcat Context Descriptor           D:\DevSoftWare\workspace\IdeaProjects\java_ar

Add Application Server specific descriptor...

Web Resource Directories
Web Resource Directory                        Path Relative to Deployment Root
D:\DevSoftWare\workspace\IdeaProjects\j...    /

Source Roots
☑ D:\DevSoftWare\workspace\IdeaProjects\java_architect_lesson_2020\chap02_Architect_Aesthetic\c
☑ D:\DevSoftWare\workspace\IdeaProjects\java_architect_lesson_2020\chap02_Architect_Aesthetic\c

OK    Cancel    Apply
```

Web 鼠标右键 -> Add -> Web



Choose Module -> chap02_01_05_30_Classes_4_SpringV2_AOP -> OK


Spring
Spring (state-pattern-lesson)
Web
Web (chap02_01_02_300_Lines_4_SpringV1)
Web (chap02_01_04_30_Classes_4_SpringV2_M
Detection
Choose Module
Web facet will be added to selected module
chap02_01_03_30_Classes_4_SpringV2_IO
chap02_01_04_30_Classes_4_SpringV2_MV
chap02_01_05_30_Classes_4_SpringV2_AC
chap02_01_Spring_Core_Principles
chap02_Architect_Aesthetic
OK
Cancel



Project Structure

Project Settings: Project, Modules, Libraries, Facets, Artifacts
Platform Settings: SDKs, Global Libraries
Problems

Spring
- Spring (state-pattern-lesson)

Web
- Web (chap02_01_02_300_Lines_4_SpringV1)
- Web (chap02_01_04_30_Classes_4_SpringV2_M
- Web (chap02_01_05_30_Classes_4_SpringV2_A

Detection

Name: Web

Deployment Descriptors

Type	Path
Web Module Deployment Descriptor	D:\DevSoftWare\workspace\IdeaProjects\java_ar

Add Application Server specific descriptor...

Web Resource Directories

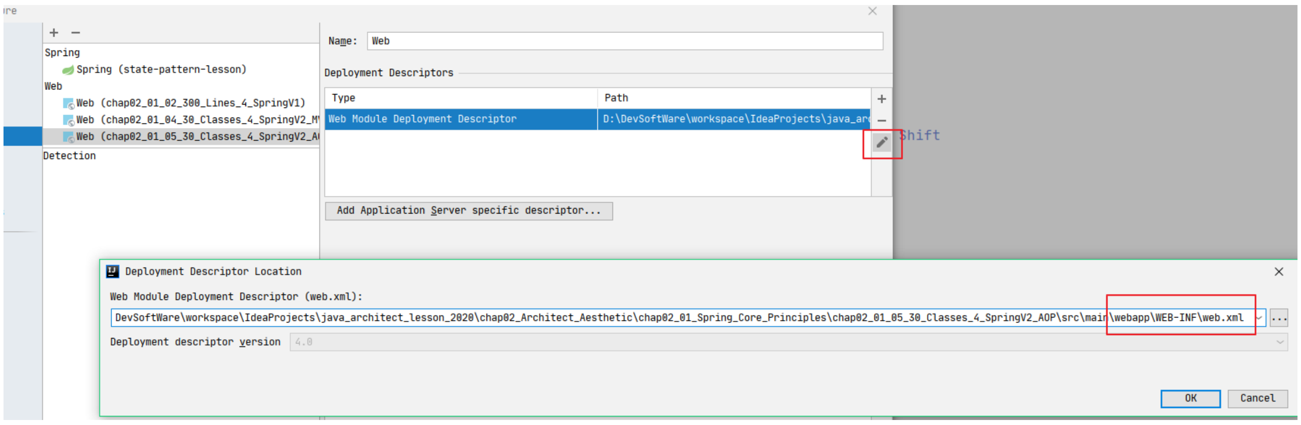
Web Resource Directory	Path Relative to Deployment Root
D:\DevSoftWare\workspace\IdeaProjects\j...	/

Source Roots

☑ D:\DevSoftWare\workspace\IdeaProjects\java_architect_lesson_2020\chap02_Architect_Aesthetic\c
☑ D:\DevSoftWare\workspace\IdeaProjects\java_architect_lesson_2020\chap02_Architect_Aesthetic\c

'Web' Facet resources are not included in an artifact

Deployment Descriptors -> Edit -> web.xml路径改为 ...\webapp\WEB-INF\web.xml -> OK

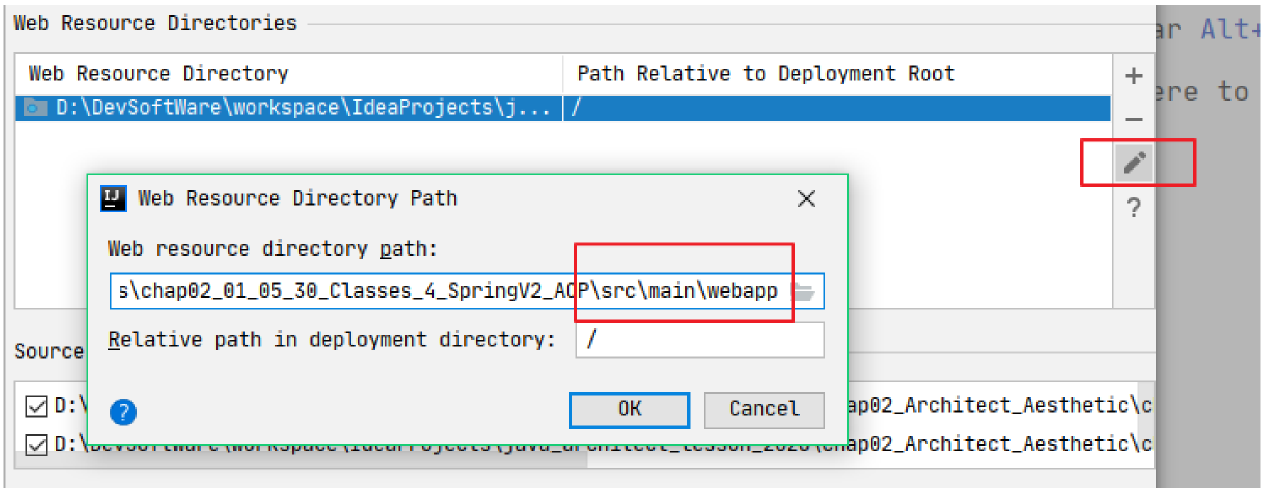


Deployment Descriptor Location

Web Module Deployment Descriptor (web.xml):
DevSoftWare\workspace\IdeaProjects\java_architect_lesson_2020\chap02_Architect_Aesthetic\chap02_01_Spring_Core_Principles\chap02_01_05_30_Classes_4_SpringV2_AOP\src\main\webapp\WEB-INF\web.xml

Deployment descriptor version 4.0

Web Resource Directories -> Edit -> web resource directory path 改为 ...\src\main\webapp -> OK

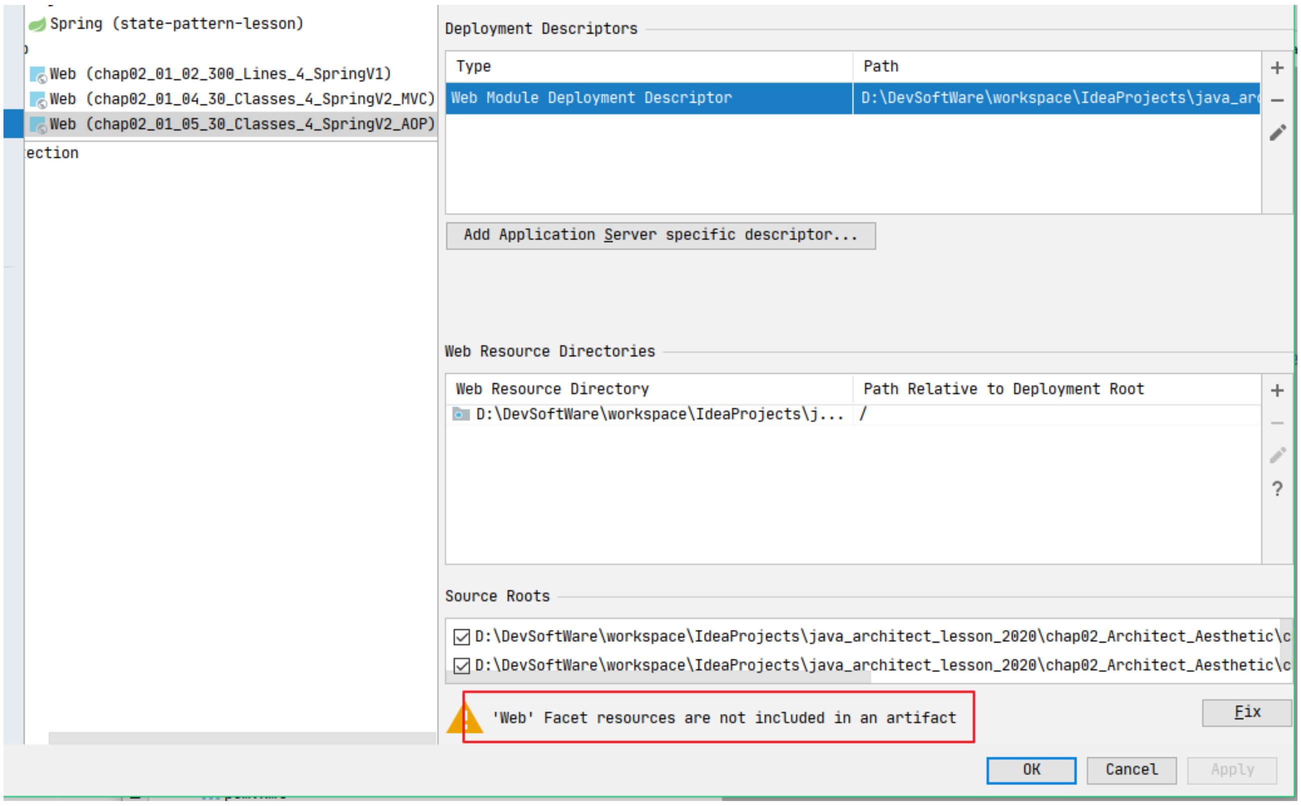


Web Resource Directory Path

Web resource directory path: s\chap02_01_05_30_Classes_4_SpringV2_AOP\src\main\webapp

Relative path in deployment directory: /

点击 "'Web' Facet resources are not included in an artifact" 右侧的Fix -> Create Artifact

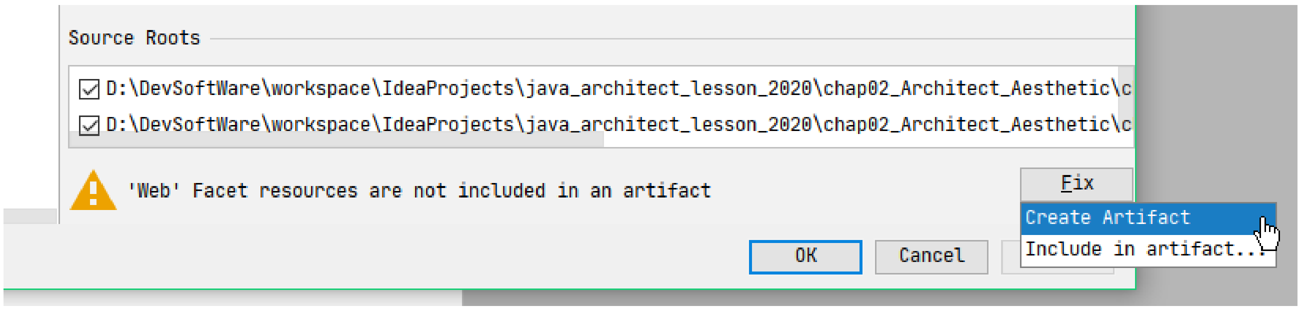

Spring (state-pattern-lesson)
Web (chap02_01_02_300_Lines_4_SpringV1)
Web (chap02_01_04_30_Classes_4_SpringV2_MVC)
Web (chap02_01_05_30_Classes_4_SpringV2_AOP)
Deployment Descriptors
Add Application Server specific descriptor...
Web Resource Directories
Source Roots
Fix
OK
Cancel
Apply

Type	Path
Web Module Deployment Descriptor	D:\DevSoftWare\workspace\IdeaProjects\java_ar

Web Resource Directory	Path Relative to Deployment Root
D:\DevSoftWare\workspace\IdeaProjects\j...	/

D:\DevSoftWare\workspace\IdeaProjects\java_architect_lesson_2020\chap02_Architect_Aesthetic\c
D:\DevSoftWare\workspace\IdeaProjects\java_architect_lesson_2020\chap02_Architect_Aesthetic\c

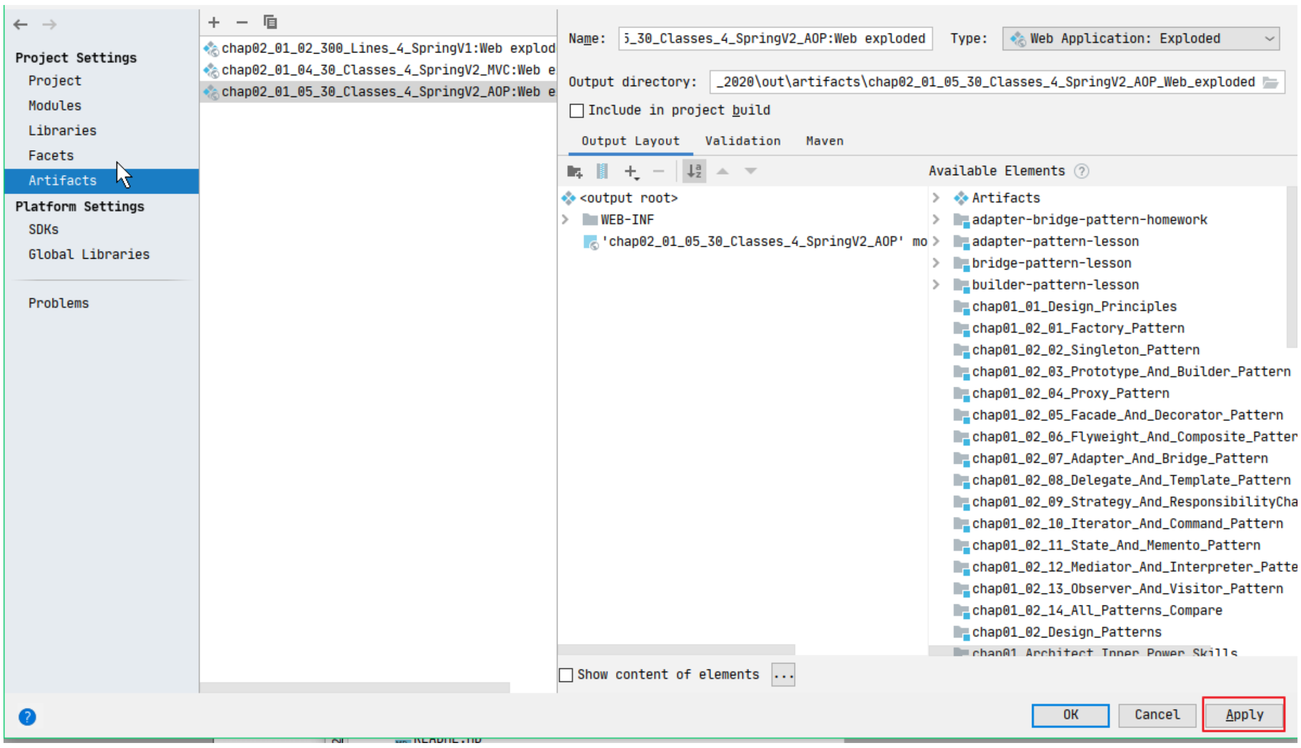
'Web' Facet resources are not included in an artifact



D:\DevSoftWare\workspace\IdeaProjects\java_architect_lesson_2020\chap02_Architect_Aesthetic\c
D:\DevSoftWare\workspace\IdeaProjects\java_architect_lesson_2020\chap02_Architect_Aesthetic\c

'Web' Facet resources are not included in an artifact

点击 Apply, 然后 OK 即可



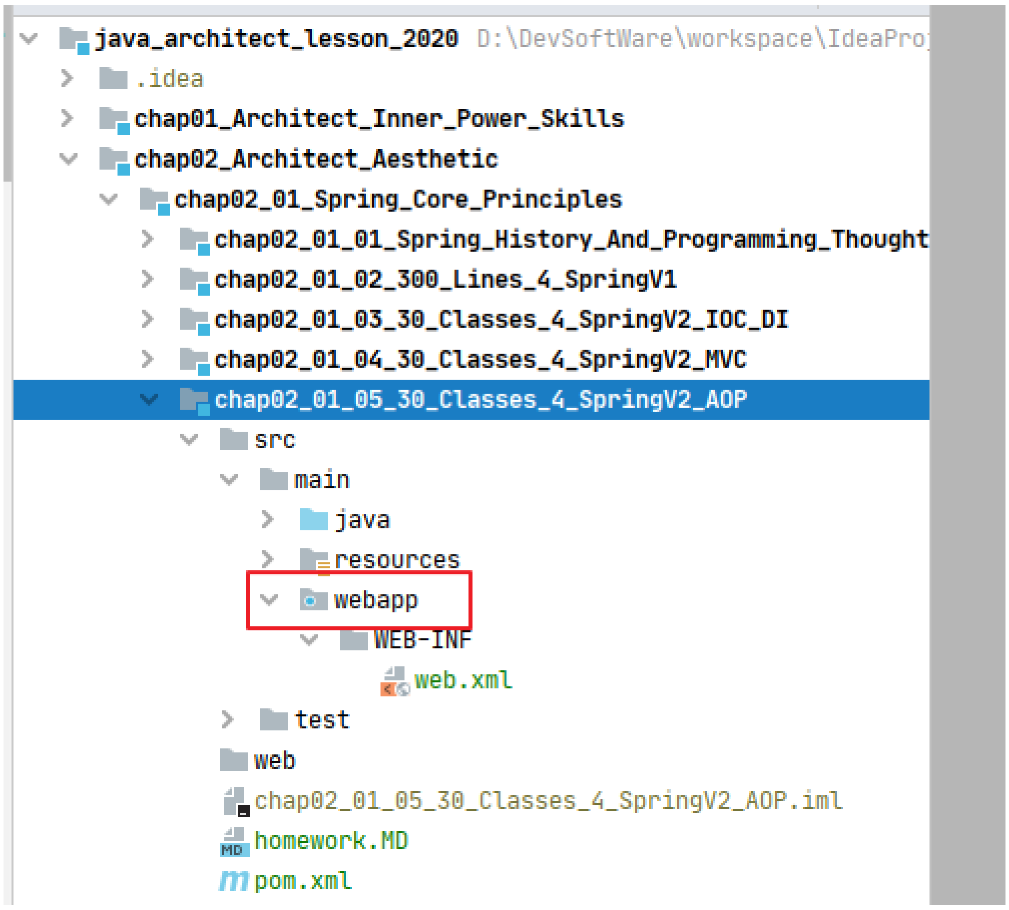
chap02_01_02_300_Lines_4_SpringV1:Web explod
chap02_01_04_30_Classes_4_SpringV2_MVC:Web e
chap02_01_05_30_Classes_4_SpringV2_AOP:Web e

Name: 5_30_Classes_4_SpringV2_AOP:Web exploded
Type: Web Application: Exploded
Output directory: _2020\out\artifacts\chap02_01_05_30_Classes_4_SpringV2_AOP_Web_exploded
Include in project build

<output root>
WEB-INF
'chap02_01_05_30_Classes_4_SpringV2_AOP' mo

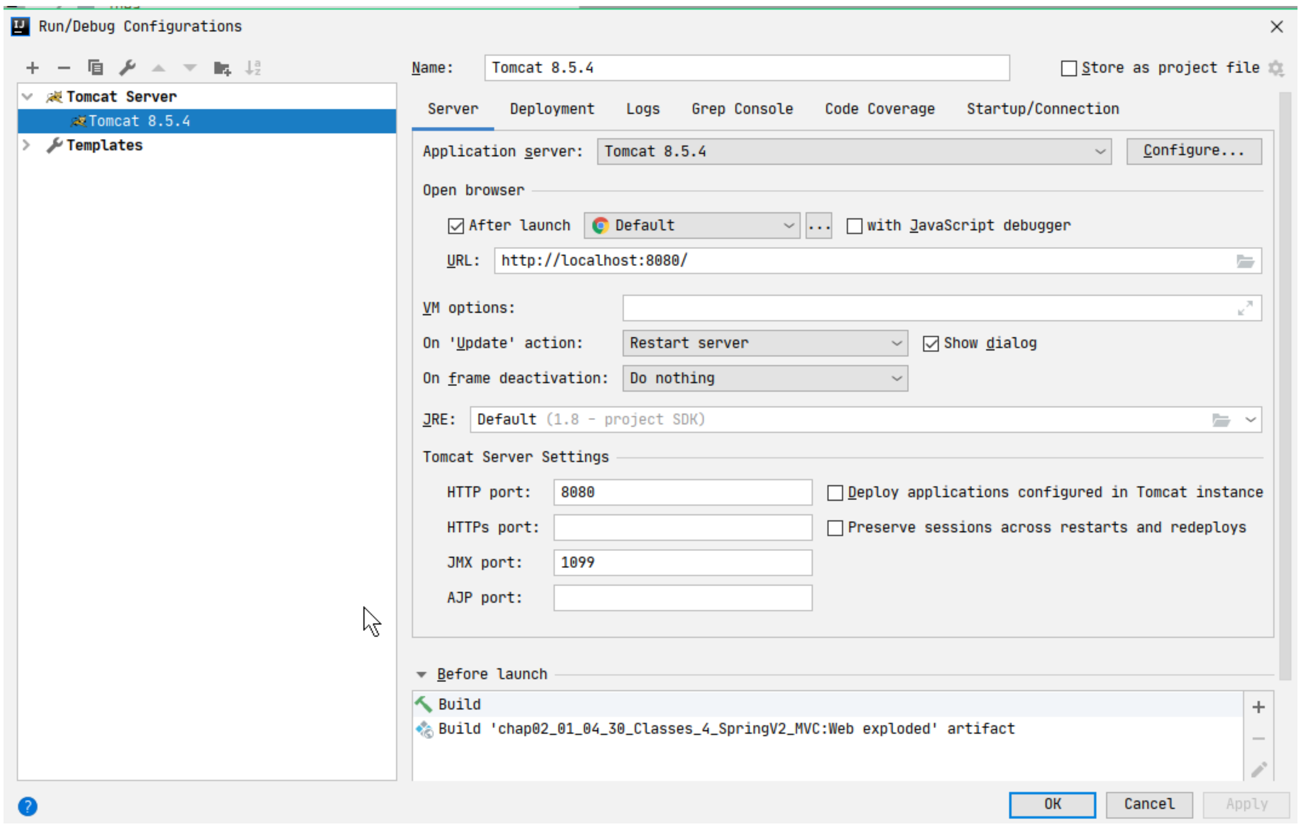
Artifacts
adapter-bridge-pattern-homework
adapter-pattern-lesson
bridge-pattern-lesson
builder-pattern-lesson
chap01_01_Design_Principles
chap01_02_01_Factory_Pattern
chap01_02_02_Singleton_Pattern
chap01_02_03_Prototype_And_Builder_Pattern
chap01_02_04_Proxy_Pattern
chap01_02_05_Facade_And_Decorator_Pattern
chap01_02_06_Flyweight_And_Composite_Patter
chap01_02_07_Adapter_And_Bridge_Pattern
chap01_02_08_Delegate_And_Template_Pattern
chap01_02_09_Strategy_And_ResponsibilityCha
chap01_02_10_Iterator_And_Command_Pattern
chap01_02_11_State_And_Memento_Pattern
chap01_02_12_Mediator_And_Interpreter_Patte
chap01_02_13_Observer_And_Visitor_Pattern
chap01_02_14_All_Patterns_Compare
chap01_02_Design_Patterns
chap01_Architect_Inner_Power_Skills

经过上述操作后, 可以看到 webapp文件夹的图标发生了变化



```
java_architect_lesson_2020  D:\DevSoftWare\workspace\IdeaProj
  .idea
  chap01_Architect_Inner_Power_Skills
  chap02_Architect_Aesthetic
    chap02_01_Spring_Core_Principles
      chap02_01_01_Spring_History_And_Programming_Thought
      chap02_01_02_300_Lines_4_SpringV1
      chap02_01_03_30_Classes_4_SpringV2_IOC_DI
      chap02_01_04_30_Classes_4_SpringV2_MVC
      chap02_01_05_30_Classes_4_SpringV2_AOP
        src
          main
            java
            resources
            webapp
              WEB-INF
                web.xml
          test
        web
        chap02_01_05_30_Classes_4_SpringV2_AOP.iml
        homework.MD
        pom.xml
```

Tomcat -> Edit Configurations...



Run/Debug Configurations

- Tomcat Server
  - Tomcat 8.5.4
- Templates

Name: Tomcat 8.5.4   ☐ Store as project file

Server | Deployment | Logs | Grep Console | Code Coverage | Startup/Connection

Application server: Tomcat 8.5.4   Configure...

Open browser
- ☑ After launch Default ... ☐ with JavaScript debugger
- URL: http://localhost:8080/

VM options:

On 'Update' action: Restart server ☑ Show dialog

On frame deactivation: Do nothing

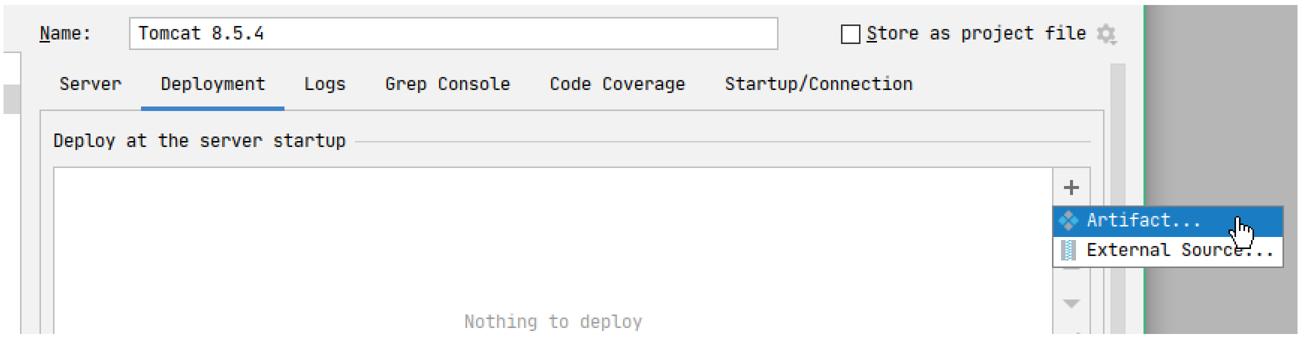
JRE: Default (1.8 - project SDK)

Tomcat Server Settings
- HTTP port: 8080 ☐ Deploy applications configured in Tomcat instance
- HTTPs port: ☐ Preserve sessions across restarts and redeploys
- JMX port: 1099
- AJP port:

Before launch
- Build
- Build 'chap02_01_04_30_Classes_4_SpringV2_MVC:Web exploded' artifact

OK Cancel Apply

Deployment -> 使用 Remove 移除之前部署的应用 -> Add -> Artifact...



Name: Tomcat 8.5.4

Store as project file

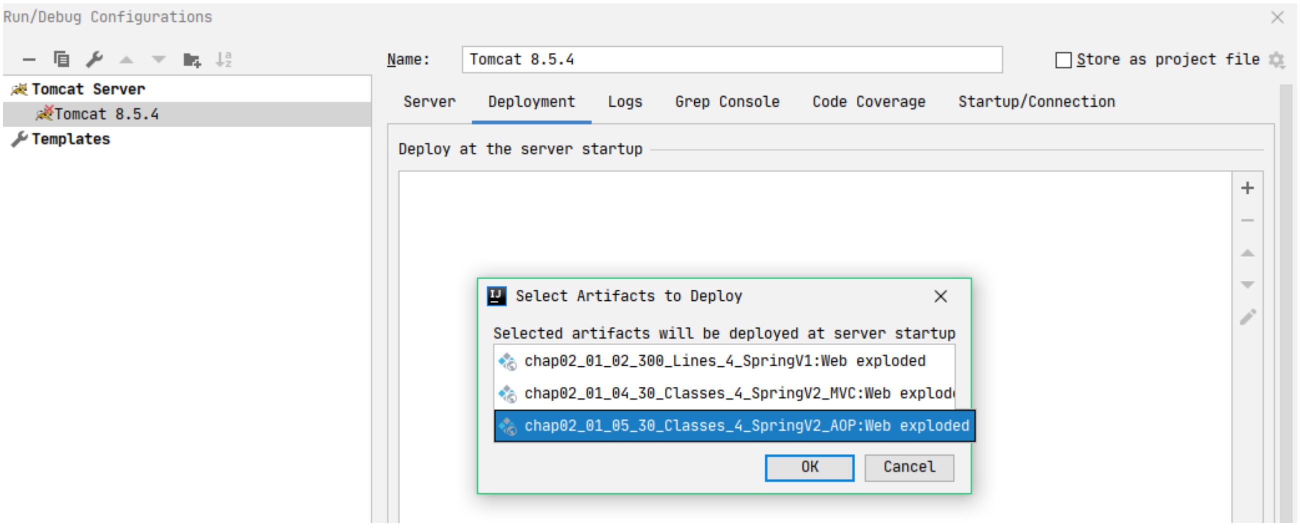
Server | Deployment | Logs | Grep Console | Code Coverage | Startup/Connection

Deploy at the server startup

Artifact...
External Source...

Nothing to deploy

选择要部署的应用, 点击OK



Run/Debug Configurations

Tomcat Server
Tomcat 8.5.4
Templates

Name: Tomcat 8.5.4

Store as project file

Server | Deployment | Logs | Grep Console | Code Coverage | Startup/Connection

Deploy at the server startup

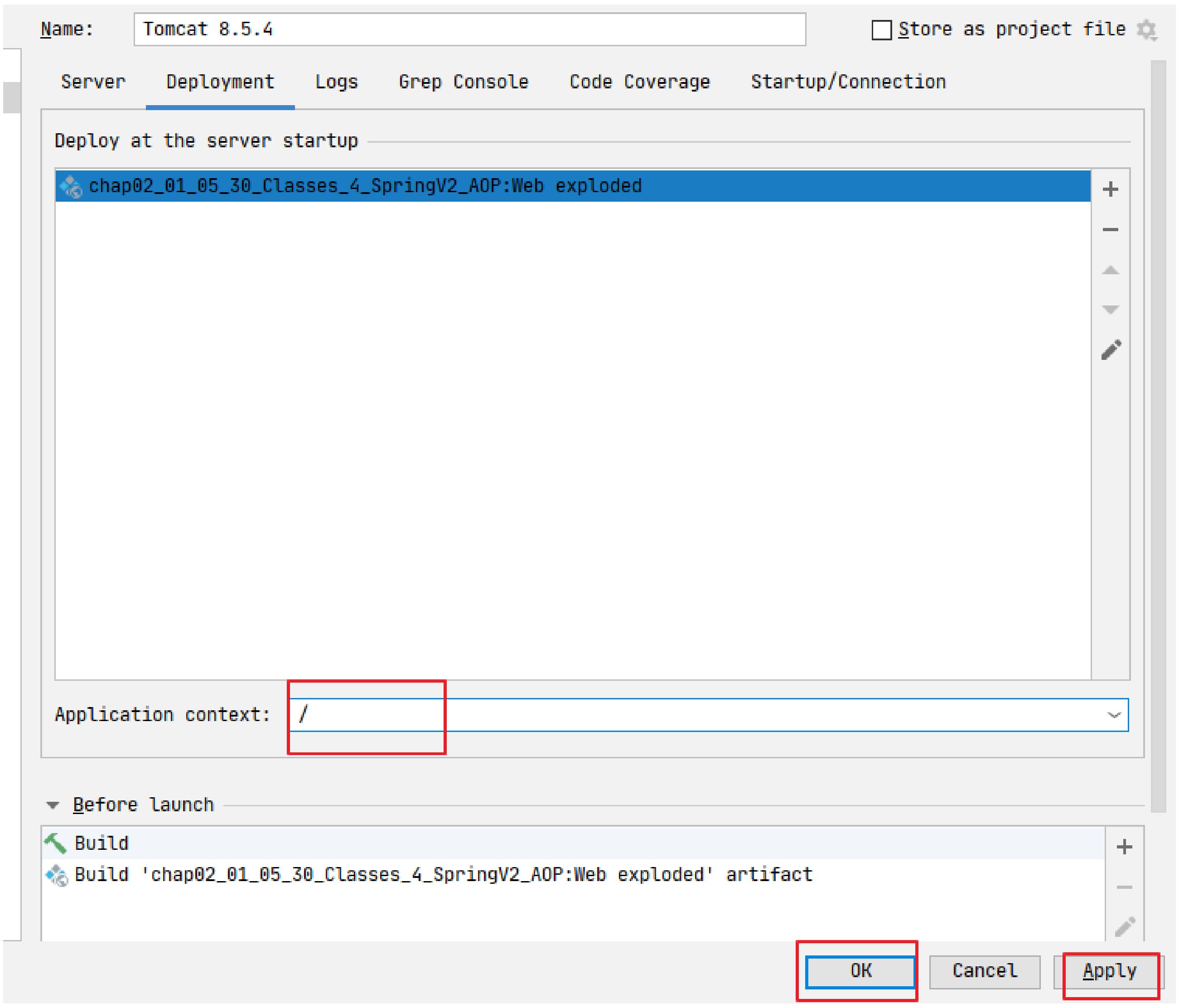
Select Artifacts to Deploy

Selected artifacts will be deployed at server startup

chap02_01_02_300_Lines_4_SpringV1:Web exploded
chap02_01_04_30_Classes_4_SpringV2_MVC:Web explod
chap02_01_05_30_Classes_4_SpringV2_AOP:Web exploded

OK Cancel

然后 修改 Application Context 为 / , 先点击Apply, 然后OK


Name: Tomcat 8.5.4

Store as project file

Server | Deployment | Logs | Grep Console | Code Coverage | Startup/Connection

Deploy at the server startup

chap02_01_05_30_Classes_4_SpringV2_AOP:Web exploded

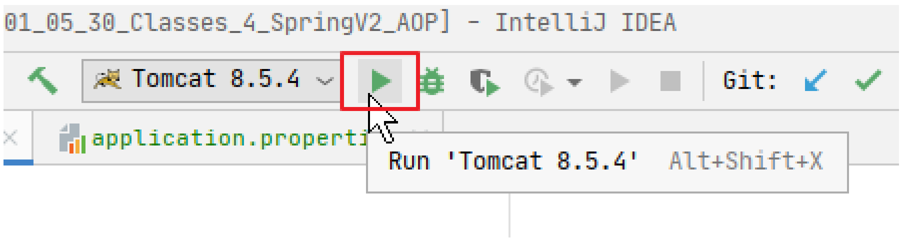
Application context: /

Before launch

Build

Build 'chap02_01_05_30_Classes_4_SpringV2_AOP:Web exploded' artifact

点击 "Run Tomcat xxx" 即可 启动应用


01_05_30_Classes_4_SpringV2_AOP] - IntelliJ IDEA