课程介绍：

在运行了自己的第一个web 程序后，我们对服务器及其运行的代码有了更进一步的认知，但是对于具体的运行细节还是一知半解。那
么服务器到底怎么运行的呢？

Servlet 介绍：

问题：
服务器在接收到浏览器的请求后，会自动调用对应的逻辑代码进行请求处理。但是逻辑代码是由程序员编写并放到服务器中，那么服务器怎么知道该怎么调用并调用哪个类和哪个方法来进行请求处理。
解决：
程序员在编写代码的时候如果能够按照服务器能够识别的规则进行编写， 浏览器按照指定的规则进行发送请求，那么服务器就可以调用并执行响应的逻辑代码进行请求处理了。

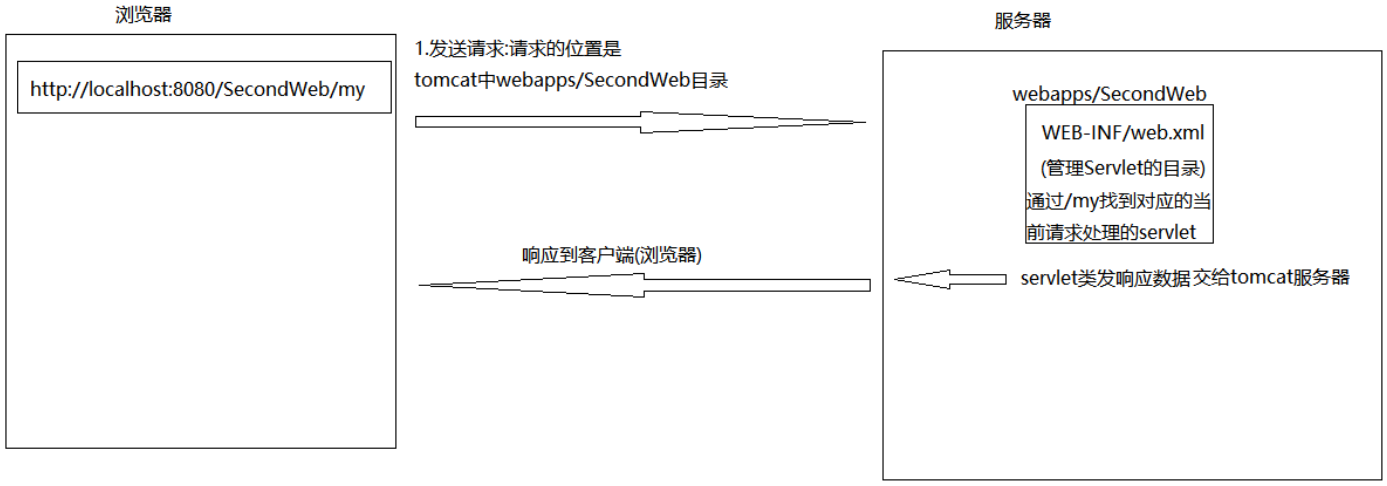
实现：

Servlet 技术

概念：
狭义的 Servlet 是指 Java 语言实现的一个接口，广义的 Servlet 是指任何实现了这个 Servlet 接口的类，一般情况下，人们将 Servlet 理解为后者。Servlet 运行于支持 Java 的应用服务器中。从原理上讲，Servlet 可以响应任何类型的请求，但绝大多数情况下 Servlet 只用来扩展基于 HTTP 协议的 Web 服务器
特点：
运行在支持 java 的应用服务器上(tomcat/ibm/微软)
Servlet 的实现遵循了服务器能够识别的规则，也就是服务器会自动的根据请求调用对应的 servlet 进行请求处理。

简单方便，可移植性强
使用：
1、 创建普通的 java 类并继承 HttpServlet 2、 覆写 service 方法
3、 在 service 方法中书写逻辑代码即可
4、 在 webRoot 下的 WEB-INF 文件夹下的 web.xml
文件中配置 servlet
运行流程:
url：http://localhost:8080/FirstWeb/abc
组成:
服务器地址： 服务器ip:端口号/虚拟项目名 /servlet 的别名
URI：虚拟项目名 /servlet 的别名
浏览器发送请求到服务器，服务器根据请求 URL 地址中的 URI 信息在 webapps 目录下找到对应的项目文件夹，然后在 web.xml 中检索对应的 servlet，找到后调用并执行 Servlet。

serlvet运行图解：




```xml
<servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>com.my.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/my</url-pattern>
</servlet-mapping>
```

```java
package com.my;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

public class MyServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.getWriter().println("this is my web!");
        String threadName = Thread.currentThread().getName();
        System.out.println("当前线程的名字:"+threadName);
        /*
        当前线程的名字:http-nio-8080-exec-5
        当前线程的名字:http-nio-8080-exec-6
        当前线程的名字:http-nio-8080-exec-7
        当前线程的名字:http-nio-8080-exec-8
        从输出的结果看出:tomcat服务器是多线程的,每个请求上来,tomcat会创建一个新的线程来处理
客户端的请求
         */
    }
}
```

Servlet 的 web.xml 配置:

服务器找到指定的项目目录后，去其下面的WEB-INF目录下加载web.xml
Servlet的执行过程:

Idea集成 Tomcat

Servlet的生命周期

生命周期概述：从创建到销毁的一段时间
生命周期方法 ： 从创建到销毁，所调用的那些方法。

init方法
在创建该servlet的实例时，就执行该方法。
一个servlet只会初始化一次， init方法只会执行一次
默认情况下是 ： 初次访问该servlet，才会创建实例。
service方法
只要客户端来了一个请求，那么就执行这个方法了。
该方法可以被执行很多次。 一次请求(是一个新线程)，对应一次service方法的调用

destroy方法
servlet销毁的时候(停止tomcat时/项目从tomcat中删除时)，就会执行该方法

```java
package com.my;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

public class MyServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        resp.getWriter().println("this is my web!");
        System.out.println("service被调用了");
        /*
        每次请求servlet都会被tomcat调用一次,且每次都是被tomcat创建新的线程调用
         */
    }

    @Override
    public void init() throws ServletException {
        System.out.println("init初始化了");
        /*
        在第一次请求servlet的时候被调用,且只被调用一次
    开发中,用来初始化在servlet中需要使用的一些数据,
        如果当这个要初始化的数据量很大的时候,会把这个init()尽量提前被调用:
        <load-on-startup>2</load-on-startup>
         */
    }
```

```
    @Override
    public void destroy() {
        System.out.println("被销毁了:destroy()");
        /*

        1在tomcat停止的情况下被调用
        2把项目同tomcat中移除的时候被调用
    开发中用来关停项目销毁一些占用了服务器的资源
         */
    }
}
```

让Servlet创建实例的时机 提前。

1.默认情况下，只有在初次访问servlet的时候，才会执行init方法。
有的时候，我们可能需要在这个方法里面执行一些初始化工作，甚至是做一些比较耗时的逻辑。 那么这个时候，初次访问，可能会在init方法中逗留太久的时间。 那么有没有方法可以让这个初始化的时机提前一点
在服务器其中过程中就让init被调用
2.在配置的时候， 使用load-on-startup元素来指定， 给定的数字越小，启动的时机就越早。 一般不写负数， 从2开始即可。

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">
    <servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>com.my.MyServlet</servlet-class>
        <load-on-startup>2</load-on-startup>  <!--会让init方法提前到服务器启动过程之中执
行-->
    </servlet>
    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/my</url-pattern>
    </servlet-mapping>
</web-app>
```

总结Servlet的生命周期为：
1.从Servlet第一次被调用(客户端发送第一个请求时)，到服务器关闭
2.如果在web.xml中配置了load-on-startup
则是从服务器开启到服务器关闭。

doGet 和 doPost

不算生命周期方法，所谓的生命周期方法是指，从对象的创建到销毁一定会执行的方法， 但是这两个方法，不一定会执行。

```
package com.my;

import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServlet;
```

```java
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

public class Demo2 extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("doGet被调用了");//处理get请求
  }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("doPost被调用了");//处理post请求
  }

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
//        super.service(req, resp);//开发中无论是get还是post都只要写service方法,并删除这句
话super.service()
        System.out.println("service被调用了");
    }
}
```

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>登录页面</title>
</head>
<body>
    <form action="Demo2" method="post">
        用户名:<input type="text" name="uname"/><br>
        密码:<input type="password" name="pwd"/><br>
        <input type="submit" value="登录"/>
    </form>
</body>
</html>
```

Service 和 doGet 和 doPost 方法的区别

service 方法:
不管是 get 方式还是 post 方式的请求，如果 Servlet 类中有
service 方法，则优先调用 Service 方法。
doGet 方法:
在没有 service 方法的情况下如果是 get 方式的请求所调

用的处理请求的方法
doPost 方法:
在没有 service 方法的情况下如果是 post 方式的请求所调
用的处理请求的方法
如果没有service方法只要doGet和doPost方法时处理两个提交方式

Servlet 的常见错误总结：

```
* Servlet的常见错误：
* 404错误:资源未找到
* 原因一：在请求地址中的servlet的别名书写错误。
* 原因二：虚拟项目名称拼写错误
* 500错误：内部服务器错误
* 错误一：java代码有错误
* java.lang.ClassNotFoundException: com.bjsxt.servlet.ServletMothod
* 解决：
* 在web.xml中校验servlet类的全限定路径是否拼写错误。
* 错误二：
* 因为service方法体的代码执行错误导致
* 解决：
* 根据错误提示对service方法体中的代码进行错误更改。
* 405错误:请求方式不支持
* 原因：
* 请求方式和servlet中的方法不匹配所造成的。
* 解决：
* 尽量使用 service 方法进行请求处理，并且不要在 service 方法中调用父类的
service（在servcie中去掉super(req，resp)）。
```

html文件请求路径的问题:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>登录</title>
</head>
<!--
关于action的路径的问题:
/loginUser 相对于项目的tomcat的根目录开始
http://localhost:8080/MyWebTest_war_exploded/loginUser    MyWebTest_war_exploded这个目录与工程名
不一样是idea自己加的后缀
直接写: loginUser,就是相对于当前工程的MyWebTest根目录开始
如果一定要写带/的请求路径,就这么写/MyWebTest_war_exploded/loginUser
-->

<body>
    <form action="/MyWebTest_war_exploded/loginUser" method="post">
        用户名<input type="text" name="uname">
        密码:<input type="password" name="pwd">
        爱好:
        抽烟<input type="checkbox" name="favor" value="1">
        喝酒<input type="checkbox" name="favor" value="2">
        spa<input type="checkbox" name="favor" value="3">
```

```
        躺平<input type="checkbox" name="favor" value="4">
        <input type="submit" value="提交">
    </form>
</body>
</html>
```