03 독립 실행형 서블릿 애플리케이션

Containerless 개발 준비

컨테이너 설치와 배포 등의 작업을 하지 않고 서블릿 컨테이너를 동작시키는 방법을 코드로 구현해본다.

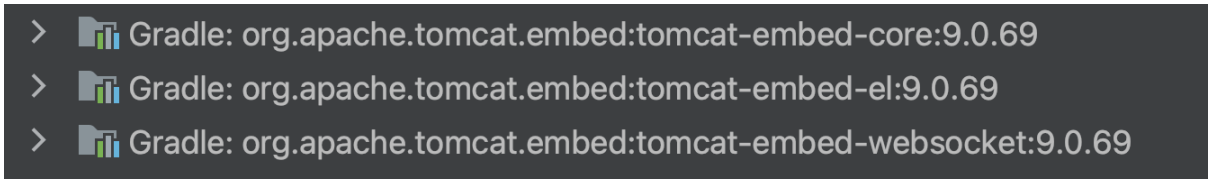
스프링 부트가 사용하는 것으로 보이는 다음 두 라인을 제거하고 빈 main() 메소드만 남긴다.

```
@SpringBootApplication

SpringApplication.run(HellobootApplication.class, args);
```

서블릿 컨테이너 띄우기

스프링 부트 프로젝트를 만들 때 web 모듈을 선택하면 다음과 같은 내장형 톰캣 라이브러리가 추가된다.



내장형 톰캣의 초기화 작업과 간편한 설정을 지원하도록 스프링 부트가 제공하는 TomcatServletWebServerFactory를 사용하면 톰캣 웹 서버(서블릿 컨테이너)를 실행하는 코드를 만들 수 있다.

```
ServletWebServerFactory serverFactory = new TomcatServletWebServerFactory();
WebServer webServer = serverFactory.getWebServer();
webServer.start();
```

서블릿 등록

코드에서 서블릿을 등록하려면 ServletContext가 필요하다. ServletContext를 전달해서 서블릿 등록과 같은 초기화 작업을 할 때는 ServletContextInitializer를 구현한 오브젝트를 ServletWebServerFactory의 getWebServer() 메소드에 전달한다.

ServletContextInitializer는 @FunctionalInterface이므로 람다식으로 전환해서 사용하면 편리하다.

```
@FunctionalInterface
public interface ServletContextInitializer {
  void onStartup(ServletContext servletContext) throws ServletException;
}
```

서블릿은 HttpServlet 클래스를 상속해서 필요한 메소드를 오버라이딩 하는 방식으로 만들 수 있다.

https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html

서블릿을 등록할 때는 서블릿 이름과 서블릿 오브젝트를 이용한다. 서블릿 등록 정보에는 매핑할 URL 정보를 지정해야 한다.

```
servletContext.addServlet("hello", new HttpServlet() {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
    }
}).addMapping("/hello");
```

서블릿에서는 HttpServletRequest를 이용해서 요청 정보를 가져오고, HttpServletResponse를 이용해서 응답을 만드는 작업을 수행한다.

3가지 요소(상태 코드, 헤더, 바디)를 이용해서 웹 요청을 생성한다.

```
resp.setStatus(HttpStatus.OK.value());
resp.setHeader(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN_VALUE);
resp.getWriter().println("Hello Servlet");
```

상태코드가 OK인 경우엔 생략이 가능하다.

“Content-Type” 헤더를 지정해서 바디의 형식을 지정해야 한다.

ContentType을 헤더를 지정할 때는 setContentType() 메소드를 이용할 수 있다.

```
resp.setContentType(MediaType.TEXT_PLAIN_VALUE);
```

서블릿 요청 처리

웹 클라이언트로부터 전달 받은 요청 정보를 서블릿 기능을 작성할 때 활용할 수 있다.

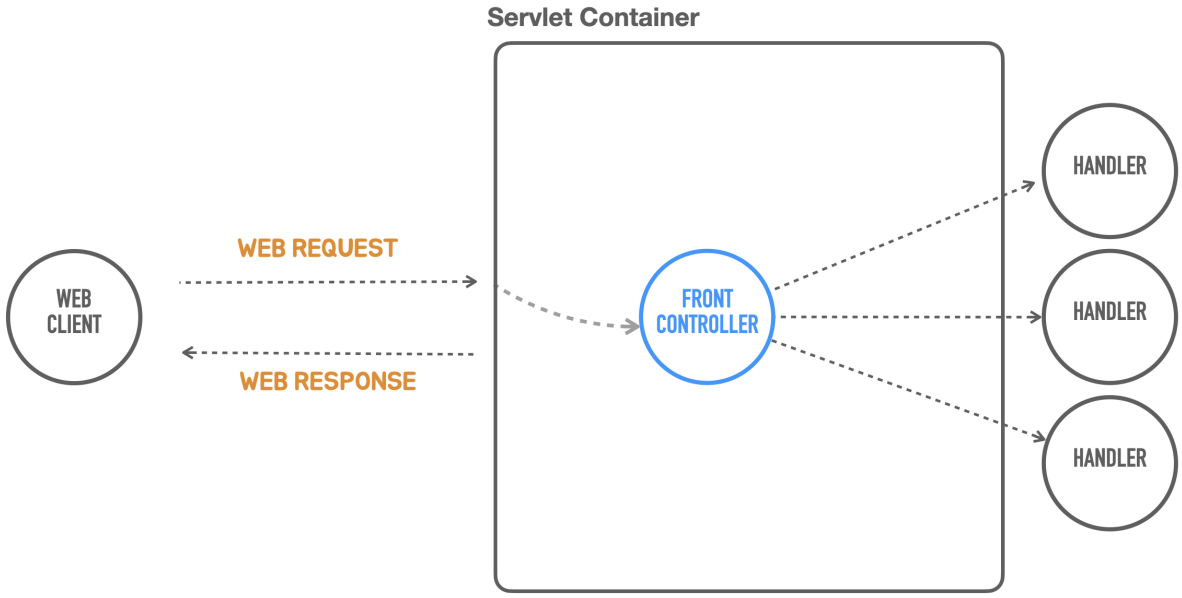
대표적으로 URL 등으로 전달된 파라미터 값을 추출해서 사용하는 방법이 있다.

HttpServletRequest에서 name 파라미터의 값을 가져올 때는 getParameter() 메소드를 이용한다.

```
String name = req.getParameter("name");
```

요청으로부터 가져온 정보를 활용해서 웹 애플리케이션 로직을 수행하는 코드를 작성한다.

프론트 컨트롤러



여러 요청을 처리하는데 반복적으로 등장하게 되는 공통 작업을 하나의 오브젝트에서 일괄적으로 처리하게 만드는 방식을 프론트 컨트롤러 패턴이라고 한다.

https://martinfowler.com/eaaCatalog/frontController.html

서블릿을 프론트 컨트롤러로 만들려면 모든 요청, 혹은 일정 패턴을 가진 요청을 하나의 서블릿이 담당하도록 매핑해준다.

프론트 컨트롤러로 전환

프론트 컨트롤러가 모든 URL을 다 처리할 수 있도록 서블릿 바인딩을 변경한다.

```
}).addMapping("/*");
```

서블릿 내에서 HTTP 요청 정보를 이용해서 각 요청을 분리한다. 만약 처리할 수 있는 HTTP 요청 정보가 없다면 상태 코드를 404로 설정한다.

```
if (req.getRequestURI().equals("/hello") && req.getMethod().equals(HttpMethod.GET.name())) {
    ...
}
else if (req.getRequestURI().equals("/user")) {
    ...
}
else {
    resp.setStatus(HttpStatus.NOT_FOUND.value());
}
```

Hello 컨트롤러 매핑과 바인딩

프론트 컨트롤러가 요청을 분석해서 처리할 요청을 구분한 뒤에 이를 처리할 핸들러(컨트롤러 메소드)로 요청을 전달한다. 핸들러가 처리하고 돌려준 리턴 값을 해석해서 웹 요청을 생성한다.

프론트 컨트롤러가 HTTP 요청을 처리할 핸들러를 결정하고 연동하는 작업을 매핑이라고 한다.

또, 핸들러에게 웹 요청 정보를 추출하고 의미있는 오브젝트에 담아서 전달하는 작업을 바인딩이라고 한다.

프론트 컨트롤러의 두 가지 중요한 기능은 매핑과 바인딩이다.

```
if (req.getRequestURI().equals("/hello") && req.getMethod().equals(HttpMethod.GET.name())) {
  String name = req.getParameter("name");

  String ret = helloController.hello(name);

  resp.setStatus(HttpStatus.OK.value());
  resp.setHeader(HttpHeaders.CONTENT_TYPE, MediaType.TEXT_PLAIN_VALUE);
  resp.getWriter().println(ret);
}
```

매핑과 바인딩은 세밀한 규칙을 부여하면 매번 코드를 작성하지 않고도 공통 코드를 이용해서 이를 처리할 수 있도록 만들 수 있다.