**고급 웹 프로그래밍 Home Work #2**

201613998 소프트웨어공학과 설경원

**INDEX.**

1. **Spring Framework 소개**
   1. 모든 소스코드 실행한 결과 **p.2**
   2. 각 소스코드를 변경해 실행한 결과 **p.2**
   3. 기존 소스코드 대비해 변경하여 실험한 항목을 표로 정리 **p.3**
2. **IOC**

2-1. 모든 소스코드 실행한 결과 **p.4**

2-2. 각 소스코드를 변경해 실행한 결과 **p.4**

2-3. 기존 소스코드 대비해 변경하여 실험한 항목을 표로 정리 **p.5~6**

1. **AOP**

3-1. 모든 소스코드 실행한 결과 **p.7**

3-2. 각 소스코드를 변경해 실행한 결과 **p.7**

3-3. 기존 소스코드 대비해 변경하여 실험한 항목을 표로 정리 **p.8~9**

1. **Spring MVC**

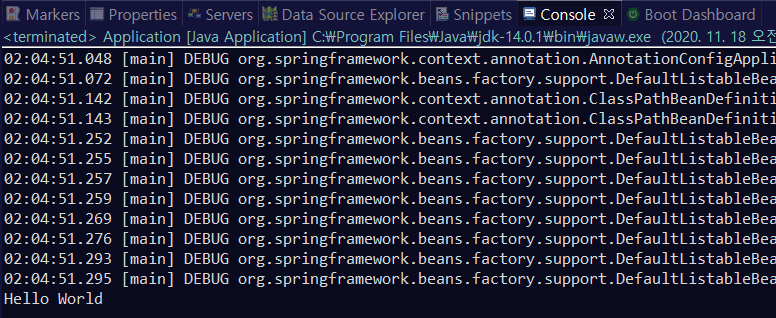
4-1. 모든 소스코드 실행한 결과 **p.10**

4-2. 각 소스코드를 변경해 실행한 결과 **p.10**

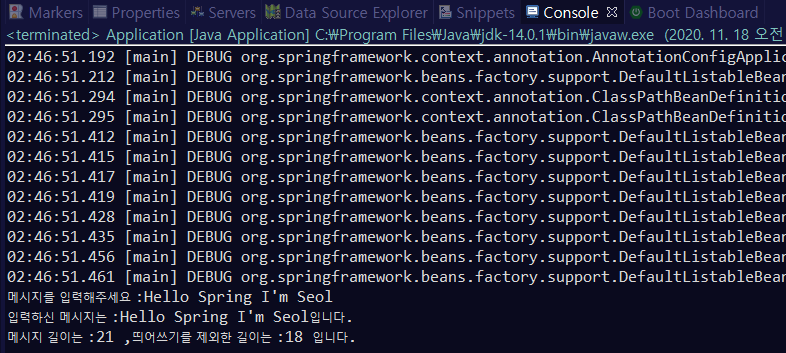
4-3. 기존 소스코드 대비해 변경하여 실험한 항목을 표로 정리 **p.11~13**

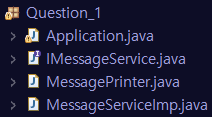
**1. Spring Framework 소개**

**-1. 모든 소스코드 실행한 결과**

****

**-2. 각 소스코드를 변경해 실행한 결과**

****

****

**[예제 1의 Explorer]**

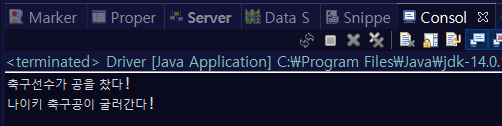
**-3. 기존 소스코드 대비해 변경하여 실험한 항목을 표로 정리**

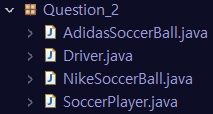
|  |  |
| --- | --- |
| 수정 전 코드 | 수정 후 코드 |
| **IMessageService Interface** |  |
| **수정 내용**  인터페이스 IMessageService에서 Message의 길이를 출력하기 위해 해당 코드를 추가하였습니다. | |
| **MessageServiceImp Class** |  |
| **수정 내용**  MessageServiceImp클래스에서 기존의 Hello World 출력하던 것을 사용자의 입력을 받기 위해 Scanner를 사용하고 입력 받은 문자열과 문자열의 길이, 띄어쓰기를 제외한 문자열의 길이를 출력하도록 수정하였습니다.  Spring framework를 소개하는 첫번째 단계에서는 코딩을 수정을 하는 과정보다 spring framework를 프로젝트에 적용하고 구축환경을 설정하는데 많은 시간을 썼던 것 같습니다. | |
| **MessagePrinter Class** |  |
| **수정 내용**  기존 출력 메시지에서 입력 받은 메시지 출력과 해당 메시지의 길이, 띄어쓰기를 제외한 길이를 추가하였습니다. | |

**2. IOC :**

기존 강의자료에 주어진 예제 코드가 아닌 IOC를 설명할 수 있는 좋은 예시의 코드의 전/후를 비교하며 설명할 수 있게 변경하였습니다.

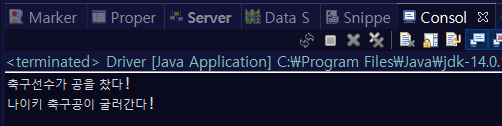
**-1. 모든 소스 코드를 실행한 결과**

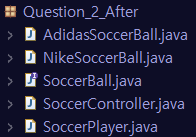


****

**[예제 2의 수정 전 Explorer]**

**-2. 각 소스코드를 변경해 실행한 결과**



****

**[예제 2의 수정 후 Explorer]**

IOC를 적용한 구조 변경을 통해 출력은 같지만 구조의 변경을 통해 관리 효율을 높이는 이점을 아래의 표를 통해 보이겠습니다.

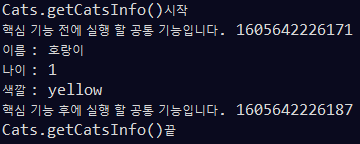
**-3. 기존 소스코드 대비해 변경하여 실험한 항목을 표로 정리**

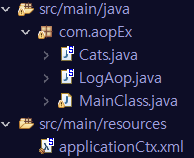
|  |  |
| --- | --- |
| 수정 전 코드 | 수정 후 코드 |
| **SoccerPlayer Class** |  |
| **수정 내용**  보통 프로그램의 실행흐름에 따른다면, 무언가 필요한 쪽에서 필요한 개개체를 만들고, 만들어진 객체의 메소드를 직접 호출해서 사용한다. 여기서 각 객체는 프로그램의 흐름에 능동적으로 참여하게 되는데 이때 모든 작업은 사용하는 쪽에서 제어한다. 그러한 의존 관계를 관리하기 위해 IOC를 사용한다. | |
| **Driver Class** | **SoccerConroller Class** |
| **수정 이유**  위의 코드에서 수정 전에는 SoccerPlayer가 playSoccer을 위해서는 다음 수정 항목인 NikeSoccerBall을 만들어서 사용해야 하는데 그것은 의존하고 있다고 표현 가능하다. 만약 다른 축구공을 사용해야할 경우, SoccerPlayer의 많은 부분을 수정해야할 것이다. | |
| **AdidasSoccerBall, NikeSoccerBall Class** | **SoccerBall interface, AdidasSoccerBall, NikeSoccerBall Class** |
| **수정 내용**  Beans 클래스를 만들어 기존의 관리 방식에서 의존관계 역전을 통한 interface를 implements하는 방식으로 어떤 공이든 코드에 변경 없이 사용 가능하게 수정하였다.  위의 @Component 애너테이션이 붙은 클래스들은 Spring의 Container가 알아서 Spring Bean 객체로 등록하고 생성한다. 이렇게 생성된 객체는 @AutoWired라는 애너테이션이 붙은 변수의 타입(타입이 같은 Bean 여러 개 있다면 이름 확인)을 보고 해당 변수에 객체를 주입하게 된다. | |

**3. AOP :**

AOP를 구현하는 방법은 XML 스키마 기반으로 구현, @Aspect 애너테이션 기반 구현이 있습니다.

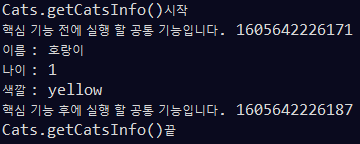
**-1. 모든 소스코드 실행한 결과**

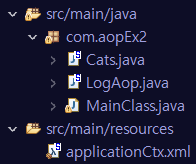
****

****

**[예제 3의 수정 전 Explorer]**

**-2. 각 소스코드를 변경해 실행한 결과**

****

****

**[예제 3의 수정 후 Explorer]**

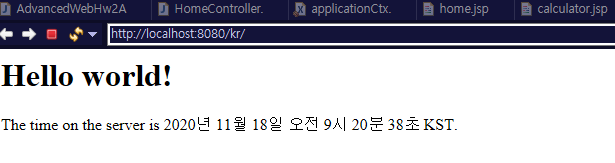
그 중 수정 전에는 XML 스키마 기반으로 구현하고 수정 후에는 애너테이션을 사용한 AOP구현을 하였습니다.

**-3. 기존 소스코드 대비해 변경하여 실험한 항목을 표로 정리**

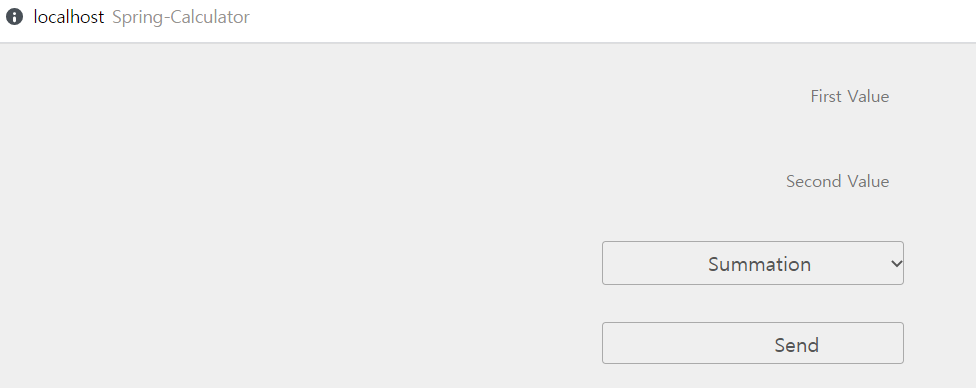
|  |  |  |
| --- | --- | --- |
| 수정 전 코드 | | 수정 후 코드 |
| **applicationCtx.xml** | | **applicationCtx.xml** |
| **수정 내용**  수정한 내용이 AOP를 구현하는 방법의 다른 종류인 한가지 이기 때문에 상세하게 작성해보겠습니다.  먼저 수정전은 pom.xml파일에 AOP 의존설정을 아래와 그림과 같이 먼저 하게 됩니다.    그 후 AOP의 사용목적인 공통 기능의 클래스(Advice 역할)를 제작하게 됩니다. 해당 코드에 대한 설명은 다음 표에서 하겠습니다. 그 후, XML 설정 파일에 Namespaces에서 aop를 선택합니다. 아래 그림 중 왼쪽은 aop를 선택한 모습, 오른쪽은 선택 후 코드 추가된 모습입니다.      후에, beans 클래스를 연동시켜주면 해당 코드가 정상 작동하게 됩니다.    수정 후는 위의 과정을 포함하고 @Aspect 애너테이션이 있는 클래스를 공통기능으로 자동 인식해주는 편리한 장치입니다. 다음 나올 수정 코드의 명시(@Aspect)를 통해 공통 기능을 인식하게 됩니다. | | |
|  |  | |
| **수정 내용**  위의 설명 기반으로 기존 공통 기능을 @Aspect 애너테이션과 @Around 애너테이션 사용을 통해 지정하여 수정하는 모습입니다. 장점은 xml 파일에서 AOP 설정을 따로 하지 않고 첫 번째 수정사항처럼 간결하게 동작하게 할 수 있습니다. | | |

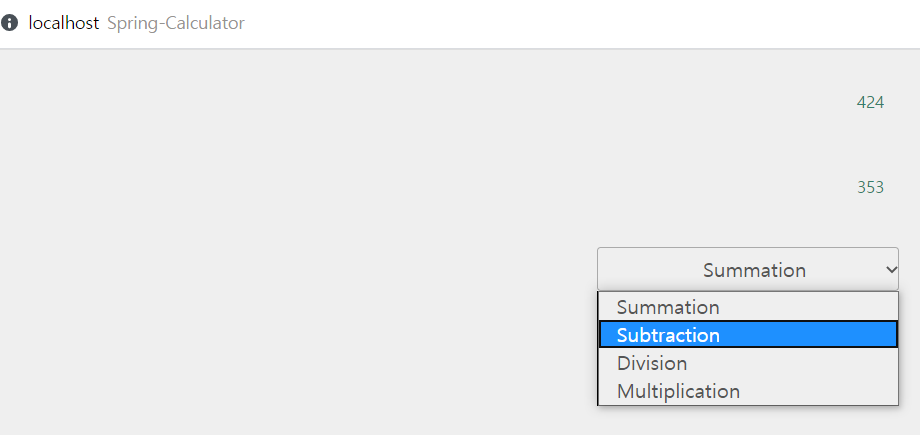
**4. Spring MVC**

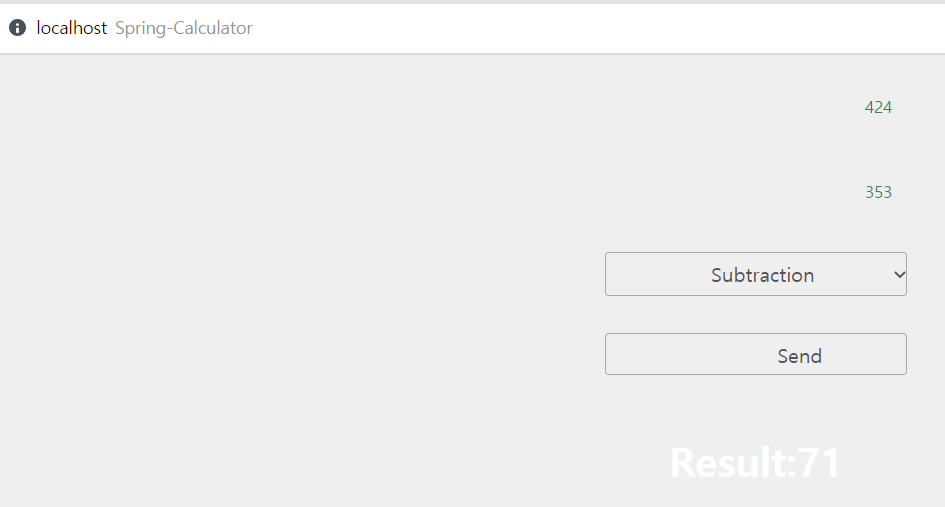
**-1. 모든 소스코드 실행한 결과**

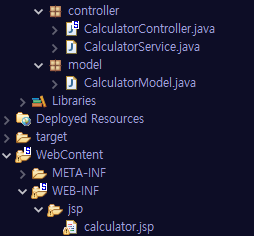


**-2. 각 소스코드를 변경해 실행한 결과**

****

****

****

****

**-3. 기존 소스코드 대비해 변경하여 실험한 항목을 표로 정리**

|  |  |
| --- | --- |
| 수정 전 코드 | 수정 후 코드 |
|  | **CalculatorService.java(Controller)** |
| **수정 내용**  기존 Hello World 출력 예제에서 간단한 계산기 프로그램을 구현하고자 controller에서 계산기 역할을 하는 사칙연산 계산 수행 코드를 먼저 작성하였습니다. | |
|  | **CalculatorController.java(Controller)** |
| **수정 내용**  View부분의 calculator.jsp와 연동할 수 있게 form으로부터 받아온 값을 controller class인 calculatorService와 연동하여 처리하는 코드를 작성하였습니다. @RequestMapping으로 불러온 값을 POST방식으로 처리하는 코드 입니다. | |
|  | **CalculatorModel.java(Model)** |
| **수정 내용**  값을 받아오고 Controller에 의해 처리되는 Model 부분의 클래스 부분을 작성하였습니다. | |
|  | **Calculator.jsp(View)** |
| **수정 내용**  디자인과 사용자에게 외형적으로 보이는 사이트의 일부분을 담은 코드를 추가하였습니다. 실질적인 코드는 사용자의 입력을 받는 form형식의 부분과 디자인을 담당하는 style코드가 대다수이기 때문에 생략하였습니다. | |