Spring Boot

Jae Hyeon Kim

Contents

1. Servlet and JSP
2. EJB
3. Spring Framework
4. Spring Boot

Servlet and JSP

Servlet

- 웹 기반의 요청에 대한 동적인 처리가 가능한 서버사이드에서 돌아가는 자바 프로그램
- 자바 코드 안에 HTML 코드
- 웹 개발을 위해 만든 표준

Servlet

```java
@WebServlet(name = "memberSaveServlet", urlPatterns = "/servlet/members/save")
public class ThreeParams extends HttpServlet {

    1개 사용 위치
    private final MemberRepository memberRepository = MemberRepository.getInstance();

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        final String username = request.getParameter( name: "username");
        final int age = Integer.parseInt(request.getParameter( name: "age"));
        final Member member = new Member(username, age);
        memberRepository.save(member);

        response.setContentType("text/html");
        response.setCharacterEncoding("utf-8");

        final PrintWriter printWriter = response.getWriter();

        printWriter.write( s: "<html>\n" +
                "<head>\n" +
                " <meta charset=\"UTF-8\">\n" +
                "</head>\n" +
                "<body>\n" +
                "성공\n" +
                "<ul>\n" +
                " <li>id=" + member.getId() + "</li>\n" +
                " <li>username=" + member.getUsername() + "</li>\n" +
                " <li>age=" + member.getAge() + "</li>\n" +
                "</ul>\n" +
                "<a href=\"/index.html\">메인</a>\n" +
                "</body>\n" +
                "</html>");
    }
}
```

Servlet

```
writer.println("<html>");
writer.println("<head>");
writer.println("</head>");
writer.println("<body>");
writer.println("<h1>helloWorld~</h1>");
writer.println("name : " + request.getParameter("name") + "<br/>");
writer.println("id : " + request.getParameter("id") + "<br/>");
writer.println("pw : " + request.getParameter("pw" + "<br/>"));
writer.println("major : " + request.getParameter("major") + "<br/>");
writer.println("protocol : " + request.getParameter("protocol") + "<br/>");
writer.println("</body>");
writer.println("</html>");
writer.close();
```

JSP(Java Server Page)

- Java 언어를 기반으로 하는 서버사이드 스크립트 언어
- HTML 코드 안에 자바 코드
- Servlet을 보완하고 기술을 확장한 스크립트 방식 표준

JSP(Java Server Page)

```
<%@ page import="hello.servlet.basic.domain.member.Member" %>
<%@ page import="hello.servlet.basic.domain.member.MemberRepository" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%
    final MemberRepository memberRepository = MemberRepository.getInstance();
    final String username = request.getParameter("username");
    final int age = Integer.parseInt(request.getParameter("age"));
    final Member member = new Member(username, age);
    memberRepository.save(member);
%>
<html>
<head>
    <title>Title</title>
</head>
<body>
성공
<ul>
    <li>id=<%=member.getId()%>
    </li>
    <li>username=<%=member.getUsername()%>
    </li>
    <li>age=<%=member.getAge()%>
    </li>
</ul>
<a href="/index.html">메인</a>
</body>
</html>
```

JSP(Java Server Page)

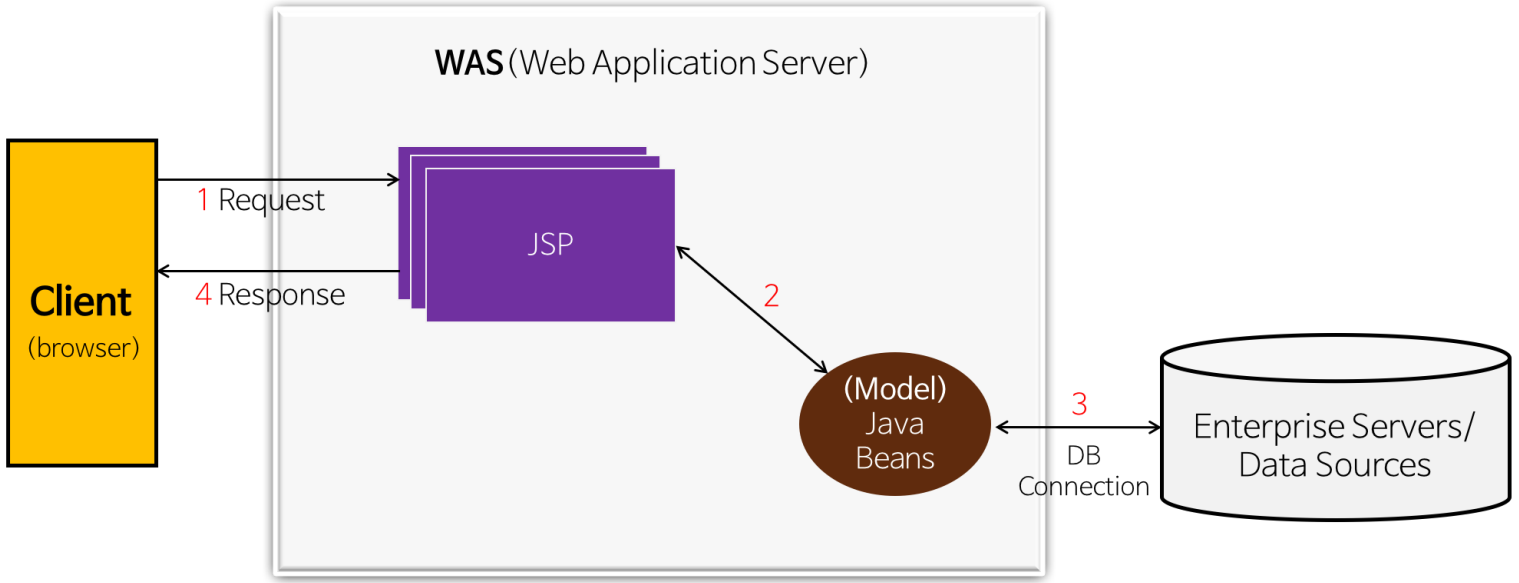
```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Reading Three Request Parameters</TITLE>
<LINK REL=STYLESHEET HREF="JSP-Styles.css" TYPE="text/css">
</HEAD>

<BODY>
<H1>Reading Three Request Parameters</H1>
<UL>
    <LI><B>param1</B>: <%= request.getParameter("param1") %>
    <LI><B>param2</B>: <%= request.getParameter("param2") %>
    <LI><B>param3</B>: <%= request.getParameter("param3") %>
</UL>
</BODY>
</HTML>
```

Servlet and JSP

- Servlet과 JSP는 모두 뷰와 비즈니스 로직을 한 곳에서 처리한다는 한계가 존재
- Servlet은 data processing에 유리
- JSP는 presentation에 유리

Servlet and JSP

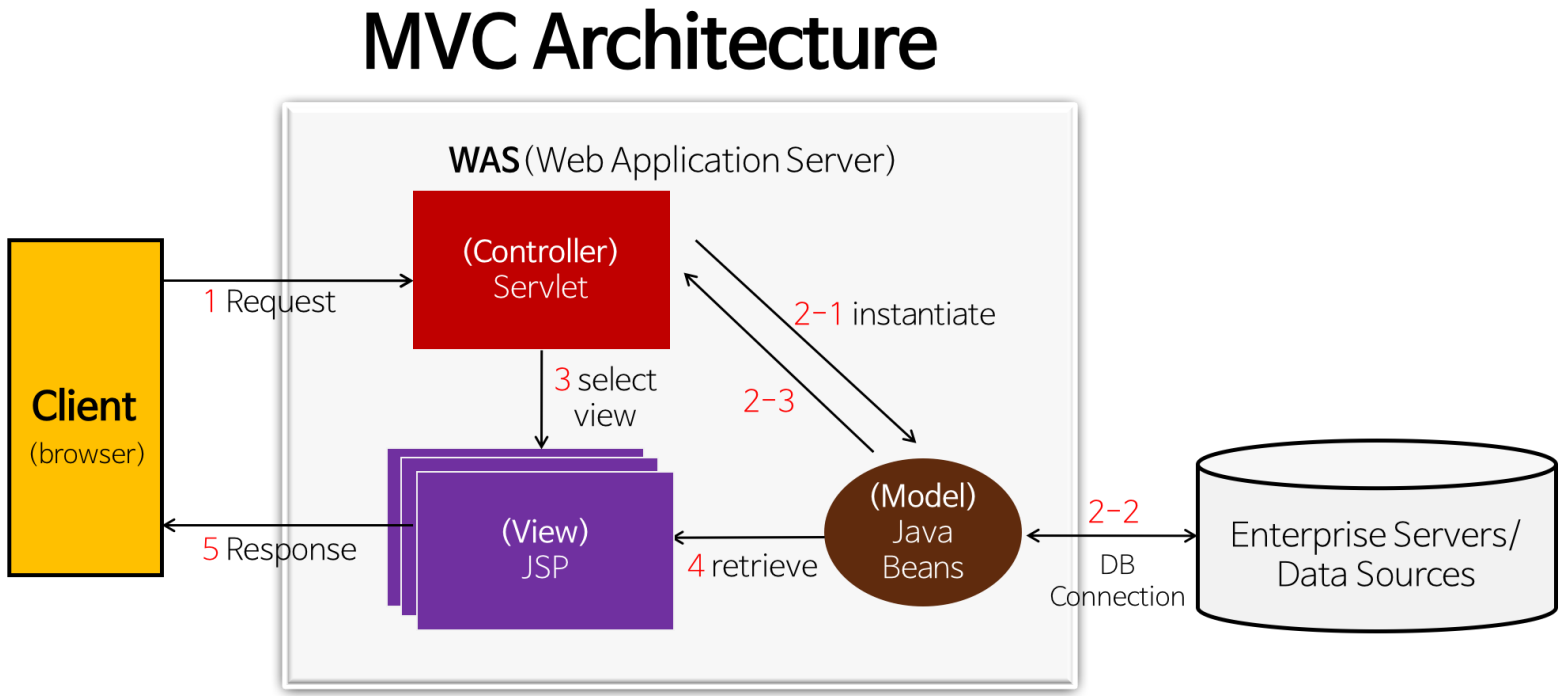
- JSP만을 이용하는 모델



MVC Architecture

- MVC Pattern은 사용자 인터페이스, 데이터 및 제어 논리를 구현하는데 일반적으로 사용되는 소프트웨어 아키텍처 패턴
- 모델은 뷰에 출력할 데이터를 담아둔다
- 뷰는 모델에 담겨있는 데이터를 사용해서 화면에 그리는 역할을 한다
- 컨트롤러는 사용자의 입력에 대한 응답으로 모델, 뷰를 업데이트하는 방법을 정의한다.

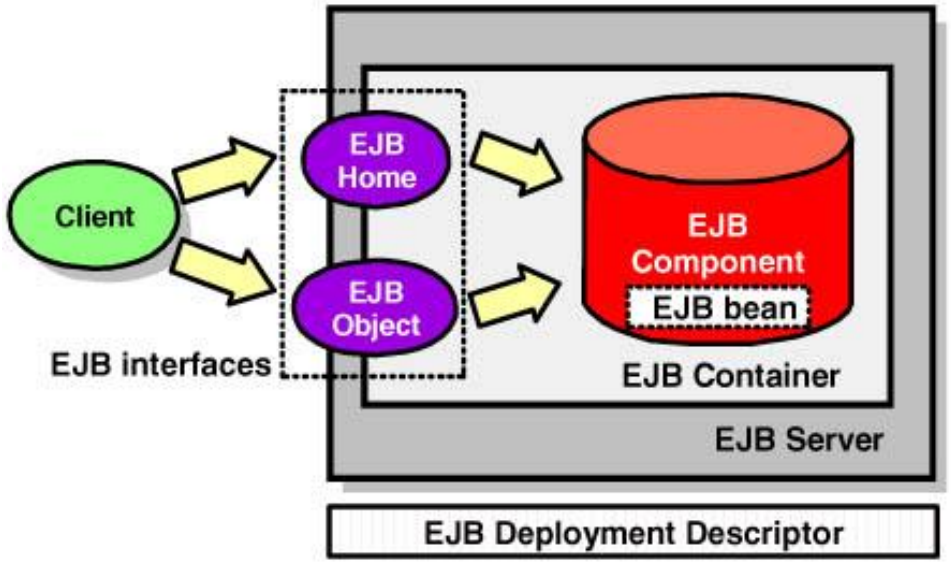
MVC Architecture



EJB

EJB(Enterprise Java Bean)

- EJB는 기업환경의 시스템을 구현하기 위한 서버 측 컴포넌트 모델
- 개발을 하다 보면 많은 객체들을 만들게 되는데, 이러한 비즈니스 객체들을 관리하는 컨테이너를 만들어서 필요할 때마다 컨테이너로부터 객체를 받는 식으로 관리하면 효율적일 것이라는 생각에서 탄생



Java Bean

- 특정한 정보(id, password, name, job...) 등을 가지고 있는 클래스를 표현하는 하나의 규칙이고, 데이터를 표현하기 위한 목적을 지니고 있음
- 이러한 규칙을 지닌 클래스를 Java Bean이라고 함

Java Bean의 규약

- 반드시 클래스는 패키지화 되어야 함
- 멤버변수는 property(프로퍼티)라고 함
- 멤버변수는 private로 지정하고, 외부접근을 위한 get, set 메소드를 정의해야 함
- get, set 메소드는 public으로 지정

Java Bean

```
public class JavaBean_Test {

    private String id;
    private String password;
    private String email;
    private String name;
    private String address;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
```

EJB의 장점

- 생명주기 관리
- 보안
- 트랜잭션 관리
- 오브젝트 풀링

EJB의 한계

- 객체지향적이지 않음
- 복잡한 프로그래밍 모델
- 특정 환경, 기술에 종속적인 코드
- 컨테이너 안에서만 동작할 수 있는 객체구조
- 자동화된 테스트가 매우 어렵거나 불가능
- 부족한 개발생산성, 이동성

Spring Framework

Spring Framework

- Spring Framework는 자바 플랫폼을 위한 오픈소스 애플리케이션 프레임워크
- 대규모 데이터 처리와 트랜잭션이 동시에 여러 사용자로부터 행해지는 애플리케이션을 개발하기 위한 모든 기능을 종합적으로 제공하는 솔루션
- EJB를 대체하는 새로운 자바 기반 프레임워크

Spring Framework의 주요 특징

- POJO(Plain Old Java Object)
- AOP(Aspect Oriented Programming)
- DI(Dependency Injection)
- IoC(Inversion of Control)

POJO(Plain Old Java Object)

- POJO는 오래된 방식의 자바 오브젝트라는 말로서 Java EE 등의 중량 프레임워크들을 사용하게 되면서 해당 프레임워크에 종속된 무거운 객체를 만들게 된 것에 반발해서 사용하게 된 용어
- 오래된 방식의 간단한 오브젝트란 특정 기술에 종속되어 동작하는 것이 아닌 순수한 자바 객체

예를들어, ORM(Object Relationship Mapping)이 새롭게 등장 했을 때를 생각해보겠습니다. ORM 기술을 사용하고 싶었다면 ORM을 지원하는 ORM 프레임워크를 사용해야 합니다. (대표적으로 Hibernate라는 프레임워크가 있습니다.) 만약 자바 객체가 ORM 기술을 사용하기 위해서 Hibernate프레임워크를 직접 의존하는 순간! POJO라고 할 수 없습니다. 특정 '기술'에 종속되었기 때문입니다.

POJO(Plain Old Java Object)

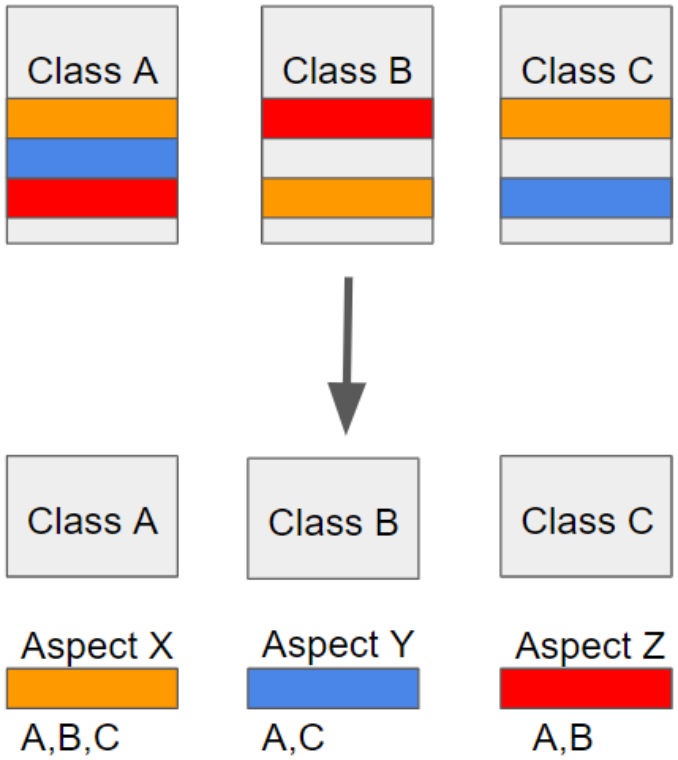
- 특정 기술과 환경에 종속되어 의존하게 된 자바 코드는 가독성이 떨어져 유지보수가 어렵다
- 특정 기술의 클래스를 상속받거나, 직접 의존하면 확장성이 매우 떨어진다
- 즉, 객체지향의 화신인 자바가 객체지향 설계의 장점들을 잃어버리지 않기 위해 POJO를 지향해야 한다

POJO(Plain Old Java Object) - PSA

- POJO이면서 특정 기술을 사용할 수 있는 이유는 Spring Framework에서 정한 표준 인터페이스가 있기 때문
- 예를 들어 ORM이라는 기술을 사용하기 위해서 JPA라는 표준 인터페이스를 정해 둠
- 이런 방법을 Spring의 PSA(Portable Service Abstraction)라고 함

AOP(Aspect Oriented Programming)

- AOP는 어떤 로직을 기준으로 핵심적인 관점, 부가적인 관점으로 나누어서 보고 그 관점을 기준으로 각각 모듈화 하는 개발 방법



- A, B, C클래스에서 동일한 색의 선들의 의미는 클래스들에 나타나는 비슷한 메소드, 필드, 코드
- 이런 식으로 반복되는 코드를 흩어진 관심사(Crosscutting Concerns)라고 부른다.
- AOP는 흩어진 관심사를 Aspect를 이용해서 해결

AOP(Aspect Oriented Programming)

- 결론적으로 Aspect로 모듈화하고 핵심적인 비즈니스 로직에서 분리하여 재사용하겠다는 것이 AOP의 취지

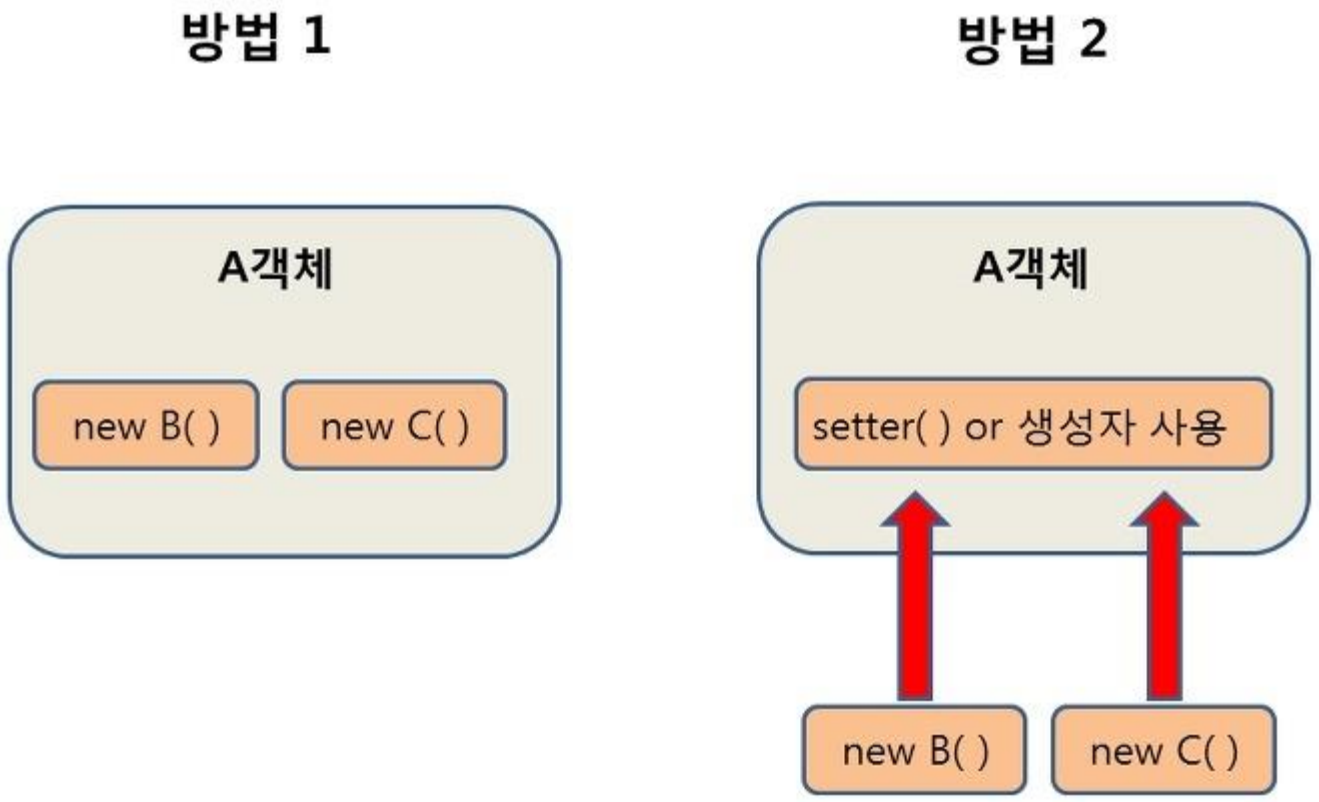
DI(Dependency Injection)

- 객체의 의존성 해결 및 관리 목적

```
public class A(){
        public static void main(String[] args){
                B b = new B(); //A클래스는 B클래스를 사용한다. 즉, A는 B에 의존한다(의존적이다).
                b.hello();
        }
}

class B(){
        public void hello(){
                system.out.print("hello");
        }
}
```

- 의존성 주입이란 사용자가 직접 new 키워드를 사용하여 객체를 생성하지 않고, 외부(컨테이너)에서 생성된 객체를 주입 받는 방식

DI(Dependency Injection)



DI(Dependency Injection) - 의존

```java
@Entity
public class Member {

    @Id @GeneratedValue
    @Column(name = "member_id")
    private Long id;
    private String username;
    private int age;

    public Member() {
    }

}
```

```java
@Service
@Transactional(readOnly = true)
public class MemberService {

    private Member member = new Member();

    @Transactional
    public void join() {
        save(member);
    }

}
```

DI(Dependency Injection) – 의존 주입

```java
@Service
@Transactional(readOnly = true)
public class MemberService {

    private Member member;

    @Transactional
    public Long join() {
        save(member);
    }

    public void setMember(Member member) {
        this.member = member;
    }
}
```

```java
@Controller
@RequireArgumentConstructor
public class MemberController {

    private final MemberService service;

    @PostMapping("/join")
    public void join() {
        Member member = new Member();
        service.setMember(member)
        serivce.join();
    }

}
```

Spring DI

```java
@Service
@Transactional(readOnly = true)
public class MemberService {

    private Member member;

    @Autowired
    public MemberService(Member member) {
        this.member = member
    }

    @Transactional
    public void join() {
        save(member);
    }

}
```

- Spring에서의 DI는 Bean으로 등록된 객체만 의존 주입이 가능
- @Autowired 어노테이션을 통해서 선언된 type을 비교하고 bean을 자동으로 주입

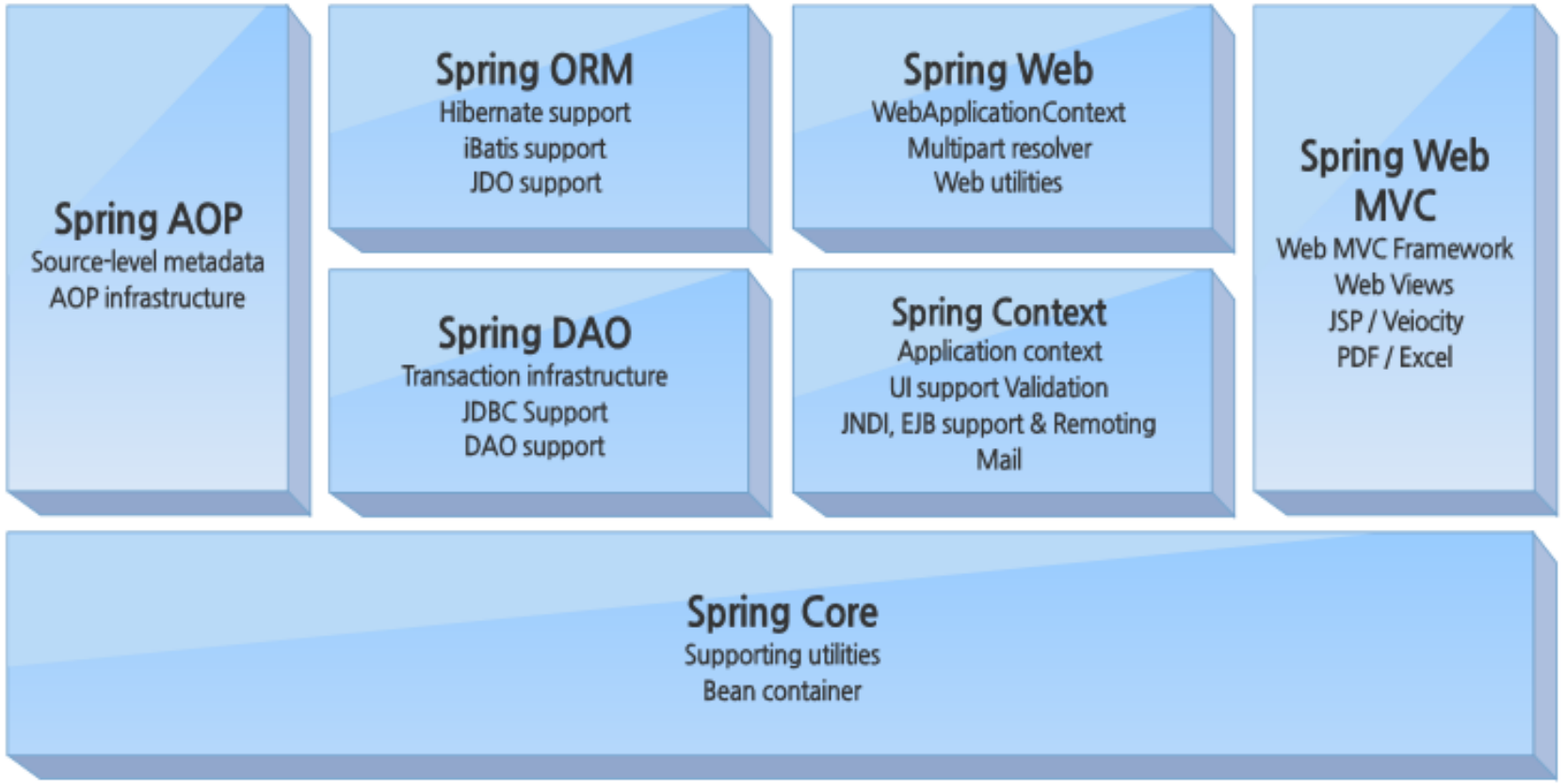
IoC(Inversion of Control)

- IoC란 제어의 역전이라는 의미로, 말 그대로 메소드나 객체의 호출작업을 개발자가 결정하는 것이 아니라, 외부에서 결정되는 것을 의미
- 객체의 의존성을 역전시켜 객체 간의 결합도를 줄이고 유연한 코드를 작성할 수 있게 하여 가독성 및 코드 중복, 유지 보수를 편하게 할 수 있다

IoC(Inversion of Control)

- 기존의 객체 생성 순서
  - 객체 생성
  - 의존성 객체 생성
  - 의존성 객체 메소드 호출
- Spring에서 객체 생성 순서
  - 객체 생성
  - 의존성 객체 주입
  - 의존성 객체 메소드 호출

Spring Framework의 구조



Spring Boot

Spring Boot

- Spring Framework는 기능이 많은 만큼 환경설정이 복잡한 편
- Spring Boot는 Spring Framework를 사용하기 위한 설정의 많은 부분을 자동화하여 사용자가 편하게 Spring을 사용할 수 있도록 돕는다

Spring Boot 장점

- Embed Tomcat을 사용하기 때문에 Tomcat을 설치하거나 버전관리의 수고로움을 덜어줌
- Dependency 자동화
- Xml 설정을 할 필요가 없다
- Jar file을 이용해 자바 옵션만으로 쉽게 배포가 가능하다