14 : 패턴 & 스트럿츠

학습 목표

- 디자인 패턴에 대해서 알아본다
- 웹 애플리케이션에 디자인 패턴을 적용한다.
- 스트럿츠에 대해 알아보고 그 기능 및 구성 요소에 대해 알아본다.

1. 디자인 패턴

2. 스트럿츠

패턴 >> 디자인 패턴

■ 소프트웨어 디자인 패턴

- 일반적이로 발생하는 소프트웨어 문제를 해결하기 위한 반복적인 해결방식
- 재사용 가능
- 비기능적인 요구사항 처리
  - 성능 **(Preformance)**
  - 모듈화 **(Modularity)**
  - 유연성 **(Flexibility)**, 유지보수성 **(Maintainablility)**, 확장성 **(Extensibility)**

패턴 >> 디자인 패턴

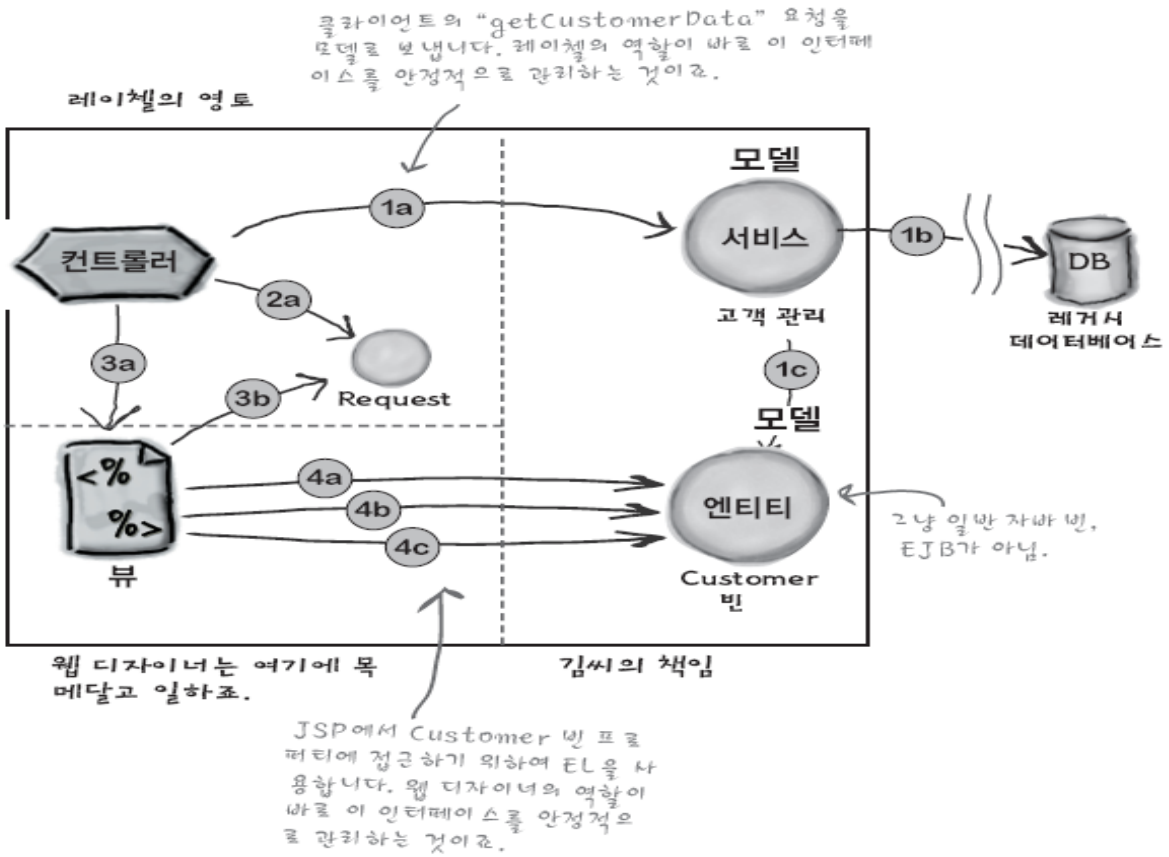
■ 디자인 원칙

- 인터페이스를 사용하라
- 관심 영역의 분리 및 응집성
- 복잡성을 숨겨라
- 느슨한 결합도 (Loose Coupling)
- 원격 프록시
- 선언적인 제어를 많이 사용하라

패턴 >> 디자인 패턴

■ 모든 구성요소가 로컬인 경우의 MVC 패턴

클라이언트가 고객 정보를 요구할 때...



1 고객 정보에 대한 요청을 접수 받으면, **컨트롤러**는 레거시 데이터베이스에 대하여 JDBC 호출을 하는 **고객 관리**(**모델**임) 서비스 컴포넌트를 호출합니다. 그 다음 **Customer** 빈을 만들어(EJB가 아닌 일반 자바 빈임), 데이터베이스로부터 읽어온 고객 정보를 설정합니다.

2 컨트롤러는 request 객체에 속성으로 Customer 빈 참조를 추가합니다.

3 컨트롤러는 이제 뷰인 JSP로 요청을 넘깁니다(forward). JSP는 request 객체에 들어있는 Customer 빈에 대한 참조를 사용할 수 있게 됐죠.

4 JSP는 EL을 사용하여 Customer 빈의 프로퍼티 정보를 뽑아내어 페이지를 완성합니다.

패턴 >> 디자인 패턴

■ 원격 객체인 경우 : JNDI 와 RMI 이용

- **JNDI : Java Naming & Directory Interface**

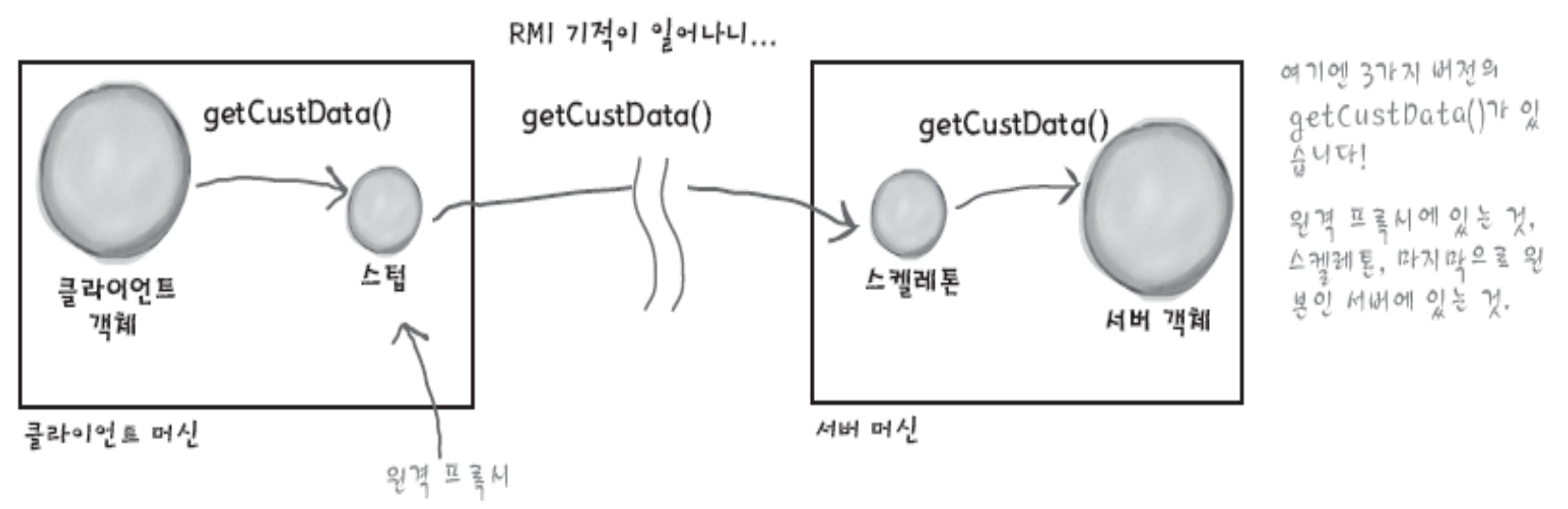
  네트워크에 존재하는 객체에 대한 위치 제공

- **RMI : Remote Method Invocation**

  원격에 있는 객체의 메소드 실행

패턴 >> 디자인 패턴

RMI 구조



패턴 >> 디자인 패턴

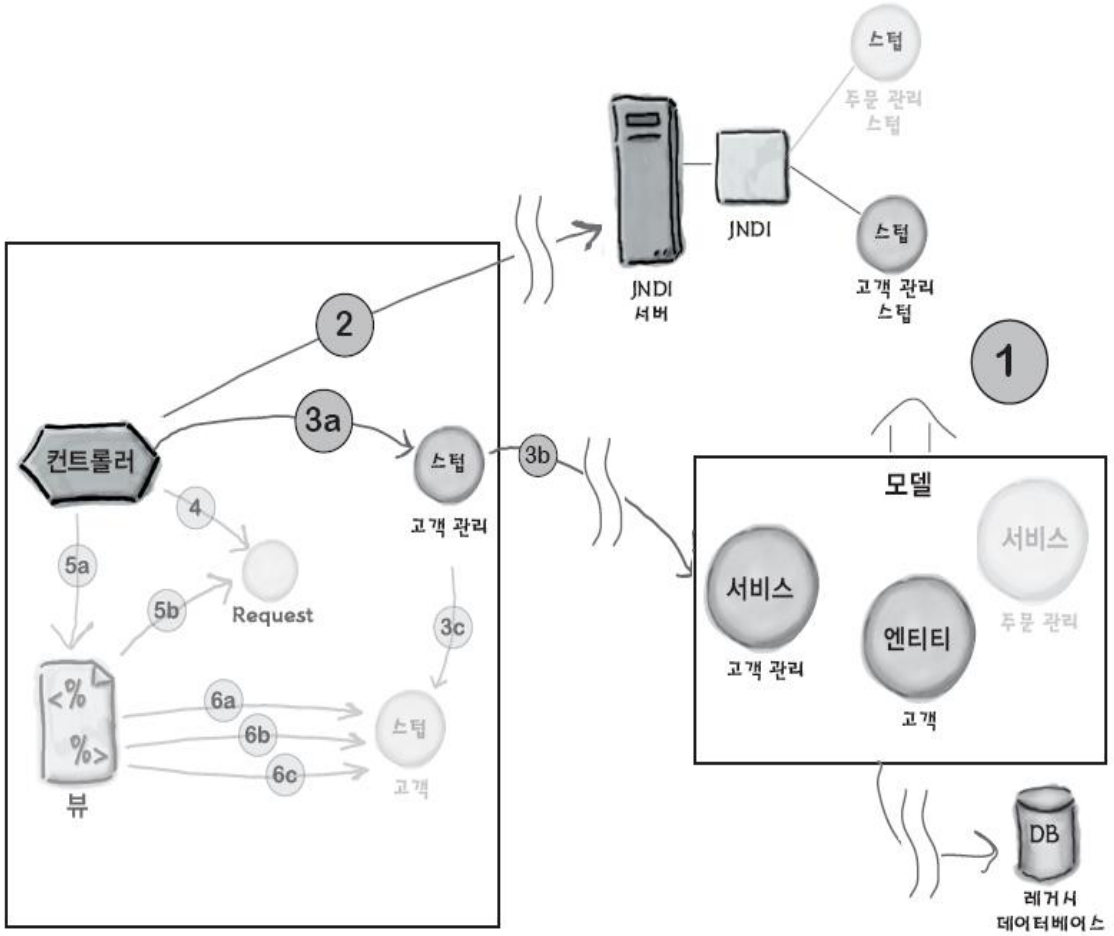
원격인 경우의 MVC 패턴 (1)

원격 객체를 사용하기 위한 3단계

① 모델 관리자 김씨가 JNDI 서비스에 자신이 관리하는 모델 컴포넌트를 등록합니다.

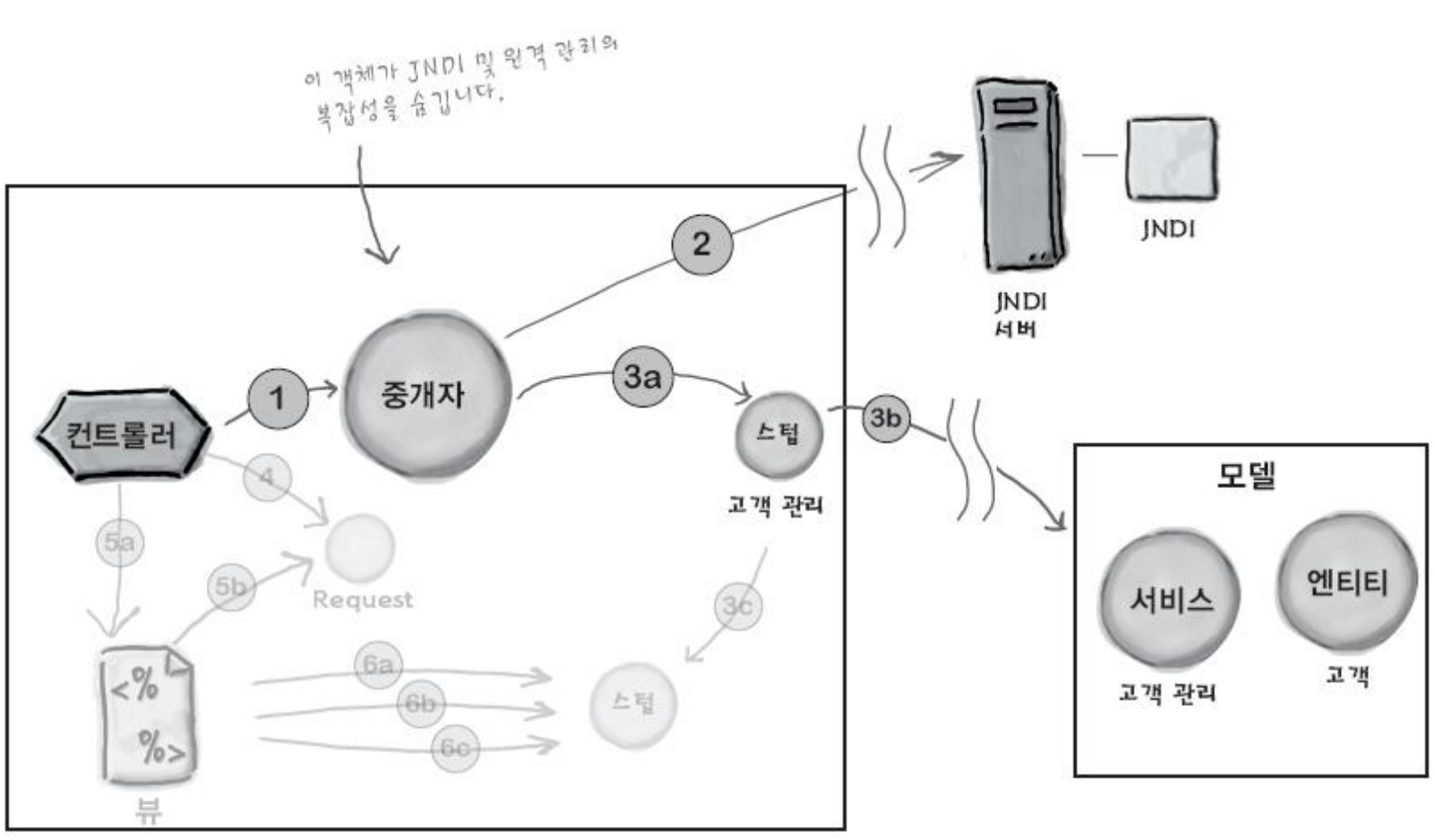
② 레이첼 컨트롤러가 요청을 받으면, 컨트롤러는 김씨가 관리하는 원격 모델 서비스에 대한 스텁 프록시를 찾기 위하여 JNDI 검색(lookup)을 합니다.

③ 컨트롤러가 스텁을 통하여 비즈니스 메소드 호출을 합니다. 스텁을 통해서 실제 모델 객체를 호출하겠죠.



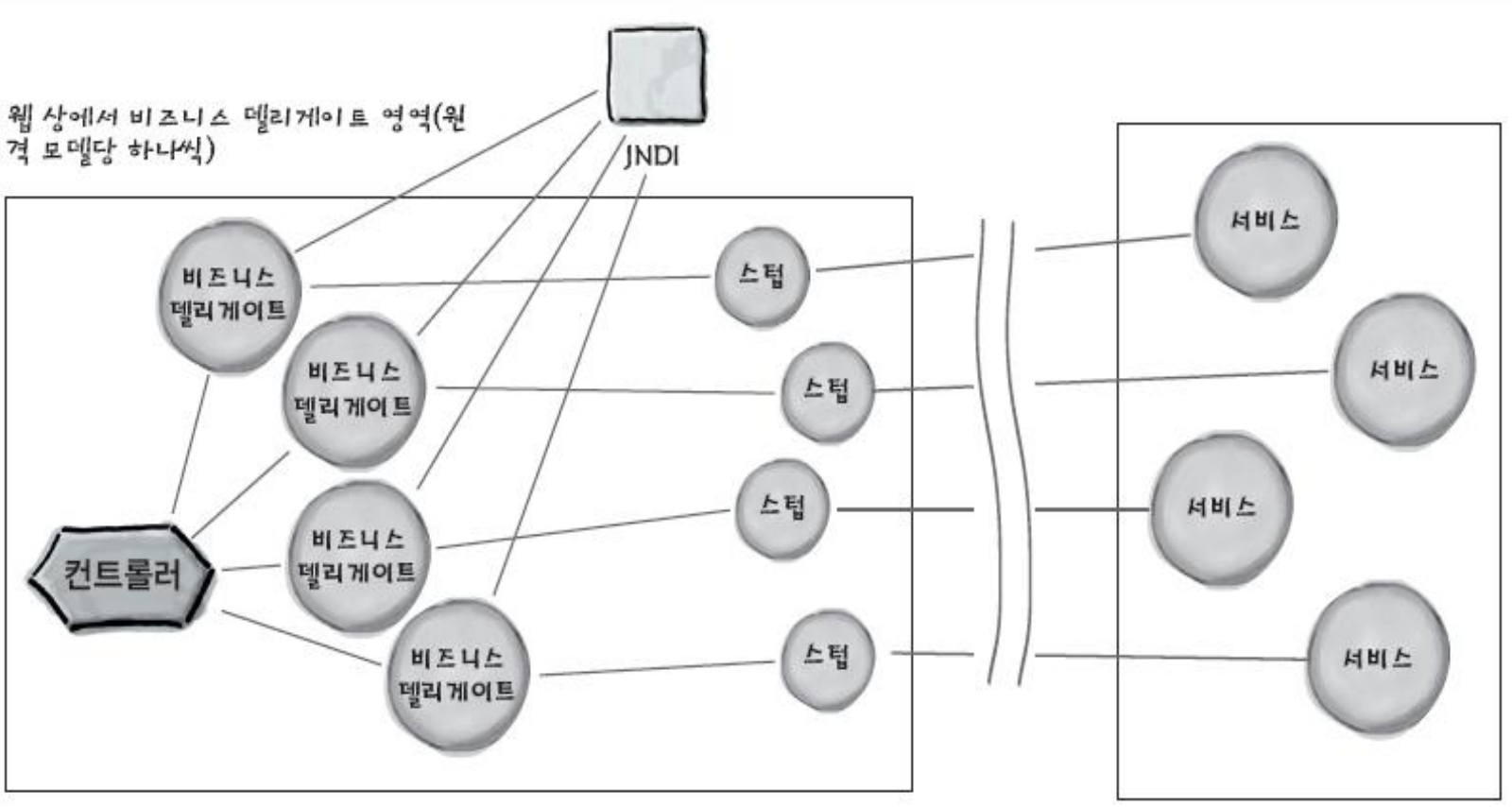
패턴 >> 디자인 패턴

■ 원격인 경우의 MVC 패턴 (2) : JNDI 객체 숨기기



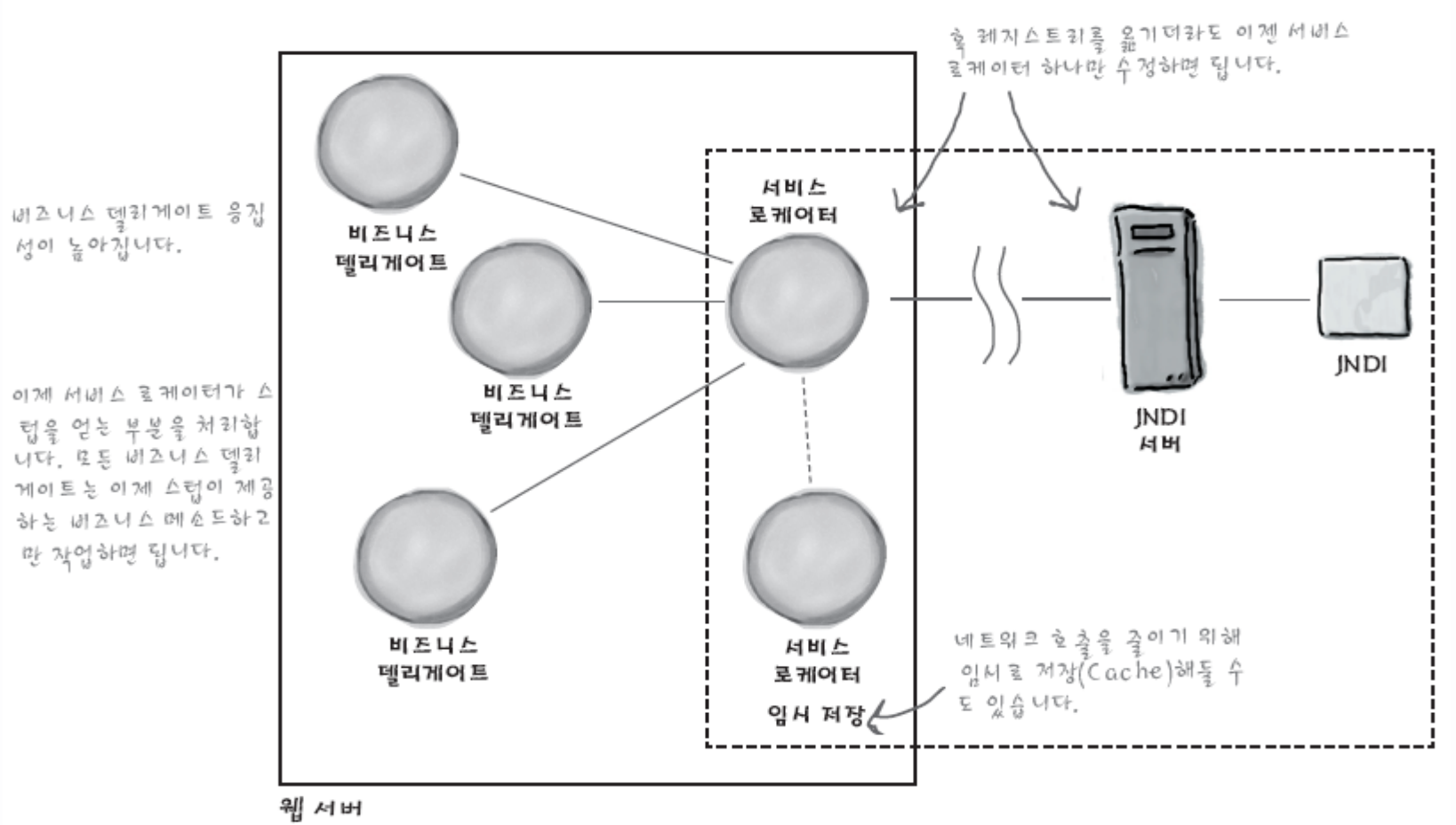
패턴 >> 디자인 패턴

■ 비즈니스 델리게이트(중개자)



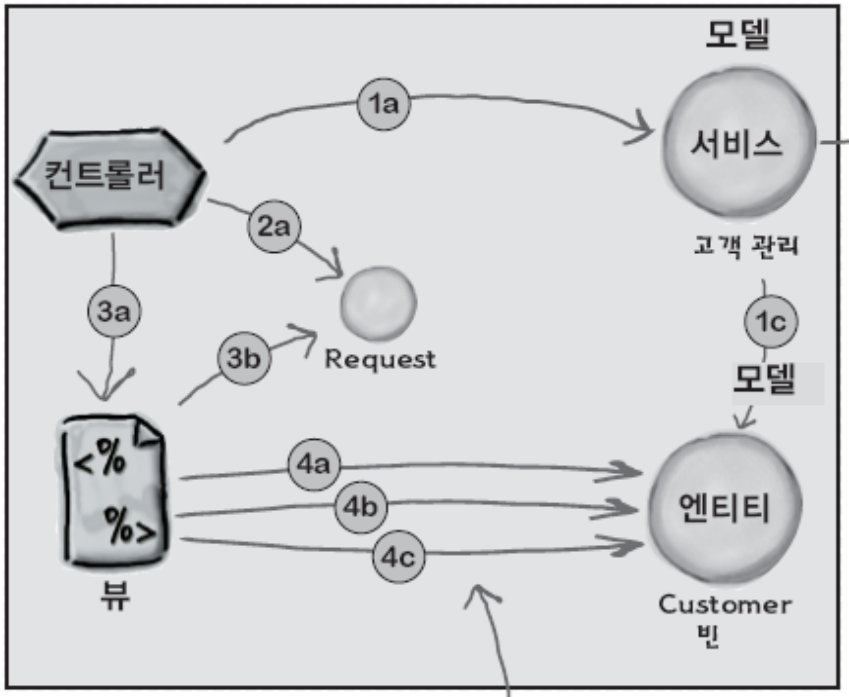
패턴 >> 디자인 패턴

■ 서비스 로케이터 : 비즈니스 델리게이트 단순화



패턴 >> 디자인 패턴

■ 로컬 JSP 에 대한 패턴



1 고객 정보에 대한 요청을 받으면, 컨트롤러는 고객 관리 모델 컴포넌트를 호출합니다. 호출받은 모델 컴포넌트는 Customer 빈 객체를 만들어, 레거시 데이터베이스에 대하여 원격 호출을 하고 빈 객체 정보를 채웁니다.

2 컨트롤러는 request에 속성으로 Customer 참조를 추가합니다.

3 컨트롤러는 요청을 뷰인 JSP로 넘깁니다(forward). JSP는 request 객체에서 Customer 객체 참조를 얻습니다.

4 JSP는 원래 요청한 의도대로 필요한 Customer 빈 프로퍼티 정보를 EL로 추출하여 페이지를 완성합니다.

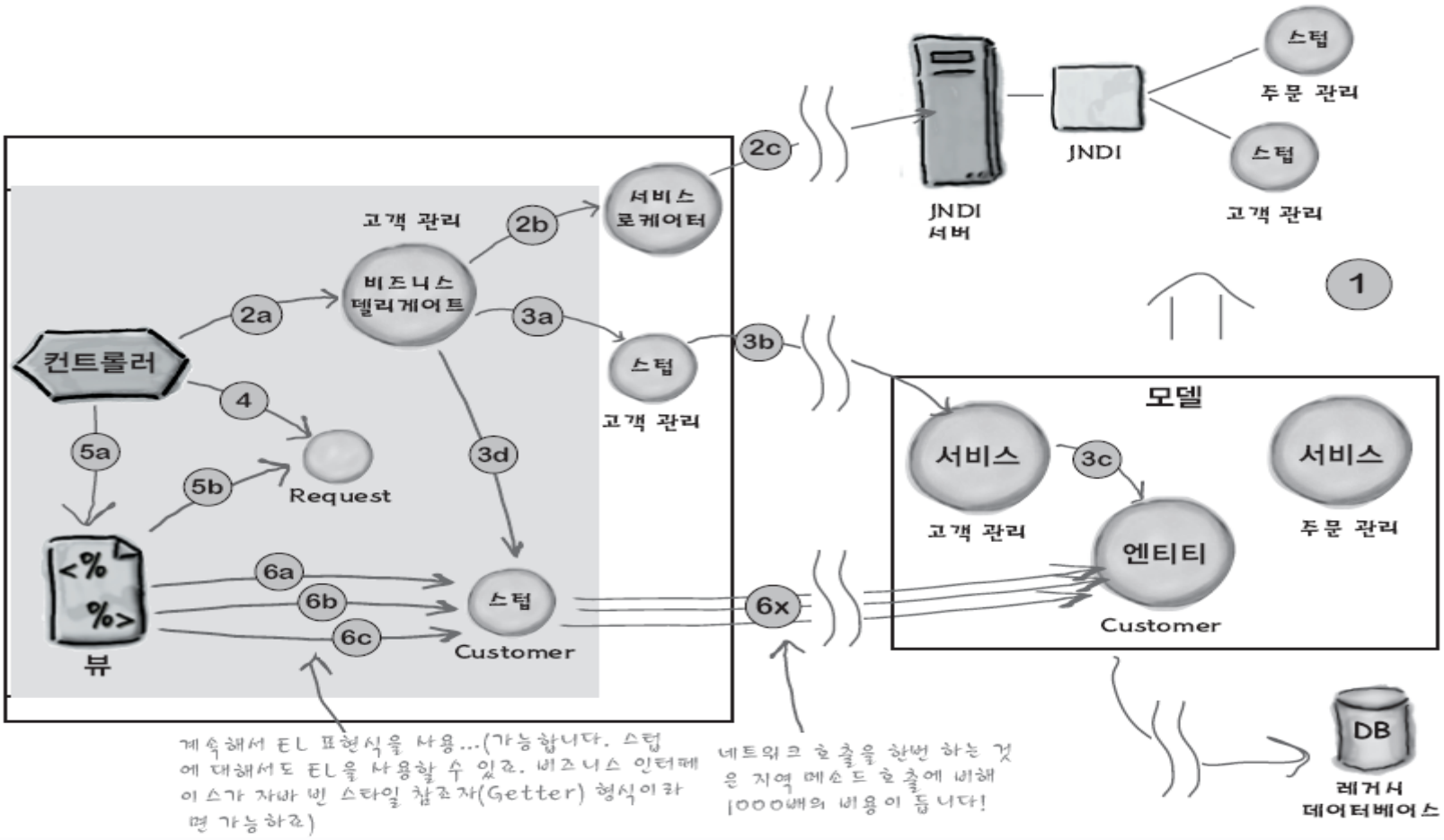
이건 아주 간단하죠.

EL로 다음과 같이 작성하면 끝나죠:

```
${customer.name}
```

패턴 >> 디자인 패턴

■ 원격 JSP 에 대한 패턴 (1)



패턴 >> 디자인 패턴

■ 원격 JSP 에 대한 패턴 (2)

A 6 단계 리뷰:

1 서비스를 JNDI에 등록합니다.

2 비즈니스 델리게이터와 서비스 로케이터를 사용하여 JNDI에서 고객 관리 스텁을 얻습니다.

3 비즈니스 델리게이터와 스텁을 사용하여 Customer 빈을 얻습니다. 여기서 Customer 빈은 또다른 하나의 스텁이죠. 이 스텁 참조를 컨트롤러로 리턴합니다.

4 request에 스텁 참조를 추가합니다.

5 컨트롤러는 요청을 JSP로 넘깁니다(forward). JSP는 request 객체에서 Customer 빈(스텁) 참조를 얻습니다.

6 JSP는 원래 요청한 의도대로 필요한 Customer 빈 프로퍼티 정보를 EL로 추출하여 페이지를 완성합니다.

중요*2: JSP가 Customer 스텁 참조자(Getter)를 호출하면 이는 네트워크 호출임을 잊지 마세요.

패턴 >> 디자인 패턴

■ 트렌스퍼 오브젝트 (Transfer Object)

- 비즈니스 서비스에서 데이터 전송 시 사용
- 직렬화된 자바 객체 사용

클라이언트 관점에서, 비즈니스 델리게이트 내부:

빈/트랜스퍼 오브젝트 유형

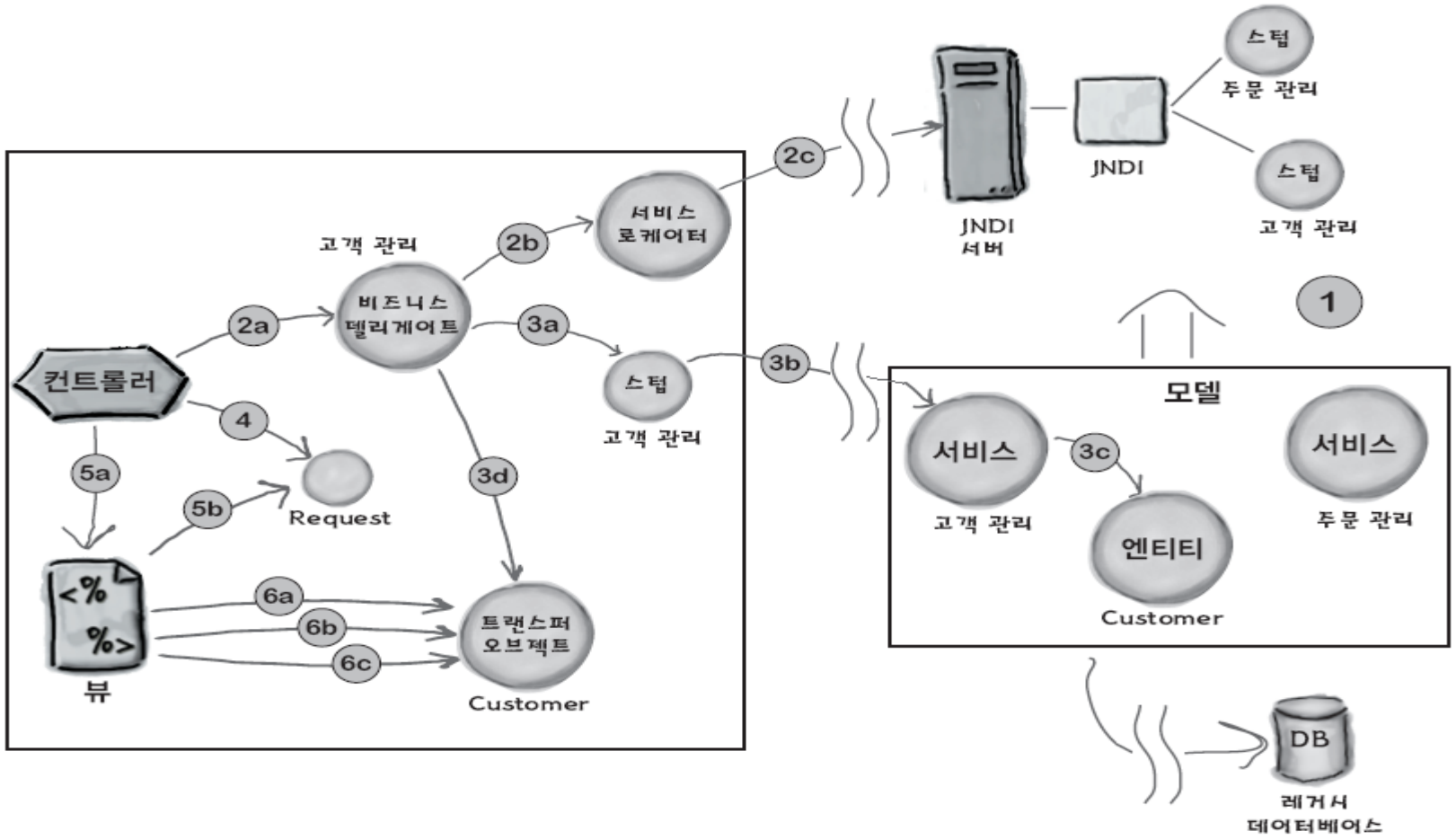
스텁으로부터 트랜스퍼 오브젝트를 요청함.

```
try {
  Customer c = custStub.getCustData(custID);
} catch (RemoteException re) {
  throw new CustomerException();
}
```

원격 예외사항을 잡아서 이를 쉽게 이해할 수 있는 예외사항으로 감쌈.

패턴 >> 디자인 패턴

■ 비즈니스 티어 패턴 (1)



패턴 >> 디자인 패턴

■ 비즈니스 티어 패턴 (2)

A 6 단계 리뷰:

1 서비스를 JNDI에 등록합니다.

2 비즈니스 델리게이트와 서비스 로케이터를 사용하여 JNDI에서 고객 관리 스텁을 얻습니다.

3 비즈니스 델리게이트와 스텁을 사용하여 Customer 빈을 얻습니다. 여기서 Customer 빈은 트랜스퍼 오브젝트죠. 이 스텁 참조를 컨트롤러로 리턴합니다.

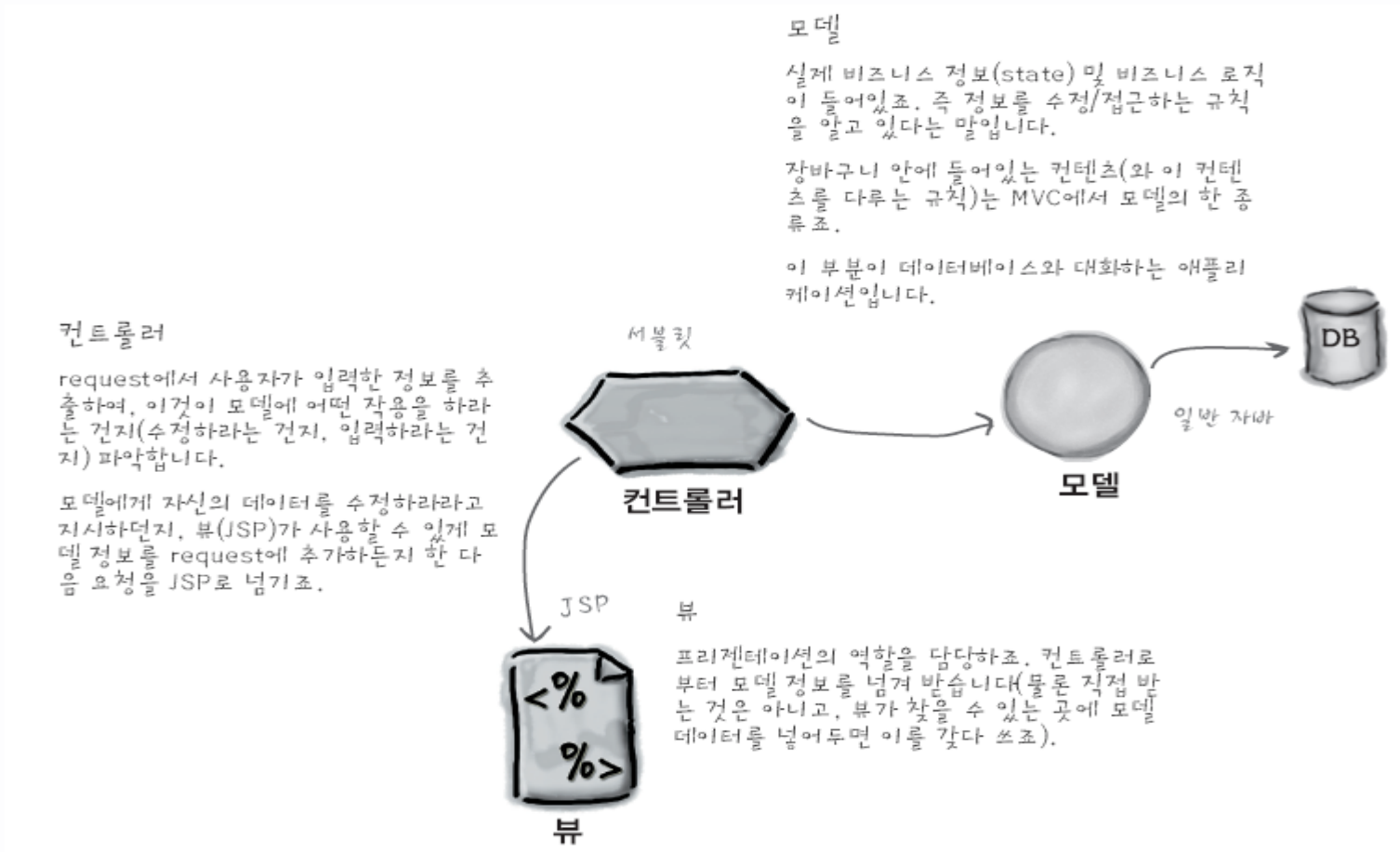
4 request에 빈 참조를 추가합니다.

5 컨트롤러는 요청을 JSP로 넘깁니다(forward). JSP는 request 객체에서 Customer 트랜스퍼 오브젝트 참조를 얻습니다.

6 JSP는 원래 요청한 의도대로 필요한 Customer 트랜스퍼 오브젝트의 프로퍼티 정보를 EL로 추출하여 페이지를 완성합니다.

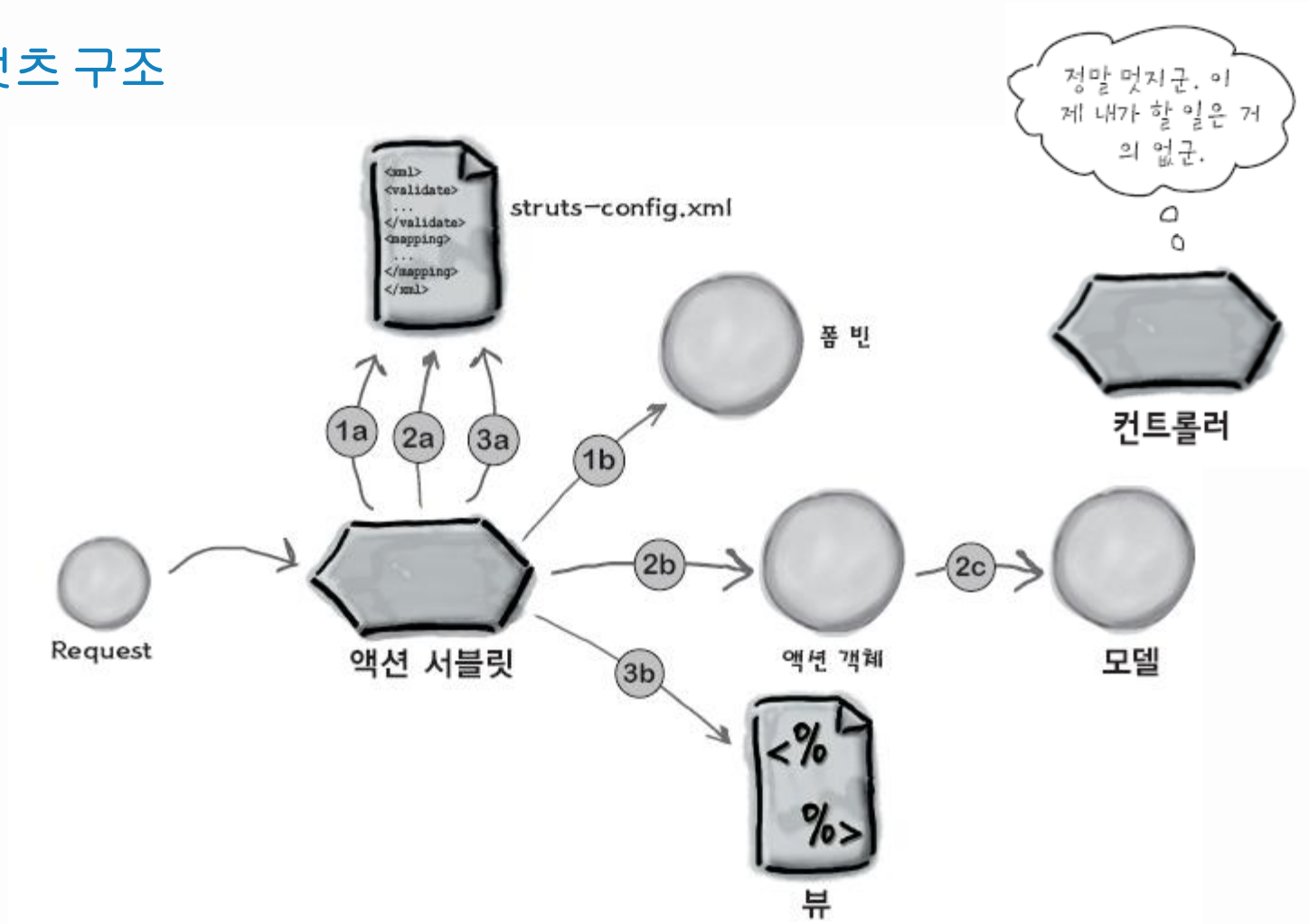
패턴 >> 디자인 패턴

MVC 디자인 패턴



스트럿츠 >> 스트럿츠 구조

■ 스트럿츠 구조



스트럿츠 >> 주요 컴포넌트

액션 서블릿

- 애플리케이션 당 하나만 존재, 스트럿츠에서 제공

폼 빈

- 애플리케이션이 처리해야 하는 **HTML** 폼 당 하나씩 작성
- 사용자가 입력한 폼 데이터 검증 가능

액션 객체

- 일반적으로 하나의 액션은 유스케이스에 있는 활동 하나에 매핑됨
- **execute()** : 폼 파라미터 검증, 모델 컴포넌트 호출

struts-config.xml

- 스트럿츠 배포 서술자

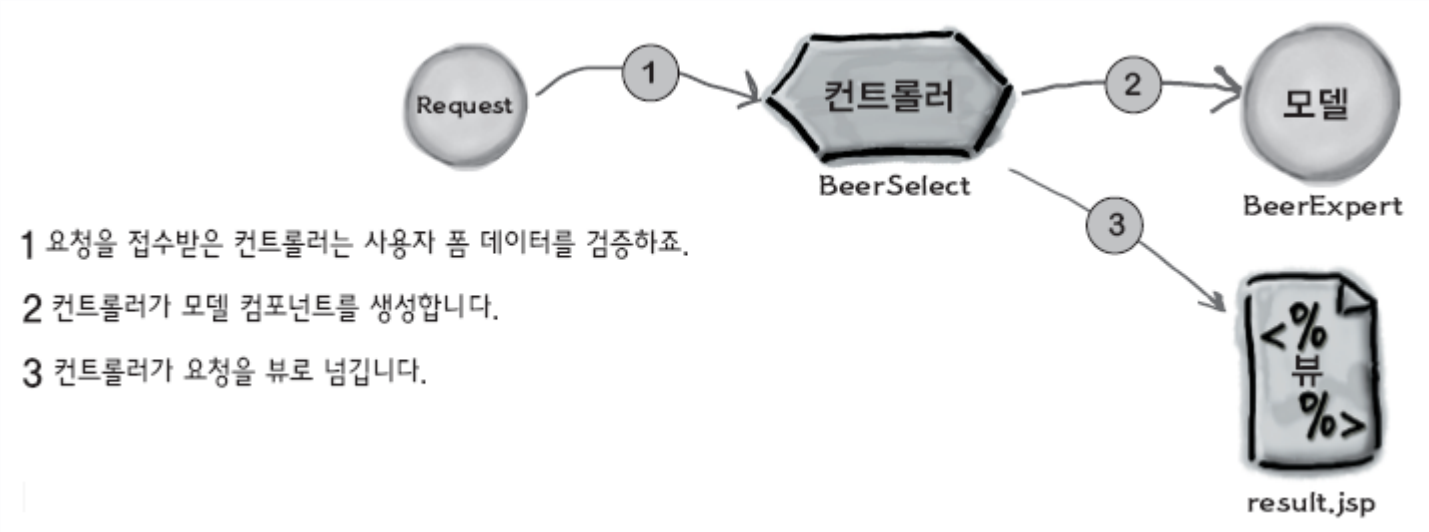
스트럿츠 >> 프론트 컨트롤러

스트럿츠에 있는 프론트 컨트로러 : RequestProcess

- 선언적인 제어 : 요청 **URL**, 검증 객체, 모델을 생성하는 객체, 뷰간의 매핑
- 자동화된 요청 디스패칭 : **ActionFoward** 를 이용. 컨트롤러와 뷰 컴포넌트간의 느슨한 결합을 가능하게 함
- **DataSource** 관리
- 커스텀 태그 제공
- 국제화 지원
- 선언적인 검증
- 전역 예외처리
- 플러그인

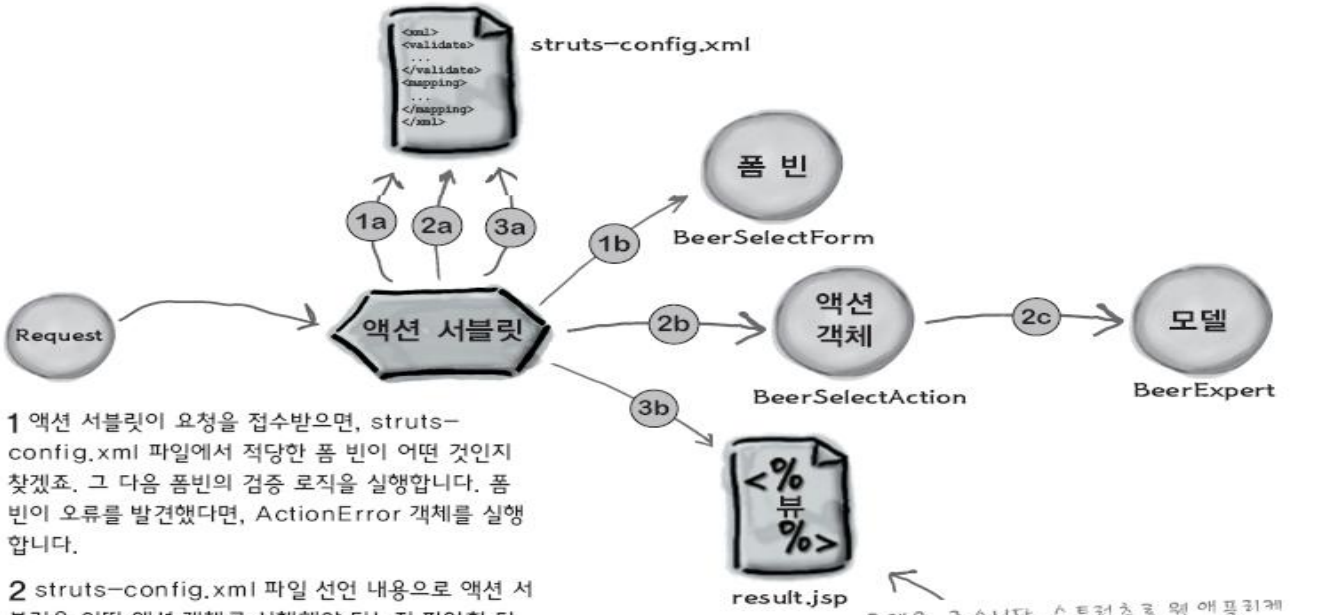
스트럿츠 >> 스트럿츠로 리펙토링

■ 기존 서블릿 구조



스트럿츠 >> 스트럿츠로 리펙토링

■ 스트럿츠로 리펙토링



1 액션 서블릿이 요청을 접수받으면, struts-config.xml 파일에서 적당한 폼 빈이 어떤 것인지 찾겠죠. 그 다음 폼빈의 검증 로직을 실행합니다. 폼 빈이 오류를 발견했다면, ActionError 객체를 실행합니다.

2 struts-config.xml 파일 선언 내용으로 액션 서블릿은 어떤 액션 객체를 실행해야 되는지 파악한 다음, 이를 실행합니다. 그 다음 액션 객체는 모델을 생성하고 액션 서블릿에 ActionForward 객체를 리턴하죠.

3 struts-config.xml 파일에서 이미 추출한 정보로 액션 서블릿은 어떤 뷰를 호출할지 이미 알고 있죠. 이 정보에 기초해서 액션 서블릿은 ActionForward 객체를 이용하여 뷰로 요청을 넘깁니다.

그래요, 좋습니다. 스트럿츠로 웹 애플리케이션을 바꾸면, 뷰도 수정해야 합니다. 예를 들면, 스트럿츠는 <html:errors/>와 같은 태그를 포함하는 태그 라이브러리를 제공합니다. <html:errors/> 태그는 폼 빈 검증 시 오류를 보여주는 것이죠. 또 HTML 태그 라이브러리에는 오류가 발생한 폼을 다시 입력할 수 있는 태그도 제공한다네요.

스트럿츠 >> 스트럿츠 정보 설정

DD (web.xml) 에 스트럿츠 정보 설정

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

  <!-- Define the controller servlet -->
  <servlet>
    <servlet-name>FrontController</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>

    <!-- Name the struts configuration file -->
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>

    <!-- Guarantee that this servlet is loaded on startup. -->
    <load-on-startup>1</load-on-startup>
  </servlet>

  <!-- The Struts controller mapping -->
  <servlet-mapping>
    <servlet-name>FrontController</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <!-- END: The Struts controller mapping -->

</web-app>
```

액션 서블릿 이름을 꼭 "FrontController"라고 지을 필요는 없죠. 하지만 어떤 목적으로 사용하는지 바로 감이 오도록 지으면 좋지 않나요.

"config"(init-param)은 액션 서블릿에게 스트럿츠 설정 파일이 어디 있는지 정보를 알려주죠.

액션 서블릿에는 복잡한 init 메소드가 있죠. 시작 시 이 서블릿을 로딩하는 것이 나을 듯...

와우! 이 서블릿 하나로 애플리케이션의 모든 요청을 다 처리할 셈이군요(요청 URL이 모두 ".do" 확장자라면....)