DefaultServlet Handler

DefaultServlet 위임

- JSTL 라이브러리를 pom.xml에 추가

```
<!-- jstl -->
<dependency>
<groupId>javax.servlet</groupId>
<artifactId>jstl</artifactId>
<version>1.2</version>
</dependency>
```

DefaultServlet 위임

- 정적 자원의 접근 실패
  - html, css, js 등의 파일 접근에 실패한다
  - DispatcherServlet이 모든 URL 처리에 서블릿 매핑을 하였기 때문에 Tomcat은 정적 자원에 대한 URL 처리도 DispatcherServlet에 넘기기 때문임
    (즉, DefaultServlet에 위임을 하지 못하는 상태)

- Spring MVC에서 DefaultServlet 위임 처리하기
  - HandlerMapping이 URL과 컨트롤러의 메서드(핸들러)와의 매핑 정보를 가지고 있다
  - HandlerMapping에서 정적 자원에 대한 URL은 DefaultServlet으로 위임할 수 있도록 설정해 주어야 한다
  - in spring-servlet.xml

```
<!-- validator, conversionService, messageConverter를 자동으로 등록 -->
<mvc:annotation-driven />
<!-- 서블릿 컨테이너의 디폴트 서블릿 위임 핸들러 -->
<mvc:default-servlet-handler/>
```

DefaultServlet 위임

- 정적 자원에 접근하기
  - jsp 파일에 jstl core tag 라이브러리를 로드한다

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

  - <c:url /> jstl tag로 정적 자원의 url에 접근

```
<link rel="stylesheet" href='<c:url value="/assets/css/bootstrap.min.css" />' />
<script language="javascript" type="text/javascript"
    src="<c:url value="/assets/bootstrap.min.js" />"></script>
```

View Resolver

ViewResolver의 이해

- ViewResolver는 HandlerMapping이 컨트롤러를 찾아주는 것처럼, View 이름을 가지고 View 객체를 찾아온다
- ViewResolver를 빈 등록하지 않으면 DispatcherServlet의 기본 ViewResolver인 InternalResourceViewResolver가 사용된다
- Default 사용에서는 View로 이동하는 전체 경로를 모두 적어줘야 한다
- prefix와 suffix를 지정하여 앞 뒤의 내용을 생략, 편리하게 View를 지정할 수 있다

ViewResolver 설정

- JSTL 라이브러리가 클래스패스에 존재하면 JstlView를 사용하고 아니면 InternalResourceView를 사용하면 된다
- 여러 ViewResolver를 등록할 수 있는데, order 프로퍼티를 지정하여 ViewResolver의 우선순위를 지정할 수 있다
- URL에 일정한 이름을 주면 View와 자동으로 연결시킬 수 있다

```
<!-- ViewResolver 설정 -->
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.sprintframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
    <property name="order" value="1" />
</bean>
```

ViewResolver 설정

- JSTL 라이브러리를 pom.xml에 추가

```
<!-- jstl -->
<dependency>
<groupId>javax.servlet</groupId>
<artifactId>jstl</artifactId>
<version>1.2</version>
</dependency>
```

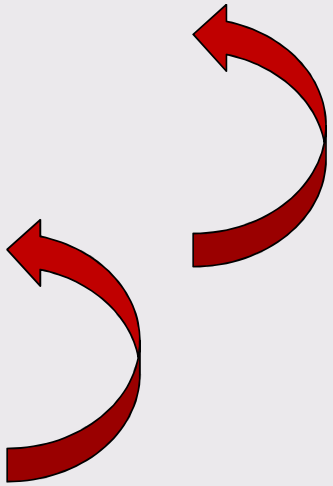
예외 처리

예외 처리
:예외 블랙홀

```
try {
    //...
} catch (Exception e) {
    //여기를 비워두는 것이 가장 나쁜 예외 처리
}
```

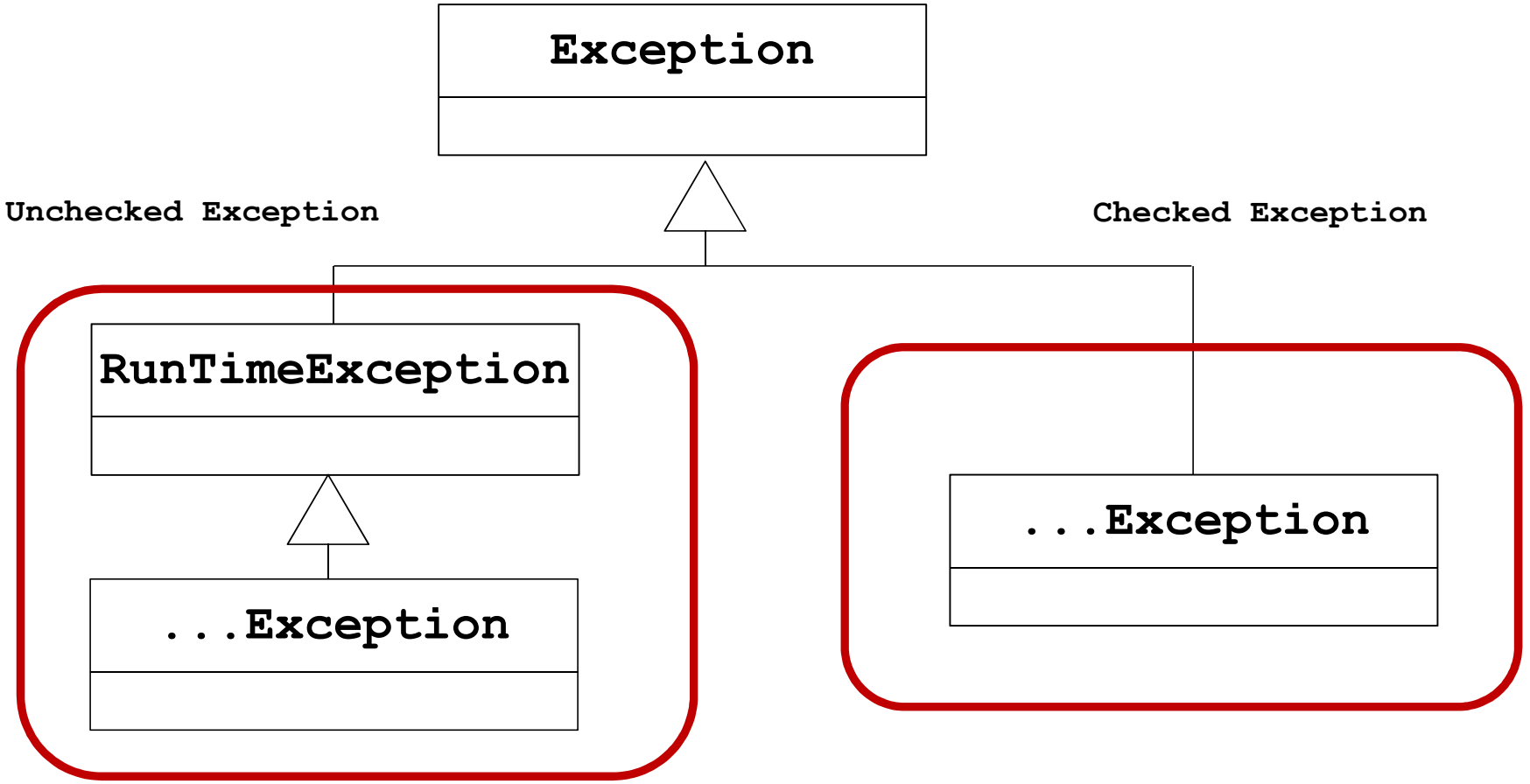
예외 처리
: 무의미하고 무책임한 throws (예외처리 회피)

```
public void method() throws Exception {
    method2();
}

public void method2() throws Exception {
    method3();
}

public void method3() throws Exception {
    method4();
}
```



예외 처리

: Checked 예외 처리 (상투적인 try ~ catch)



예외 처리 방법

- 예외 복구
  - 예외 상황을 파악하고 문제를 해결해서 정상 상태로 돌려 놓는 것
  - 예외를 어떤 식으로 복구 가능성이 있을 때 예외 처리를 강제하는 checked 예외를 사용할 수 있다
  - 예외는 복구 가능한가?

- 예외 처리 회피
  - throws 문을 선언하여 예외가 발생하면 외부로 던지게 한다
  - 또는, catch로 예외를 잡아 로그를 남기고 다시 예외를 던지는 방법
  - DAO가 SQLException을 외부로 던지면 서비스, 컨트롤은 처리 가능한가?

Spring에서의 예외 처리 방법

- 예외 전환
  - 대부분의 예외는 복구해서 정상적인 상태로 만들 수 없기 때문에 예외를 메서드 밖으로 던진다
  - 예외 처리 회피처럼 그대로 넘기지 않고 적절한 예외로 전환한다
  - Low Level의 예외 상황에 대한 적합한 의미를 가진 예외로 변경하는 것이 중요

```
try {
    //...
} catch (SQLException e) {
    throw new UserDaoException(e);
}
```

Spring에서의 예외 처리 방법

- 런타임 예외의 보편화
  - 예외는 미리 파악되어야 하며 예외가 발생하지 않는 것이 가장 좋다
  - 빨리 예외 발생 작업을 중지하고 서버 관리자나 개발자에게 통보해 주고 예외 내용을 로그로 남겨야 한다
  - 자바 엔터프라이즈 서버 환경에서의 체크 예외의 활용성은 Spring에서 다시 고려
  - **복구할 수 없는 예외는 Unchecked 예외로 만든다**
  - **Unchecked 예외도 필요시 catch가 가능하다**
  - **RuntimeException을 사용하여 전환, 포장하여 사용하도록 한다**

Spring에서의 예외 처리 방법

- Controller에서의 예외 처리
  - @ExceptionHandler를 사용하여 Exception과 핸들러를 매핑한다
  - Controller의 개발 핸들러 매서드에서 예외를 매핑하는 것보다 컨트롤러 어드바이스를 사용, 애플리케이션의 같은 종류의 예외를 처리하는 것이 효과적

```
@ExceptionHandler(UserDaoException.class)
public String handleUserDaoException() {
    return "/WEB-INF/views/error/exception.jsp";
}
```

Spring에서의 예외 처리 방법

- 어드바이징 컨트롤러에서의 예외 처리 (Spring 3.2 이상)
  - @ControllerAdvice는 @Component를 상속한 어노테이션이기 때문에 컴포넌트 스캐닝을 통해 선택됨
  - 실용적인 방법은 @ExceptionHandler를 사용하여 하나의 예외에 하나의 예외 핸들러를 묶는 방식
  - 핸들러에 @ResponseStatus를 사용하여 클라이언트의 응답 코드를 지정할 수 있다

```
@ControllerAdvice
public class ApplicationExceptionHandler {
    @ExceptionHandler(UserDaoException.class)
    public String handleDaoException(Exception e) {
        return "/WEB-INF/views/error/exception.jsp";
    }
}
```

Spring에서의 예외 처리 방법

- 다음 코드를 참고하여 예외 처리를 해 봅시다

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler( Exception.class )
    public ModelAndView handlerException(
        HttpServletRequest request,
        Exception e){

        // 1. 로깅
        e.printStackTrace();

        // 2. 시스템 오류 안내 화면
        ModelAndView mav = new ModelAndView();
        mav.setViewName( "error/exception" );

        return mav;
    }
}
```

Spring에서의 예외 처리 방법

- 애플리케이션 밖에서 발생하는 오류 안내 페이지
  - web.xml에 다음과 같은 설정을 한다

```
<!-- 공통 에러 페이지 -->
<error-page>
  <error-code>404</error-code>
  <location>/WEB-INF/views/error/404.jsp</location>
</error-page>

<error-page>
  <error-code>500</error-code>
  <location>/WEB-INF/views/error/500.jsp</location>
</error-page>
```