Day 11 (2019-02-25)

1. Spring Web Application 개발 - Session, Cookie, Interceptor, DateTimeFormat

(1) Cookie와 과 Session

Connection-less하고 State-less한 HTTP 프로토콜에서 A라는 클라이언트(사용자)를 어떻게 구별할까?

1) Cookie

● 클라이언트의 로컬에 키-값 형태의 데이터가 저장된 데이터 파일로 유효시간을 지정할 수 있어
브라우저가 종료되어도 그 값을 사용할 수 있다

● 하나의 도메인에서 20개의 값을 가질 수 있으며 총 300개를 저장할 수 있다.

● Response Header에 Set-Cookie attribute를 통해 생성할 수 있으며 클라이언트에서 별다른 처리 없이
브라우저가 해당 값을 알아서 서버에 전송한다

● 클라이언트 측에 데이터가 저장되기 때문에 쿠키 변조 등에 대한 공격에 취약하다

2) Session

● 사용자가 웹 브라우저를 통해 서버에 접속한 시점부터 웹 브라우저를 종료하는 시간 동안
발생하는 요청을 하나의 상태로 보고 일정하게 유지시키는 기술

● 사용자의 요청에 따른 정보를 클라이언트 측에 저장하는 것이 아닌 서버에 저장한다

● Cookie를 통해 세션에 대한 키 값(Session ID)을 관리한다

● 만료 시간을 지정할 수 있으며 브라우저가 종료되면 만료기관에 상관 없이 세션은 자동으로 삭제된다

● 서버 측에 데이터가 저장되기 때문에 많은 사용자가 접속하여 세션이 존재하는 경우 메모리가 부족할 수 있으며 Scale-out한 확장에 불리하다

※ LocalStorage, SessionStorage

Cookie와 비슷하게 key-value 형태의 데이터를 도메인별로 나누어 저장하는 HTML5의 웹스토리지로
만료시간을 지정할 수 없으며 단순 문자열이 아닌 Javascript Object를 저장 할 수 있다.

Cookie와 다르게 서버로 전송되지 않으며 Front-end Back-end 형태로 나뉜 Application을 개발할 때 주로 사용한다

서로다른 브라우저가 공유하지 않는 점을 주의해야 한다.

(2) Interceptor

1) Handler Process (preHandler, handler, postHandler)

```
try {
    processedRequest = checkMultipart(request);
    multipartRequestParsed = (processedRequest != request);

    // Determine handler for the current request.
    mappedHandler = getHandler(processedRequest);
    if (mappedHandler == null || mappedHandler.getHandler() == null) {
        noHandlerFound(processedRequest, response);
        return;
    }

    // Determine handler adapter for the current request.
    HandlerAdapter ha = getHandlerAdapter(mappedHandler.getHandler());

    // Process last-modified header, if supported by the handler.
    String method = request.getMethod();
    boolean isGet = "GET".equals(method);
    if (isGet || "HEAD".equals(method)) {
        long lastModified = ha.getLastModified(request, mappedHandler.getHandler());
        if (logger.isDebugEnabled()) {
            logger.debug("Last-Modified value for [" + getRequestUri(request) + "] is: " + lastModified);
        }
        if (new ServletWebRequest(request, response).checkNotModified(lastModified) && isGet) {
            return;
        }
    }

    if (!mappedHandler.applyPreHandle(processedRequest, response)) {
        return;
    }

    try {
        // Actually invoke the handler.
        mv = ha.handle(processedRequest, response, mappedHandler.getHandler());
    }
    finally {
        if (asyncManager.isConcurrentHandlingStarted()) {
            return;
        }
    }

    applyDefaultViewName(request, mv);
    mappedHandler.applyPostHandle(processedRequest, response, mv);
}
catch (Exception ex) {
    dispatchException = ex;
}
processDispatchResult(processedRequest, response, mappedHandler, mv, dispatchException);
```

preHandle	Handler가 실행되기 이전에 처리되는 인터셉터 메소드
postHandle	Handler 실행 이후 처리되는 인터셉터 메소드 비동기 처리가 되는 경우 실행되지 않을 수 있음 Handler 실행 중 에러가 발생하는 경우 실행되지 않을 수 있음
afterCompletion	최종적으로 요청에 대한 처리가 완료된 후에 호출되는 인터셉터 메소드 요청에 대한 처리 완료는 View rendering이 종료된 시점을 말하며, 주로 할당받은 Resource를 해제하는 처리를 이행한다

2) HandlerInterceptor 인터페이스와 HandlerInterceptorAdaptor

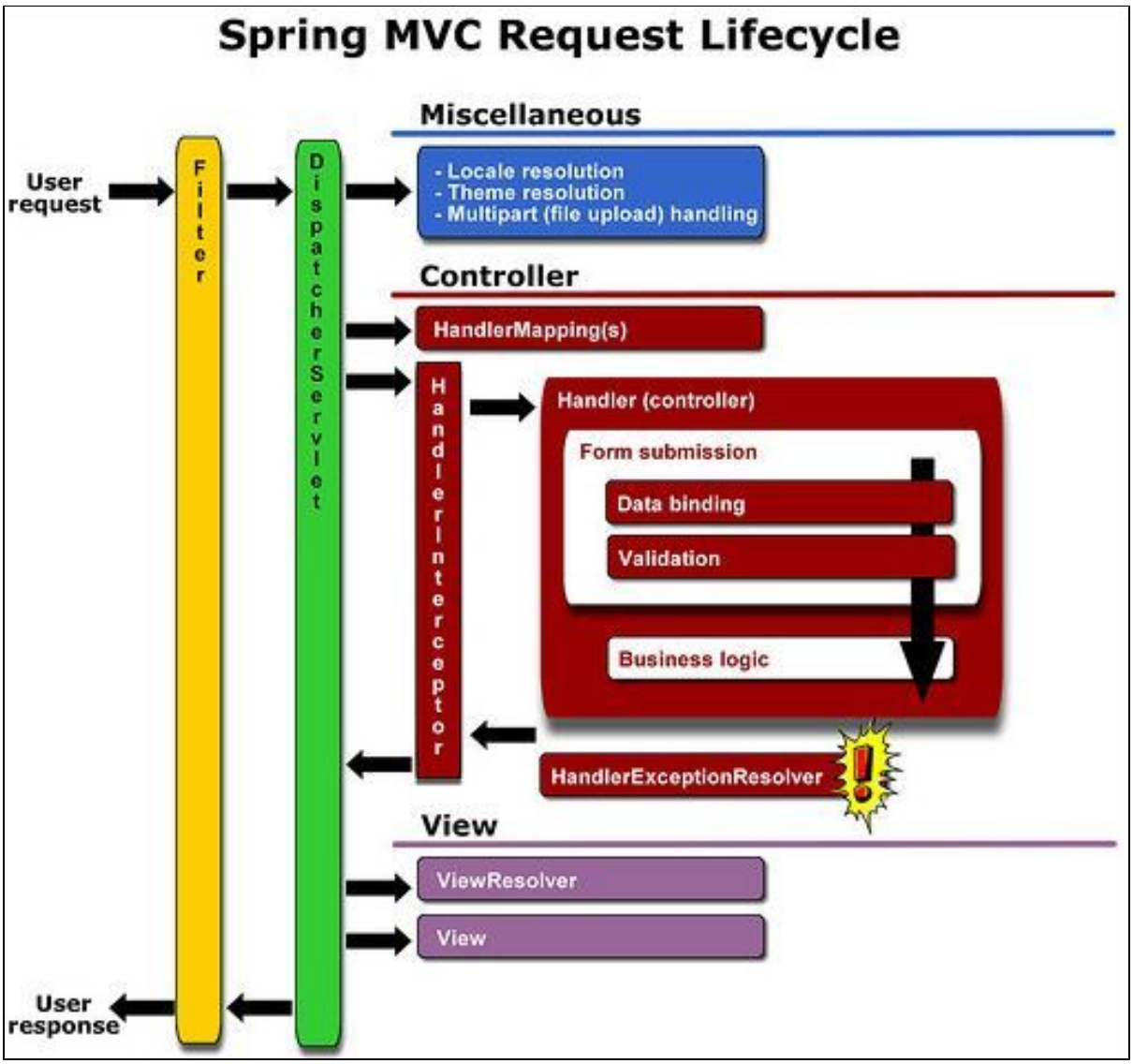
Interceptor는 각 구현체의 역할에 맞게 여러개를 등록 할 수 있으며 preHandle, postHandle의 호출 순서는 대개 각 Interceptor가 등록되는 순서이다(또는 Order 프로퍼티를 통해 조절 가능)

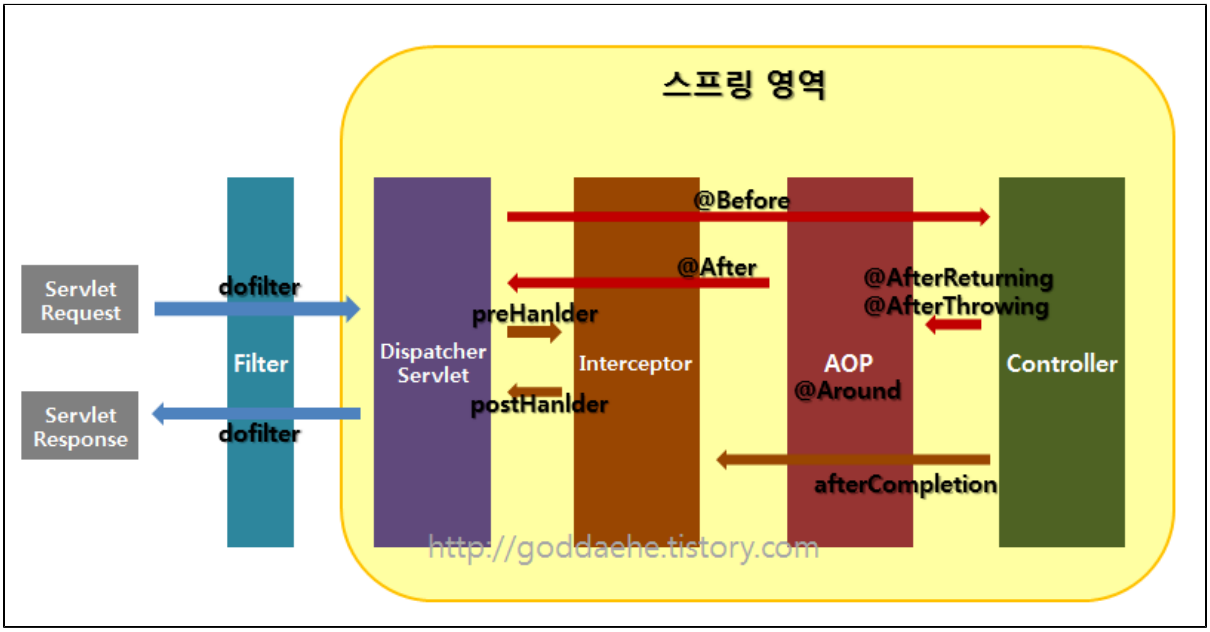
→ 순서에 의존하는 로직을 구현하지 않는 것을 권장한다

UserAuthCheckInterceptor를 개발한다고 가정할 때 우리가 구현해야 하는 메소드는?

→ 필요하지 않은 나머지 메소드들은 body가 비어있는 형태로 남아있게 된다

※ Filter, Interceptor, AOP의 차이점


Spring MVC Request Lifecycle
Miscellaneous
User request
Filter
DispatcherServlet
- Locale resolution
- Theme resolution
- Multipart (file upload) handling
Controller
HandlerMapping(s)
HandlerInterceptor
Handler (controller)
Form submission
Data binding
Validation
Business logic
HandlerExceptionResolver
View
ViewResolver
View
User response


스프링 영역
Servlet Request
dofilter
Filter
dofilter
Servlet Response
Dispatcher Servlet
preHanlder
postHanlder
Interceptor
@Before
@After
AOP
@Around
@AfterReturning
@AfterThrowing
Controller
afterCompletion
http://goddaehe.tistory.com

(3) DateTimeFormat을 통해 포매팅

1) @DateTimeFormat을 Handler에 선언하여 처리

```
@GetMapping("/date")
public ResponseEntity<String> date(@RequestParam("date")
                                   @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) Date date) {
    return ResponseEntity.ok(date.toString());
}

@GetMapping("/localdate")
public ResponseEntity<String> localDate(@RequestParam("localDate")
                                        @DateTimeFormat(iso = DateTimeFormat.ISO.DATE) LocalDate
    return ResponseEntity.ok(localDate.toString());
}

@GetMapping("/localdatetime")
public ResponseEntity<String> dateTime(@RequestParam("localDateTime")
                                       @DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME) LocalD
    return ResponseEntity.ok(localDateTime.toString());
}
```

2) @DateTimeFormat을 Dto에 선언하여 처리

```
@JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "dd.MM.yyyy")
private LocalDate date;
```

3) GlobalFormatter를 등록하여 처리

```java
@Configuration
@EnableWebMvc
@ComponentScan("io.crscube.semina.day11")
public class DispatcherServletConfig extends WebMvcConfigurerAdapter {
    @Bean
    public ObjectMapper objectMapper() { return new ObjectMapper(); }

    /**
     * {@inheritDoc}
     * <p>This implementation is empty.
     *
     * @param registry
     */
    @Override
    public void addFormatters(FormatterRegistry registry) {
        registry.addConverter(new Converter<String, LocalDate>() {
            /**
             * Convert the source object of type {@code S} to target type {@code T}.
             *
             * @param source the source object to convert, which must be an instance of {@code S}
             * @return the converted object, which must be an instance of {@code T} (potentially
             * @throws IllegalArgumentException if the source cannot be converted to the desired
             */
            @Override
            public LocalDate convert(String source) {
                return LocalDate.parse(source, DateTimeFormatter.ofPattern("dd.MM.yyyy"));
            }
        });
    }
}
```

4) Jackson Datebind의 JsonDeserializer를 통해 처리

```java
@JsonDeserialize(using = JsonLocalDateSerializer.class)
private LocalDate date;
```

```java
public class JsonLocalDateSerializer extends JsonDeserializer<LocalDate> {
    @Override
    public LocalDate deserialize(JsonParser jsonParser, DeserializationContext deserializationContext)
        return LocalDate.parse(jsonParser.getText(), DateTimeFormatter.ofPattern("dd.MM.yyyy"));
    }
}
```

2. Service Layer

(1) 기본 개념

실질적인 Business logic이 위치하는 곳이며 Controller에서 요청에 대한 기본적인 validation 후

Service layer에 처리를 위임하며 Service는 Persistence 영역 (Mybatis mapper, JPA Repository) 구현체를 주입받아 처리한다

Application의 전체적인 Domain model을 Request, Response로 사용할 수 있으나 권장하지 않는다

즉, Request, Response에 대한 DTO를 별도로 두고 Applicaiton에서 사용하는 핵심 오브젝트인 Domain model은 이와 별개의 Service layer에서 다루도록 한다

(2) Exception 처리

Application에서 발생하는 예외는 보통 Unchecked Exception일 것이다.

따라서 Service layer에서 발생하는 예외를 Controller에서 try-catch를 통해 잡아내는 것보다 일괄적으로 관리하는 것이 더 일반적이다.

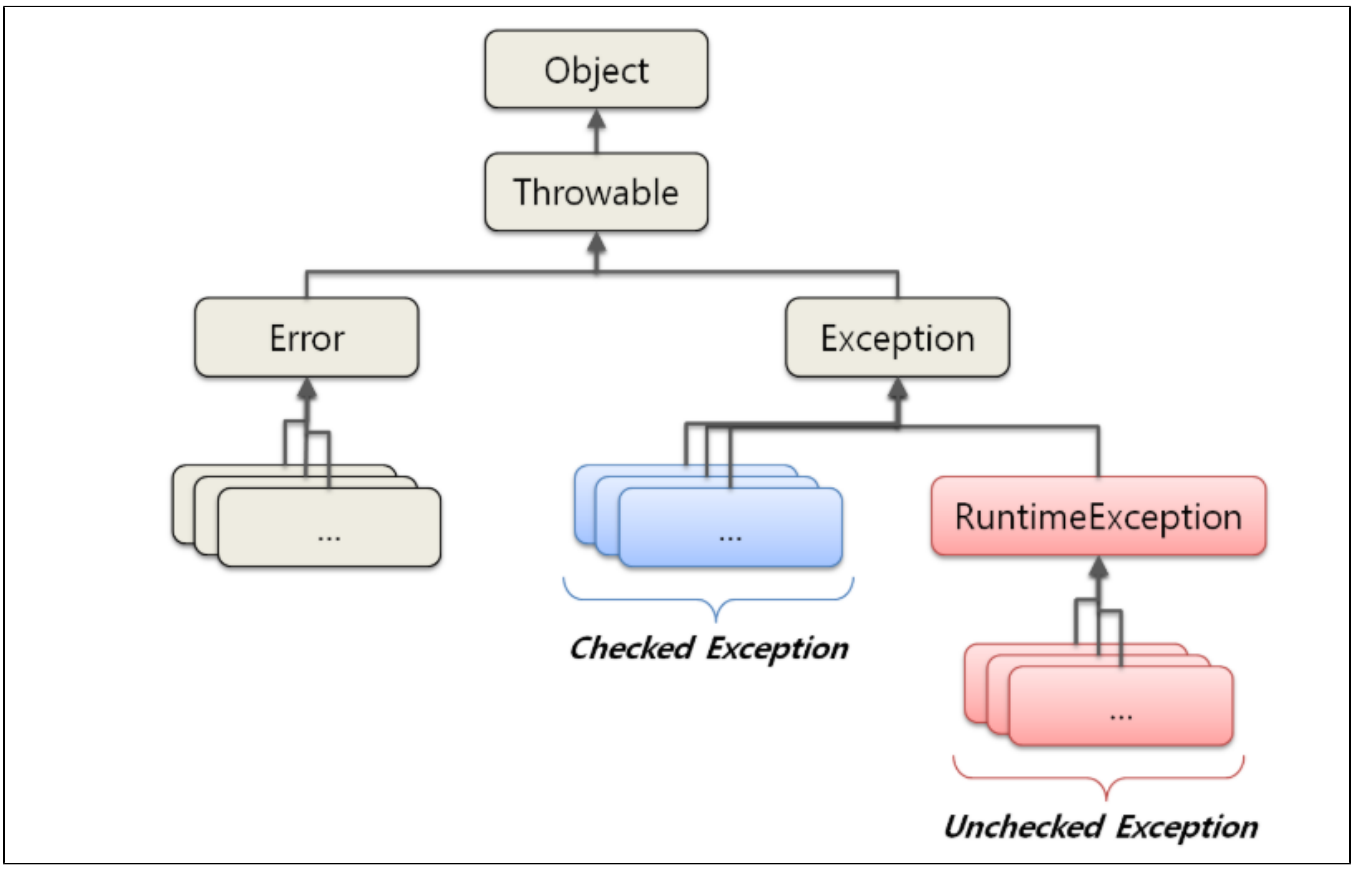
1) Spring의 ControllerAdvice와 ExceptionHandler

ControllerAdvice는 Controller 영역에서 발생한 모든 예외를 감지하는 AOP이며 ExceptionHandler는 발생한 예외의 종류별 handler이다.

ExceptionHandler를 아래와 같이 ControllerAdvice가 선언된 GlobalExceptionHandler에 선언할 수 있으며
개별 컨트롤러에도 선언하여 처리 내역을 세분화 할 수 있다.

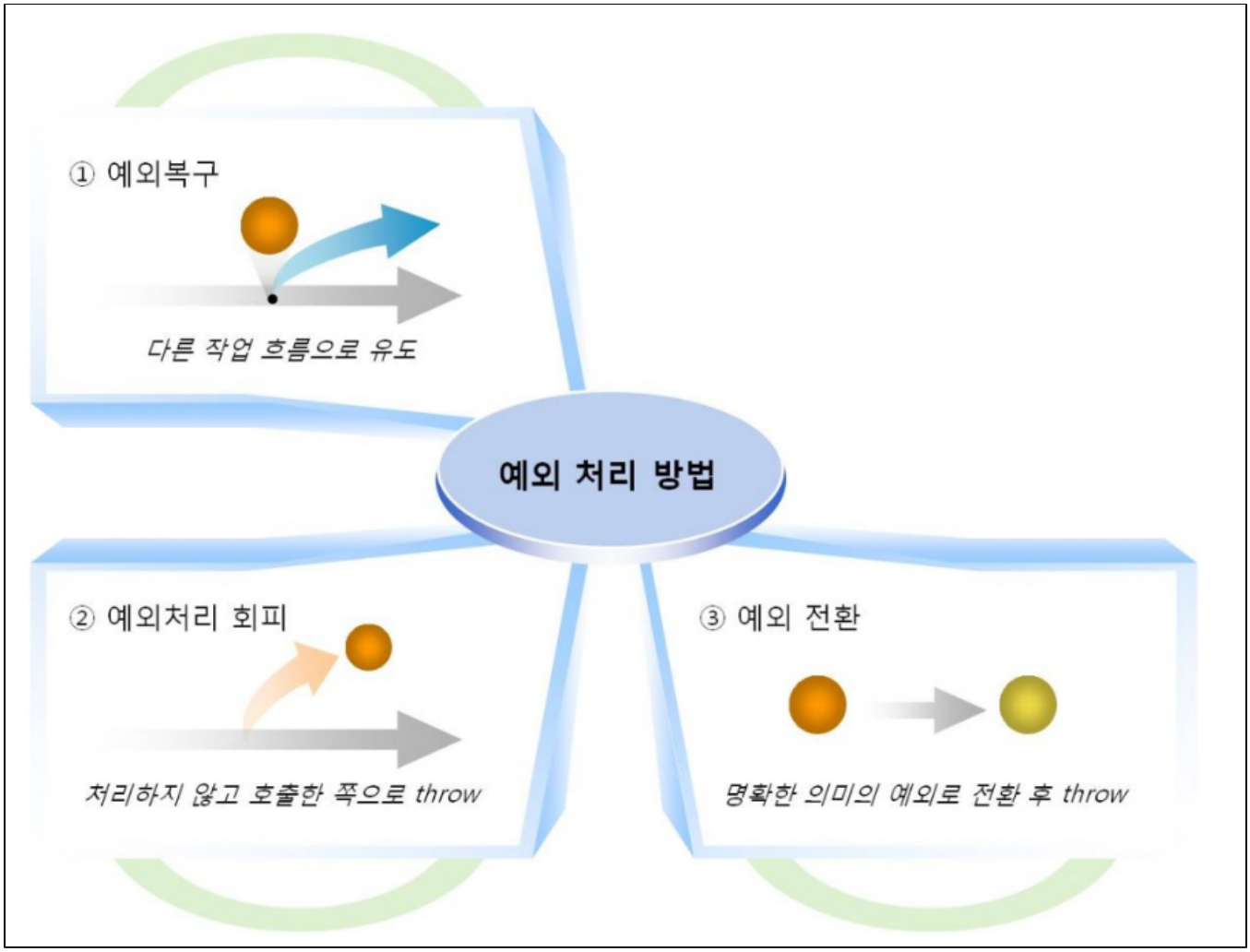
```
@ControllerAdvice
public class GlobalExceptionHandler {

	private static final Logger logger = LoggerFactory.getLogger(GlobalExceptionHandler.class);

	@ExceptionHandler(SQLException.class)
	public String handleSQLException(HttpServletRequest request, Exception ex){
		logger.info("SQLException Occured:: URL="+request.getRequestURL());
		return "database_error";
	}

	@ResponseStatus(value=HttpStatus.NOT_FOUND, reason="IOException occured")
	@ExceptionHandler(IOException.class)
	public void handleIOException(){
		logger.error("IOException handler executed");
		//returning 404 error code
	}
}
```

2) Checked Exception과 Unchecked Exception



항목	Checked Exception	Unchecked Exception
처리여부	반드시 처리해야 함	반드시 처리하지 않아도 됨
확인 시점	Compile time	Run time
예외 발생 시 트랜잭션	roll-back 되지 않음	roll-back 됨
종류	• IOException • SQLException	• NullPointerException • IllegalArgumentException • IllegalStateException • IndexOutOfBoundException • ConcurrentModificationException
특징	• 예측 가능한 • 복구 가능한 • 비즈니스 로직을 표현할 수 있음	• 예측 불가능한 • 복구 불가능한

예외의 처리 방법



메소드에서 발생 가능한 모든 예외는 문서화 하며 catch한 Exception은 무시하지 말 것 (처리하지 못할 예외는 catch 하지 말 것)

```
/**
 *
 * 5
 * 5
 *
 * {@link SystemService#getUserLoginFailCountInMinute}  {@code UsernameNotFoundException}
 *            .
 *
 * WARNING
 *
 * MON_CONNECT       .
 *
 *    5
 *    Account status Locked    .
 *
 * ex)
 * (1) 4 15 5    Account status Locked
 * (2) 4 18
 * (3) 4 19
 *     4 15 ~ 4 19  6    Account status Locked
 *
 * @param request   HttpServletRequest
 * @param exception
 */
private void handleAuthenticationFailure(HttpServletRequest request, AuthenticationException exception) {
    ConnectionLogHelper.handleAuthenticationFailureLog(request, getUserId(request, exception), this.
systemService);
    try {
        int userLoginFailCountInMinute = this.systemService.getUserLoginFailCountInMinute(getUserId(request,
exception), 5);
        if (userLoginFailCountInMinute >= 5) {
            this.systemService.lockUserAccountStatus(getUserId(request, exception));
        }
        request.setAttribute("userLoginFailCountInMinute", userLoginFailCountInMinute);
    } catch (UsernameNotFoundException e) {
        //Ignore exception
        //
    }
}
```