Q) URI 과 URL의 차이점은?

URL은 Uniform Resource Locator로 인터넷 상 자원의 위치를 의미합니다. 자원의 위치라는 것은 결국 어떤 파일의 위치를 의미합니다. 반면에 URI는 Uniform Resource Identifier로 인터넷 상의 자원을 식별하기 위한 문자열의 구성으로, URI는 URL을 포함하게 됩니다. 그러므로 URI가 보다 포괄적인 범위라고 할 수 있습니다.

- Rest API 특징 -

<https://jeong-pro.tistory.com/180> <- 이것이 핵심 restful은 rest 제약조건을 다 만족하는 것.....

1. Uniform Inteerface (일관된 인터페이스)

- URI만 통일되게 사용을 하면 특정 언어나 기술에 종속 받지 않는다.

- 클라이언트 플랫폼에 대해 무관하다.

- HTTP를 사용하는 플랫폼이면 다 사용가능하다.

- Loosely Coupling(느슨한 결함)형태를 가진다.

1. Stateless(무상태성)

- 서버는 각각의 요청을 개별로 인식해서 처리 해야한다.

- 이전 요청이 다음 요청과 연관 X

- 세션정보, 쿠키정보 저장 X

- 자유도 높음

- 서버에 불필요한 정보를 관리 X -> 구현이 단순

- 무상태성 - > 일관성 부여, 서버의 부담 줄임

1. Cacheable(캐시 가능)

- HTTP라는 기존의 웹표준을 그대로 사용하기 때문에, 웹의 기존 인프라를 그대로 활용 -> 캐시 기능 적용 가능

- HTTP 프로토콜 표준에서 사용하는 Last-Modified Tag OR E-Tag를 이용하여 캐싱을 구현가능 -> 대량의 요청을 효율적으로 처리

1. Client-Server Architecture(서버-클라이언트 구조)

- 서버는 API를 제공하여, 클라이언트는 사용자 인증, Context(세션, 로그인 정보)등을 직접 관리하는 등 역할을 확실히 구분시켜서 서로 간의 의존성을 줄임.

1. Self-Descriptiveness(자체 표현)

- Rest API는 구조 자체가 쉽게 이해할 수 있는 자체 표현 구조

1. Layered System(계층 구조)

- Rest API의 서버는 다중 계층으로 구성될 수 있으며 보안, 로드 밸런싱, 암호화 등을 위한 계층을 추가하여 구조를 변경할 수 있다.

- Proxy, gateway와 같은 네트워크 기반의 중간매체를 사용할 수 있게 해줍니다.

- 단. 클라이언트는 서버와 직접 통신하는지 중간매체와 통신하는지 알 수 없다.

Spring이란?

- 자바 플랫폼을 위한 오픈소스 애플리케이션 프레임워크

- 프로젝트를 분할하여 개발을 하기 위한 뼈대

프레임워크?

- 복잡한 문제를 해결하거나 서술하는데 사용되는 기본 개념구조

\* 프레임워크 VS Library

1. Flow(흐름)에 대한 제어 권한이 어디에 있는냐?

- 프레임워크는 전체적인 흐름을 자체적으로 가지고 있다. 그러나 라이버러리는 사용자가 흐름에 대해 제어가 필요하다.

Maven 프레임워크의 최대 장점 - Pom.xml을 통한 라이브러리 관리.

Maven [Pom.xml]

- 프로젝트 의존성 관리, 라이브러리 관리, 프로젝트 생명주기 관리 기능 등을 제공하는 프로젝트 관리 도구.

- 모든 pom.xml은 최상위 pom.xml을 참조

- 메이븐에서 기본적으로 제공하고 있는 것이며 개발자가 만드는 모든 pom.xml은 이를 상속 받아서 사용

▶ 태그

1. <build>

- 메이븐의 기본 디렉토리 구조가 정의

1. < version>

- 메이븐에서 버전에 대한 명명 규칙을 SNAPSHOT 과 RELEASE로 구분한다.

- SNAPSHOT은 완성되지 않은 버전, RELEASE는 완성되어서 공식 배포된 버전.

1. <properties>

- pom.xml파일 내부에서 사용할 속성을 정의한다. 주로 버전 정보 같이 공통적으로 많이 사용하는 내용을 속성으로 정의한다.

1. <scm>

- 버전 관리 시스템과 연계하기 위한 정보를 기술

1. <repositories>

- 메이븐 저장소 정보를 기술. 주로 사내 사설 저장소를 사용할 때 사용

1. <distributionManagement>

- 빌드의 결과물을 배포할 위치를 지정. 주로 사내 사설 저장소의 정보를 사용

1. <profiles>

- 특정 항목 혹은 특정 속성에 따라 빌드를 다르게 수행

1. <dependency>

- 라이브러리 정보를 기술하는 태그.

- 태그 안에 몇가지 정보를 기술하고 라이브러리 지정한다.

1. <scope>

- 라이브러리가 이용되는 범위를 지정하는것.

- “어떤 때 사용하는지”를 나타내는 것

Servlet(서블릿)

- 웹프로그래밍에서 클라이언트의 요청을 처리하고, 그 결과를 다시 클라이언트에게 전송하는 Servlet 클래스의 구현 규칙을 지킨 자바 프로그래밍 기술

- 특징

1. 클라이언트의 요청에 대해 동적으로 작동하는 웹 어플리케이션 컴포넌트(하나의 모듈같은 한개의 단위 같은 개념)
2. html을 사용하여 요청에 응답
3. Java Thread를 이용하여 동작
4. MVC패턴에서 Controller로 이용
5. HTTP 프로토콜 서비스를 지원하는 javax.ervlet.http.HttpServlet클래스를 상속 UDP보다 속도가 느리다.
6. HTML변경시 Servlet을 재컴파일해야 하는 단점

Dispatcher-Servlet

- Servlet Container에서 HTTP프로토콜을 통해 들어오는 모든 요청을 프레젠테이션 계층의 제일 앞에 둬서 중앙집중식으로 처리해주는 프론트 컽르롤러(Front Controller)

- 클라이언트 요청 -> Tomcat과 같은 Servlet 컨테이너 요청 받음 -> 제일 앞에서 서버로 들어오는 모든 요청을 처리하는 프론트 컨트롤러를 정의 한것이 Dispatcher-Servlet -> Dispatcher-Servlet에서 공통처리 작업을 처리 -> 적절한 세부 컨트롤러로 작업을 위임

\* Dispatcher-Servlet이 처리하는 URL 패턴을 지정해주어야 하는데 일반적으로는 /\*.do와 같은 /로 시작하며 .do로 끝나는 URL패턴에 대해서 처리하라고 지정

Front Controller

- 서블릿 컨테이너의 제일 앞에서 서버로 들어오는 클라이언트의 모든 요청을 받아서 처리해주는 컨트롤러, MVC 구조에서 함께 사용되는 패턴.

**Spring - JSP파일 구동, web.xml, servlet-context.xml**

필수 요소 : jsp, web.xml, servlet-context.xml

구동 flow

- 브라우저 url 요청 -> web.xml의 설정된 Dispatcher Servlet으로 들어옴 ->

Servlet에 <param-value>에 설정된 servlet-context.xml로 이동 ->

servlet-context.xml에 <context:component-scanbase-package=*"--------"* /> 를 통해서 java코드의 controller에 mapping되는 url을 찾는다

-> Controller에서 return시 ViewResolver에 넘겨주는대 이때 servlet-context.xml에 설정되어있는 jsp파일을 찾는다.

<property name=*"prefix"* value=*"/WEB-INF/jsp/"*></property>

<property name=*"suffix"* value=*".jsp"*></property>

-> ViewResolver가 -> View에 넘겨준다.

-> View는 Dispatcher Servlet에 넘겨주고 이를 다시 -> 클라이언트(브라우저)에게 넘겨준다.

**jqxGrid 사용법**

1. jsp파일

- grid가 들어갈 위치에 <div> 태그 추가 + div 태그를 식별할 id도 같이 입력

Ex) <!-- grid -->

<div class=*"grid-group"*> //<- 원하는 css용 class

<div id=*"gridPolicyDetail"*></div>

</div>

<!-- / grid -->

단) id를 찾기 위해서는 js파일에 연동이 필요하다

<script src=*"*<c:url value=*"/resources/app/0/1/2.js"* />*"*></script>

1. Js 파일

- jsp파일과 연결되어 있다는 가정하에 jquery를 통해서 #(아이디셀렉터)을 이용을 통해 id를 변수에 입력을 한다. ex) var gird = gridPolicyDetail;

- grid테이블의 필드 명을 (key,value)값으로 변수에 저장한다. 향후 grid의 컬럼에 사용이된다.

Ex) var arrDefViewFlds = {

"이름" : "홍길동",

"주소" : "서울시....",

},

- drawGrid 함수에 Form을 입력해준다.

1. **gridSource (지금 형태는 url을 통해서 DB의 데이터를 가지고와서 입력해주는 방식)**

**var** gridSource = {

datatype: "json",

datafields: [ //DB에서 가저온 데이터 컬럼명과 일치하게 해준다.

{ name: "name", type: "string"},

{ name: "addr", type: "string"}

],

root: 'rows',

beforeprocessing: **function**(data){

**if** (data != **null**){

gridSource.totalrecords = data.totalRows;

}

**for**( **var** i=0; i<data.rows.length; ++i){

data.rows[i].bOpend = **false**;

}

},

cache: **false**,

url: urlList //설정해 놓은 URL을 넣는다. <http://fsdafdsa.json>

},

1. Dataadapter

dataadapter = **new** $.jqx.dataAdapter(gridSource, {

formatData : **function**(data) {

data.s\_eqp\_ip = m$.dialog.find("[name=s\_eqp\_ip]").val(); // jsp파일에 입력된 값이 있으면 그걸 통해서 처리를 하기 위해서 사용

**var** params = {}, param, flds = $(mCfg.formId).serializeArray();

**if**(params.s\_eqp\_ip=='undefined' || params.s\_eqp\_ip=='') params.s\_eqp\_ip = '';

**else** params.s\_eqp\_ip = m$.dialog.find("[name=s\_eqp\_ip]").val();

**for**(param **in** flds) {

params[flds[param].name] = flds[param].value;

};

$.extend(data, params);

**return** data;

},

loadError: **function**(xhr, status, error){

alert(error);

}

});

1. $grid.jqxGrid

$grid.jqxGrid({

source: dataadapter,

width: '99%',

pagesize: ctrlCookie.getValue(),

virtualmode: **true**,

enablehover:**false**,

autorowheight: **true**,

autoheight: **true**,

enablebrowserselection: **true**,

rendergridrows: **function**(obj){

**return** obj.data;

},

columns: GridManager.getColumns(), //<- 원하는 컬럼을 만들어 넣어준다.

rendered : **function**() {

bindEvent();

}

});

//////////////GridManager

GridManager = {

//여기 부터는 각 컬럼에 들어가는 data들을 CSS를 설정 및 html설정

cellsrenderer : **function**(row, columnfield, value, defaulthtml, columnproperties, rowdata) {

**var** resultHtml = "<div class='jqx-grid-cell-middle-align' style='margin-top:8px;' />",

valToArr = value.split("@"),

showAll = **false**,

cellBgColor = '#fff';

**if**(row % 2 === 1) {

cellBgColor = '#f8f8f8';

}

**if**(currentShowCell && currentShowCell.row == row) {

**for**(**var** i **in** valToArr) {

resultHtml += "<p>" + valToArr[i] + "</p>";

}

**if**(currentShowCell.columnfield == columnfield) {

resultHtml += "<br><span class='icon-chevron-up btn-show-cell' row='" + row + "' columnfield='" + columnfield + "' style='font-size:8px;position:absolute;bottom:3px;padding:3px;cursor:pointer;background-color:"+cellBgColor+";' />";

}

}

**else** {

**for**(**var** i **in** valToArr) {

**if**(i == maxCellRow) {

resultHtml += "<br><span class='icon-chevron-down btn-show-cell' row='" + row + "' columnfield='" + columnfield + "' style='font-size:8px;position:absolute;bottom:3px;padding:3px;cursor:pointer;background-color:"+cellBgColor+";' />";

**break**;

}

resultHtml += "<p>" + valToArr[i] + "</p>";

}

}

setTimeout(**function**(){ GridManager.bindBtnShowCell(); }, 500);

**return** resultHtml;

},

cellsrendererNumber : **function**(row, columnfield, value, defaulthtml, columnproperties, rowdata) {

**var** resultHtml = "<div class='jqx-grid-cell-middle-align' style='margin-top:8px;' />";

resultHtml += "<p>" + (value + 1) + "</p>";

**return** resultHtml;

},

cellsrendererDefault : **function**(row, columnfield, value, defaulthtml, columnproperties, rowdata) {

**var** resultHtml = "<div class='jqx-grid-cell-middle-align' style='margin-top:8px;' />";

resultHtml += "<p>" + (value) + "</p>";

**return** resultHtml;

},

bindBtnShowCell : **function**(){

$(btnShowCell).off().on('click',**function**(){

**var** btn = $(**this**);

**if**(currentShowCell && currentShowCell.row === btn.attr('row')) {

currentShowCell = **null**;

}

**else** {

currentShowCell = {

row : btn.attr('row'),

columnfield : btn.attr('columnfield')

};

}

$grid.jqxGrid('render', 'cells');

});

},

// 이부분이 데이터를 가져와서 테이블에 입력할 수 있게 해준부분.

getColumns : **function**(){

**var** columns = []; //표시필드 jqxGrid적용컬럼

columns.push({text: 'No', columntype: 'number', width:40, cellsalign:'center', cellsrenderer:GridManager.cellsrendererNumber});

//$.each는 자체적으로 for문처럼 만들어놓은 함수개념. 위에 입력한 필드 별로 key와 value를 통해서 테이블 컬럼에 넣어준다.

$.each(arrViewFlds, **function**(key, val) {

**switch**(key){

**case** "name":

columns.push({ text: val, datafield: key, cellsalign:'center', cellsrenderer:GridManager.cellsrendererDefault });

**break**;

**case** "addr" :

columns.push({ text: val, datafield: key, cellsalign:'center', cellsrenderer:GridManager.cellsrendererDefault });

//위에 적용된 CSS를 적용시키겠다는 말이다.

**break**;

}

});

**return** columns;

},

};

---------------------------------------------------------------------

Docker (<http://pyrasis.com/docker.html>)

\* 반가상화보다 좀더 경량화된 방식

\* OS를 설치 X

\* Docker 이미지에 서버 운영을 위한 프로그램과 라이브러리만 격리 설치 가능

\* OS 자원(시스템콜)은 호스트와 공유한다.

\* 이미지 용량이 크게 줄었다.

\* Docker는 하드웨어를 가상화하는 계층이 없기 때문에 메모리 접근, 파일시스템, 네트워크 속도가 가상 머신에 비해 월등히 빠르다.

Docker이미지

\* 베이스 이미지 - 리눅스 배포판의 유저랜드만 설치된 파일 OR 리눅스 배포판 유저랜드에 Redis나 Nginx 등이 설치된 베이스 이미지도 있다.

\* 결론 - Docker 이미지라고 하면 베이스 이미지 + 필요한 프로그램과 라이브러리, 소스를 설치한 뒤 파일 하나로 만든 것

\* Docker는 이미지를 통째로 생성하지 않고, 바뀐 부분만 생성한 뒤 부모 이미지를 계속 참조하는 방식으로 동작.

\* 장점 : 내가 원하는 라이브러리, 실행파일, 스크립트를 docker에 묶어서 이미지를 만들어서 올려놓으면 매번 서버를 개설할때 처음부터 다 설치할 필요없이 docker이미지 다운만으로도 서버를 개설 할 수 있다는 장점. 즉 많은 서버를 개설할 때 아주 유용하다.

D3.js(<https://www.tutorialspoint.com/d3js/d3js_selections.htm>)

**Selections**

- select()

- selectAll() -> 전체 선택

Ex) d3.select(“.myclass”) OR d3.select(“#myid”) ...

뒤에 붙이는 것들

- append()

- text()

Ex) d3.select(“div.myclass”).append(“span”)

d3.select(“div.myclass”).text(“from D3.js”)

- attr() -> 속성 부여

- html() -> HTML화해준다

- style() -> 스타일적용

d3.select(".myclass").attr("style", "color: red");

d3.select(".myclass").style("color", "red");

Add class

- classed()

d3.select(".myclass").classed("myanotherclass", true);

Delete class

d3.select(".myclass").classed("myanotherclass", false);

클래스가 있는지 없는지 확인하고 추가 해주는 토글 방식

var element = d3.select(".myclass")

element.classed("myanotherclass", !oneBar.classed("myanotherclass"));

**Data join**

현 html 상태

<ul id = "list">

<li><li>

<li></li>

</ul>

.js

d3.select("#list").selectAll("li")

.data([10, 20, 30, 25, 15])

.text(function(d) { return "This is pre-existing element and the value is " + d; })

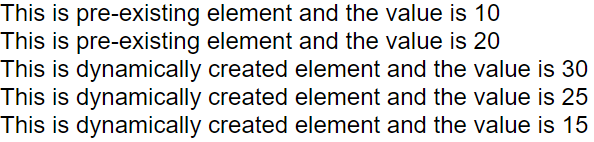
.enter()

.append("li")

.text(function(d)

{ return "This is dynamically created element and the value is " + d; });

출력 화면



\* 설명 10,20은 html에 li가 두개가 있는대 이거에 매칭되게 data가 들어간 것이고 나머지는 새로 생성을 한것이다.

data에서 remove

d3.selectAll("li")

.data([10, 20, 30, 15])

.exit()

.remove()

이러면 4번째 숫자인25가 사라지게 된다.

datum() -> single element를 설정 할 수 있게 해준다.

d3.select("p")

.datum(50)

.text(function(d) {

return "Used existing paragraph element and the data " + d + " is assigned.";

});

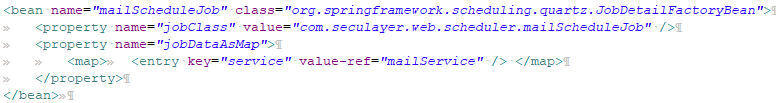
-> Used existing paragraph element and the data 50 is assigned.

enter() -> element가 없다면 만들어주는 역할을 하는 부분 직접 만들어주는건 .append()지만 enter()를 통해서 만들어주고자 하는걸 있는지 없는지 확인한다는 느낌??

**스프링 quartz를 이용한 스케쥴링**

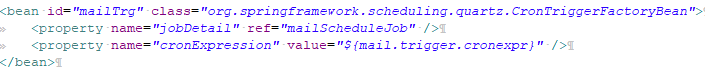
1. Servlet--context.xml <bean> 설정

- bean 안에 property 그리고 serive로 실행을 할거기 때문에 service도 설정

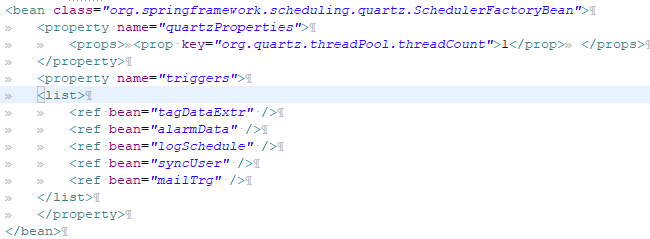


- cron을 어떤 방식으로 몇분 몇시 이런식으로 실행을 할 파일을 가지고 있어야 한다.

- 즉 그 파일과 연동해서 스케쥴러가 시간때에 돌수 있게 성정을 하는 부분



- 실행하기 위한 설정



여기까지 servlet-context.xml설정 이후 cron 파일과 위에 선언해 놓은 클래스들 서비스 클래스 포함 작성을 해주어야 한다.

여기서는 mailScheduleJob.java, mailService.java, cron-trigger.properties를 만들어주어야 한다.

mailScheduleJob.java



mailService.java

- 이 파일은 위에 ((mailService)svc).mailProcess();를 실행 시켜줄수 있는 파일을 하나 만들어주면 된다. 단순히 mailService클래스 안에 public void mailProcess(){ ..... }가 끝일 수도 있다.

\* 여기까지 완료를 한다면 설정해 놓은 빈을 통해서 스프링이 자체적으로 시간을 가지고 자동 실행을 하게 된다.

% Cron 파일 시간 설정 관련 <https://aljjabaegi.tistory.com/400> 참고

1. Spring MVC 구조의 처리 과정을 설명해보시오.

<https://jeong-pro.tistory.com/96?category=793347>

간략화...

DispatcherServlet이 url catch를 한다. -> Controller에게(@RequestMapping)을 통해서 url을 맵핑한다. -> 컨트롤러는 해당 요청의 service를 주입(DI)받아 비즈니스로직을 Service에게 위임. -> DB 작업을 할게 있으면 Mybatis 기준 DAO를 통해서 쿼리 정보를 받아온 후 따로 클래스를 만들어서 즉 VO(dto)클래스로 받을것인지 아니면 map, int, String... 기본 변수로 받을 것인지 resultType을 선언하면 된다. -> 서비스가 완료가 되면 즉 로직이 완료가 되면 컨트롤에게 넘긴다 -> 컨트롤은 Model 객체에 어떤 jsp 파일을 보여줄 것인지 DispatcherServlet에게 보낸다. -> DispatcherServlet은 ViewResolver에게 View의 정보를 넘기고 -> ViewResolver는 .jsp파일을 찾아서 DispatcherServelt에게 알려준후 -> DispatcherServlet은 클라이언트에게 랜더링된 View를 넘긴다.

1. 스택, 큐, 덱의 특징에 대해 설명.

<https://jeong-pro.tistory.com/97?category=793347>

1. 디자인 패턴

<https://jeong-pro.tistory.com/98?category=793347>

1. 제네릭

<https://jeong-pro.tistory.com/100?category=793347>

1. DB index

<https://jeong-pro.tistory.com/114?category=793347>

**HTTP 프로토콜**

* 브라우저와 서버간에 데이터를 주고받기 위한 방식
* 상태가 없는(stateless)프로토콜이다. > 데이터를 주고 받기 위한 각각의 데이터 요청이 서로 독립적으로 관리된다는

**-상태 코드-**

### 2xx - 성공

200번대의 상태 코드는 대부분 성공을 의미합니다.

* 200 : GET 요청에 대한 성공
* 204 : No Content. 성공했으나 응답 본문에 데이터가 없음
* 205 : Reset Content. 성공했으나 클라이언트의 화면을 새로 고침하도록 권고
* 206 : Partial Conent. 성공했으나 일부 범위의 데이터만 반환

### 3xx - 리다이렉션

300번대의 상태 코드는 대부분 클라이언트가 이전 주소로 데이터를 요청하여 서버에서 새 URL로 리다이렉트를 유도하는 경우입니다.

* 301 : Moved Permanently, 요청한 자원이 새 URL에 존재
* 303 : See Other, 요청한 자원이 임시 주소에 존재
* 304 : Not Modified, 요청한 자원이 변경되지 않았으므로 클라이언트에서 캐싱된 자원을 사용하도록 권고. [ETag](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag)와 같은 정보를 활용하여 변경 여부를 확인

### 4xx - 클라이언트 에러

400번대 상태 코드는 대부분 클라이언트의 코드가 잘못된 경우입니다. 유효하지 않은 자원을 요청했거나 요청이나 권한이 잘못된 경우 발생합니다. 가장 익숙한 상태 코드는 404 코드입니다. 요청한 자원이 서버에 없다는 의미죠.

* 400 : Bad Request, 잘못된 요청
* 401 : Unauthorized, 권한 없이 요청. Authorization 헤더가 잘못된 경우
* 403 : Forbidden, 서버에서 해당 자원에 대해 접근 금지
* 405 : Method Not Allowed, 허용되지 않은 요청 메서드
* 409 : Conflict, 최신 자원이 아닌데 업데이트하는 경우. ex) 파일 업로드 시 버전 충돌

### 5xx - 서버 에러

500번대 상태 코드는 서버 쪽에서 오류가 난 경우입니다.

* 501 : Not Implemented, 요청한 동작에 대해 서버가 수행할 수 없는 경우
* 503 : Service Unavailable, 서버가 과부하 또는 유지 보수로 내려간 경우