UrlMappings

Url Mappings

Table of Contents

현재까지는 URL이 grails Convention 에 따라서 결정되었지만,
grails-app/conf/UrlMappings.groovy 에서 제어할 수 있다.

UrlMappings 클래스는 코드의 블럭을 할당할 수 있는 mappings이라는 프로퍼티 하나만 가진다

Mapping to Controllers and Actions

url에 컨트롤러, 액션을 매핑할 수 있다

UrlMappings.groovy에 아래 내용을 추가한다. (mapping 프로퍼티 안에 추가)

```
1   "/myBook"(controller:"book", action:"view")
```

확인주소 : http://localhost:8080/mygrails/myBook/
myBook 이라는 url은 BookController의 view Action으로 연결된다.
action을 생략할 경우 BookController의 기본 Action으로 연결된다.

실행 후 아래와 같이 확인할 수 있다.

/book/view로 접근했었던 내용이 /myBook 으로 접근 가능하게 되었다

메소드에 넘기는 블럭 안에 컨트롤러와 액션을 명시하는 방법도 있다

UrlMappings.groovy에 추가했었던

```
1  "/myBook"(controller:"book", action:"view")
```

내용을 지우고 아래 내용을 추가한다

```
1  "/myBook" {
2                  controller = "book"
3                  action = "view"
4                  }
```

확인 해보면 내용은 같다.

어떤 방법을 사용하든 같다.

Embedded Variables

Simple Variables(간단한 변수)

고정된 문자열로 url을 구성을 하였으나, 변수를 사용할 수도 있다.

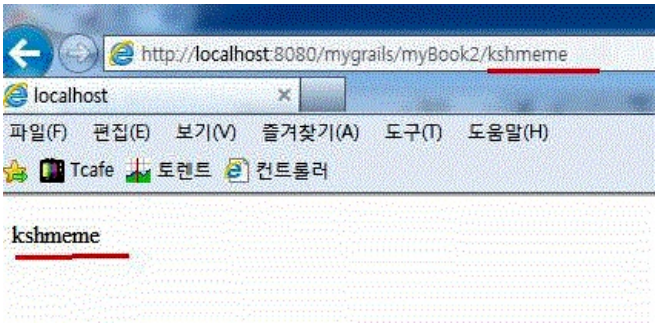
UrlMappings.groovy 에 아래의 내용을 추가한다.

```
1  "/myBook2/$id"(controller : "book", action : "view2")
```

BookController 에 아래의 내용을 추가한다.

```
def view2 = {
      render params.id
  }
```

/myBook2/id 형태의 url로 접근하면 화면에 id가 렌더링 되는 걸 볼 수 있다.
/myBook2/ 형태의 url은 404이다.
확인주소 : http://localhost:8080/mygrails/myBook2/kshmeme



좀더 복잡한 매핑도 정의할 수 있다
확인주소 : http://localhost:8080/mygrails/book/2014/04/02/1

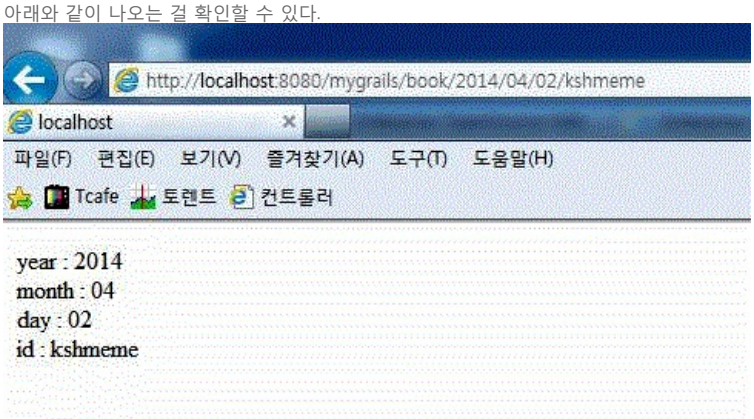
UrlMappings.groovy 에 아래의 내용을 추가한다.

```
"/book/$year/$month/$dav/$id"(controller:"book", action:"view3")
```

BookController 에 아래의 내용을 추가한다.

```
def view3 = {

      def year = params.year
      def month = params.month
      def day = params.day
      def id = params.id

      def text = "year : " + year + "<br />"
      text = text+ "month : " + month + "<br />"
      text = text+ "day : " + day + "<br />"
      text = text+ "id : " + id + "<br />"

     render text
  }
```

아래와 같이 나오는 걸 확인할 수 있다.



Dynamic Controller and Action Names(동적 컨트롤러와 액션 이름)

변수는 동적으로 컨트롤러와 액션의 이름을 매핑 하는데에서 사용 가능하다.
아래는 Grails의 기본적으로 설정되어있는 매핑 규칙이다.
컨트롤러이름/액션이름/아이디 의 주소 규칙을 가지고 있다.

```
"/$controller/$action?/$id?"{
            constraints {
                // apply constraints here
            }
        }
```

Optional Variables(필수가 아닌 변수)

기본적으로 설정되어있는 매핑 규칙은
토큰(/ / 사이에 있는 문자열) 끝에 ? 를 붙여 생략이 가능 하도록 했다

기본 매핑은 토큰 끝에 ? 를 붙여 생략이 가능 하도록 했다

UrlMappings.groovy 에 추가했었던
"/book/$year/$month/$day/$id"(controller:"book", action:"view3") 의 내용을 아래와 같이 변경한다.

```
"/book/$vear/$month/$dav?/$id?"(controller:"book", action:"view3")
```

day와 id 는 생략 가능하여 연/월 만 받을 수도있다

아래와 같이 여러 가지 경우에 처리를 할 수 있다.
http://localhost:8080/mygrails/book/2014/04/
http://localhost:8080/mygrails/book/2014/04/01
http://localhost:8080/mygrails/book/2014/04/01/kshmeme

Arbitrary Variables(임의 변수)

URL이 매핑될 때 임의의 파라미터를 넘기도록 할 수 있다
매핑시 넘겨지는 블럭에 임의의 파라미터들을 설정하면 된다.

UrlMappings.groovy에 아래와 같이 추가하면 paramtest Action에서 params.id 또는 params.year에 접근 할 수 있다.

```
"/book/paramtest"{
    controller = "book"
    action = "paramtest"
    id="kshmeme"
    year="2014"
}
```

Dynamically Resolved Variables(동적으로 변수 이름 결정하기)

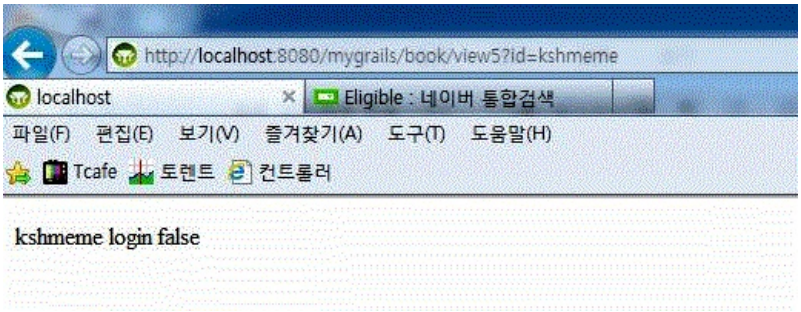
변수의 이름을 런타임에 결정해야 할 때도 있다.
변수이름에 블럭을 할당함으로써 가능하다.
UrlMappings.groovy 에 아래의 내용을 추가한다.

```
"/book/view5"{
            controller = "book"
            action = "view5"
            userid = {params.id}
            isLogin = { session.user != null } // must be logged in
        }
```

BookController 에 아래의 내용을 추가한다.

```
def view5 = {
      params.userid

      render params.userid  + " login " + params.isLogin

   }
```

아래와 같이 확인할 수 있다



Mapping to Views

view를 컨트롤러나 액션 없이 연결할 수 있다.
UrlMappings.groovy 에 기본적으로 설정되어있다

```
static mappings = {
     "/"(view:"/index")  // // 루트 URL이 매핑된다.
}
```

Mapping to Response Codes

HTTP응답 코드에 따라 컨트롤러, 액션, 뷰에 매핑할 수 있다.
UrlMappings.groovy 에 기본적으로 설정되어있다.
(경우에 따라서는 404, 403 등 코드별로 각각 설정할 수 있다)

```
"500"(view:'/error')
```

Mapping to HTTP methods
HTTP 메소드(GET, POST, PUT, DELETE)에 따라 URL이 다르게 매핑되도록 설정할 수 있다
RESTful 같은 방식에 유용하게 사용할 수 있다.

확인주소 : http://localhost:8080/mygrails/book/methodTest
UrlMappings.groovy에 아래 내용을 추가한다

```
"/book/view6"{
            controller = "book"
            action= [
                GET:"view6_1", PUT:"view6_2", DELETE:"view6_3", POST:"view6_4"
            ]
        }
```

BookController에 아래 내용을 추가한다.

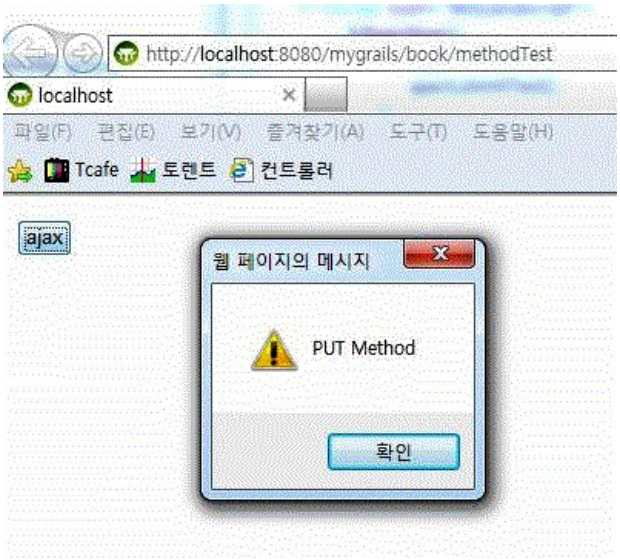
```
def view6_1 = {
```

```
        render "GET Method"
    }

    def view6_2 = {
        render "PUT Method"
    }

    def view6_3 = {
        render "DELETE Method"
    }

    def view6_4 = {
        render "POST Method"
    }


    def methodTest = {}
```

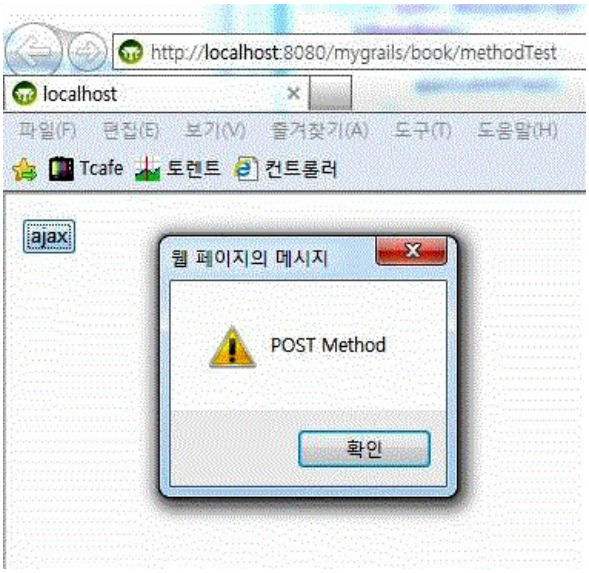
views/book 디렉터리에 methodTest.gsp를 추가한다.
methodTest.gsp의 내용을 아래와 같이 변경한다.

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
<script type="text/javascript">
    $(function(){
        $("#ajaxTest").click(function(){
            ajaxSubmitTest();
        });
    })
    function ajaxSubmitTest(){
        $.ajax({
            url: '/mygrails/book/view6',
            type: 'PUT',
            data: {
                    // _method: 'PUT'
            },
            success : function(data){
                    alert(data);
        }
    });
    }
</script>
  <div class="body">
    <input type="button" value="ajax" id="ajaxTest" />
  </div>
```

실행후 ajax 버튼을 클릭하면 javascript에서 ajax 호출시
type 에 따른 Action이 실행되는 것을 확인할 수 있다.(현재는 PUT)



type을 POST로 변경 후 실행 하면 아래처럼 POST으로 연결된 Action(view6_4)이 실행된다


http://localhost:8080/mygrails/book/methodTest
localhost
파일(F) 편집(E) 보기(V) 즐겨찾기(A) 도구(T) 도움말(H)
Tcafe 토렌트 컨트롤러
ajax
웹 페이지의 메시지
POST Method
확인