02.19 목

RAP
멘토링 2회차

FIORI 문법 복습

멘토 김예원

• INDEX

목차

01 Controller 안에서의 this 정리

⇒ 현재 View를 제어하고 있는 컨트롤러 자기 자신

```
this.getView().byId("idInput")
```

→ 내(컨트롤러)가 관리하는 현재 화면(View) 가져오기.
특정 화면의 컨트롤 접근할 때 사용.

→ View 접근은 this.getView() 통해서 가져오기.

01 success 함수 안에서의 this

→ 바로 실행되는 함수가 아니라 나중에 서버에서 데이터가 오면 그 때 실행되는 함수
그래서 이 함수가 실행될 때 this가 뭐였는지 잊어버림

1) 미리 this를 복사해두기

```
let that = this;
success: function() {
    that.getView();
}
```

2) success 실행 시 this를 강제로 현재 컨트롤러에 묶음

```
success: function() {
    this.getView();
}.bind(this)
```

01 getView() vs getOwnerComponent()

Component → 아파트 단지 관리소 (앱 전체를 관리하는 존재)
View → 우리 집 거실 (현재 화면)
Controller → 거실에 있는 나 (화면을 조작하는 주체)

1) this.getView()

- 컨트롤러가 관리하는 특정 화면에 접근할 때 사용
- this.getView().getRouter() 불가능 : Router는 View 소속이 아니고 Component가 관리함

01 getView() vs getOwnerComponent()

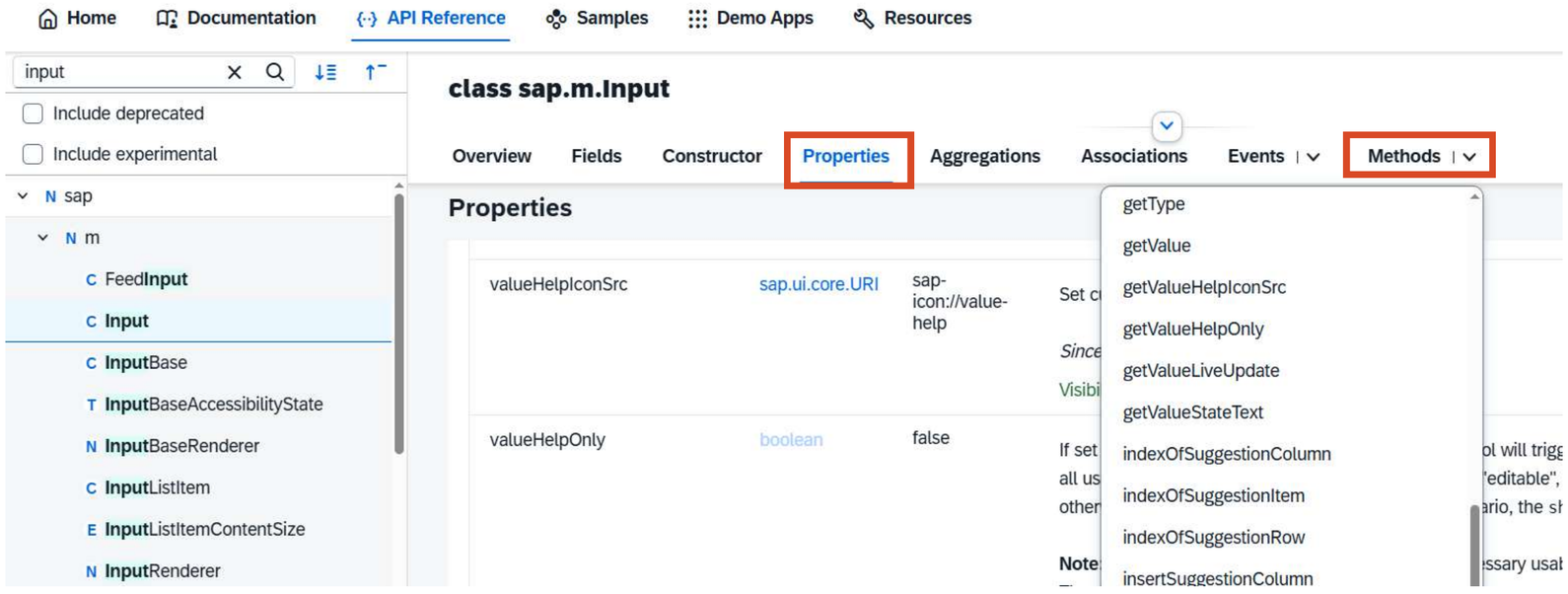
2) this.getOwnerComponent()

- 프로젝트 전체를 총괄하는 Component.js에 접근
- 프로젝트 공통 자원인 모델이나 화면 이동을 담당하는 라우터를 가져올 때 사용

```
this.getOwnerComponent().getModel()
this.getOwnerComponent().getRouter()
```

01 Method 복습

- 객체(Object): UI 화면을 구성하는 부품 (예: 버튼, 입력창, 텍스트)
- 메소드(Method): 객체가 할 수 있는 행동 (예: getValue(), setValue())
- 속성(Property): 객체가 가진 값, 상태 (예: value, text, visible)



01 Method 복습

1. getValue()
→ 사용자가 입력한 값을 읽음
2. setValue(value)
→ Input에 표시될 값을 설정함
3. getText()
→ 현재 표시 중인 텍스트를 읽어옴
4. setText(text)
→ 버튼이나 텍스트 컨트롤의 글자를 변경함
5. getView()
→ Controller와 연결된 View를 가져옴
6. byId(id)
→ View 안에서 특정 컨트롤을 찾음

01 Control과 이벤트 구조

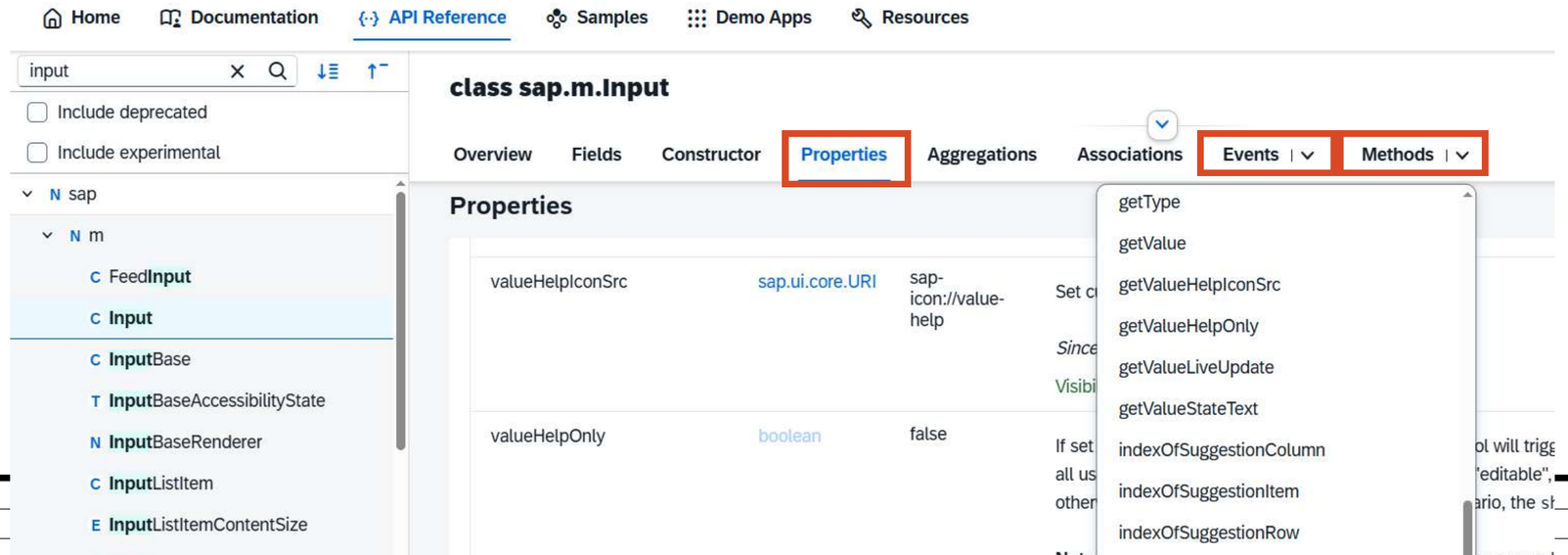
< Button >

→ 속성(property) : text, icon, type, ...

→ 이벤트(event) : press, ... → Controller(JS)에서 구현

→ 메서드는 개발자가 직접 호출해서 실행하는 기능 (예시: setText())

이벤트는 사용자 행동이 발생했을 때 자동으로 발생하는 신호 (예시: press)



01 VerticalLayout vs VBox 비교

공통점 : 둘 다 내부 컨트롤들을 위에서 아래로 세로로 순서대로 쌓는 용도로 사용

차이점 : VerticalLayout은 화면이 바뀌어도 그대로 쌓기만 하는 고정형 구조,
VBox는 화면에 맞게 크기·간격이 조절 가능.

```
<l:VerticalLayout>
</ l:VerticalLayout>
```

```
<VBox>
</VBox>
```

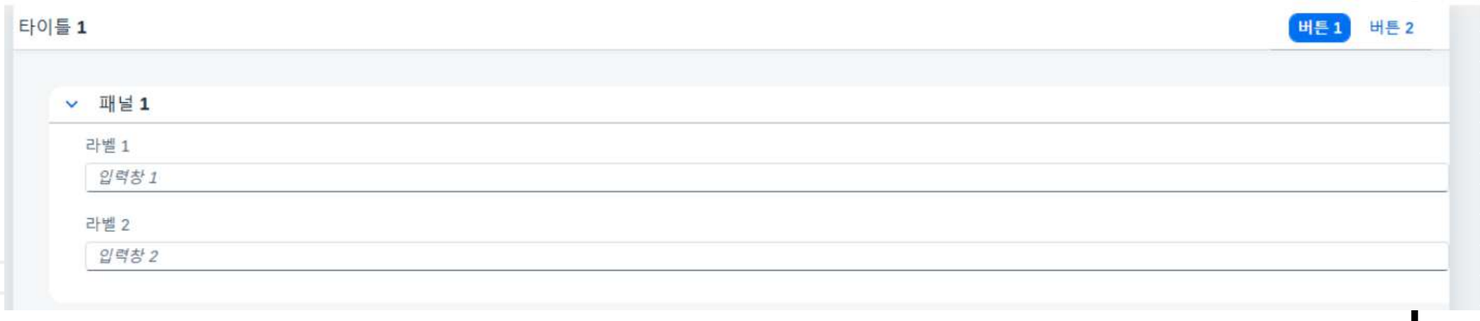
01 UI5 화면 구성 요소

1. Grid → 여러 요소를 표처럼 칸을 나누어 바둑판 모양으로 정교하게 정렬할 때 사용
2. Panel → 연관된 내용을 박스로 묶고, 제목 옆 화살표로 구역을 접거나 펼칠 때 사용
3. Page → 앱의 화면 한 장을 구성하는 가장 기본이 되는 틀

1. Label + Input → "이름 : [입력창]" 처럼 항목명과 입력칸을 한 쌍으로 묶어 입력받을 때 사용
2. ToolbarSpacer → 툴바 안에서 빈 공간을 밀어내어 버튼을 왼쪽이나 오른쪽 끝으로 보낼 때 사용
3. Border / Padding / Margin → 테두리, 안쪽 여백, 바깥 간격을 조절 가능

01 UI5 화면 구성 요소

```
6   <Page id="page" title="타이틀 1">
7
8       <headerContent>
9           <Toolbar>
10              <ToolbarSpacer />
11              <Button text="버튼 1" type="Emphasized" />
12              <Button text="버튼 2" />
13          </Toolbar>
14      </headerContent>
15
16      <content>
17          <Panel headerText="패널 1" expandable="true" expanded="true" class="sapUiMediumMargin">
18              <l:Grid defaultSpan="L12 M12 S12">
19                  <VBox>
20                      <Label text="라벨 1" />
21                      <Input placeholder="입력창 1" />
22                  </VBox>
23                  <VBox>
24                      <Label text="라벨 2" />
25                      <Input placeholder="입력창 2" />
26                  </VBox>
27              </l:Grid>
28          </Panel>
29      </content>
30  </Page>
```



01 Model

→ 데이터를 관리하고 View와 데이터를 연결해주는 역할

1) 클라이언트 모델 (JSON) → 서버 요청 없이 바로 화면에서 사용
2) 서버 모델 (OData) → 서버와 통신 후 데이터를 받아 사용

1) setModel() → View, Dialog 등에 모델을 연결
2) getModel() → 연결된 모델을 가져옴

1) 지역(Local) 모델 → Controller에서 생성, 해당 View에서만 사용 (getView())
2) 전역(Global) 모델 → manifest.json에 정의, 앱 전체에서 사용 (getOwnerComponent(), oData)

01 JSONModel 주요 메서드 정리

1) getData() → 모델이 가진 전체 데이터를 가져오는 함수

```
oModel.getData();
```

2) getProperty(path) → 모델의 특정 경로 데이터만 가져오는 함수

```
oModel.getProperty("/경로");
```

3) setData(data, merge) → 모델 데이터를 전체 교체 또는 병합하는 함수

```
oModel.setData(새로운데이터객체, 병합여부);
```

→ true → 기존 데이터 유지하면서 덮어쓰기
→ false → 전체 데이터 교체 (기본값)

4) setProperty(path, value) → 모델의 특정 경로 값만 부분 변경하는 함수

```
oModel.setProperty("/경로", 변경할값);
```

01 Binding

→ 모델(Model)과 화면(View)을 연결하여 데이터를 자동으로 표시하고 동기화하는 기능

< 바인딩 방식 >

1) One-Way: 모델 → 뷰 (데이터가 바뀌면 화면이 변함)
2) Two-Way: 모델 ↔ 뷰 (화면에서 값을 바꿔도 모델에 반영)
3) One-Time: 모델 → 뷰 (최초 1회만 반영)

01 1. Property Binding (속성 바인딩)

→ 컨트롤의 단일 속성(Text, Value 등)을 모델의 특정 값 하나에 연결

→ **{모델명>/경로}**

```
<Text text="{모델명>/경로}" />

<Text text="{customer>/CustomerName}" />
<Text text="{main>/state}" />
```

- 단일 속성 바인딩

```
<Page id="page" title="{/inputValue/firstName}">
    <!-- 기본 모델의 inputValue.firstName -->
</Page>
```

- 다중 속성 바인딩

```
<Page id="page" title="{local>/inputValue/firstName} {local>/inputValue/lastName}">
    <!-- local 모델의 name 객체 속성 firstName과 lastName 바인딩 -->
</Page>
```

01 2. Aggregation Binding (집합 바인딩)

→ 리스트나 테이블처럼 반복되는 UI에 모델의 배열(Array) 데이터를 연결

→ items="{모델명>/배열경로}"

```
<Table items="{모델명>경로}">
    <columns>...</columns>
    <items>
        <ColumnListItem>
            <cells>
                <Text text="{모델명>속성}" />
            </cells>
        </ColumnListItem>
    </items>
</Table>

<Table id="idCustomers" items="{customer>/Customers}">
    <columns>
        <Column><Text text="이름" /></Column>
        <Column><Text text="나이" /></Column>
    </columns>
    <items>
        <ColumnListItem>
            <cells>
                <Text text="{customer>CustomerName}" />
                <Text text="{customer>Age}" />
            </cells>
        </ColumnListItem>
    </items>
</Table>
```

01 3. Element Binding (바인딩 컨텍스트)

→ 컨트롤 하나를 특정 데이터 위치에 연결하여 그 안에 있는 값들을 바로 사용할 수 있게 하는 방식

→ control.bindElement("경로")

```
this.byId("idVBox").bindElement({
    path: "/company",
    model: "main"
});
```

```
<VBox id="idVBox" class="sapUiSmallMargin">
    <Text text="{main>name}" />
    <Text text="{main>city}" />
</VBox>
```

Q. 왜 {main>name}이고 {main>/name}이 아닐까?

{main>/name} (슬래시 있음)

→ 무조건 모델의 맨 처음(Root)부터 데이터를 찾기

{main>name} (슬래시 없음)

→ 기준점 내부에서 데이터를 찾기

01 4. Expression Binding (표현식 바인딩)

→ 데이터를 그대로 보여주지 않고, 화면(View)에서 간단한 조건문을 돌려 결과를 표시

→ **{= 조건식 ? '참일때' : '거짓일때'}**

```
{= ${model>/arr} >= 100}

<Text text="{= ${모델명>속성} > 18 ? '성인' : '청소년'}" />

<Text text="{= ${customer>Age} >= 20 ? '성인'
                : ${customer>Age} >= 14 ? '청소년'
                : '어린이' }" />
```

01 선택 이벤트 관련 함수

1. *onSelectionChange* → 사용자가 테이블에서 행을 클릭하면 자동으로 실행되는 함수
2. *getSource()* → 이 이벤트를 발생시킨 원래 컨트롤(예: 테이블, 버튼)을 알려줌
3. *getParameters()* → 이벤트 안에 들어있는 상세 정보 묶음을 가져옴 (선택된 행 등 포함)
4. *getSelectedItem()* → 현재 선택된 행을 가져오는 함수
5. *listItem* → 사용자가 클릭해서 선택된 그 행(Row) 객체를 가리킴

```
onSelectionChange: function(oEvent) {
    // 이벤트 처리 로직
}
```

```
let oTable = oEvent.getSource(); // 클릭한 테이블 객체
```

```
let oParams = oEvent.getParameters();
let oSelectedItem = oParams.listItem; // 선택된 Row
```

```
let oRow = oTable.getSelectedItem();
```

01 BindingContext & 데이터 연결 흐름

1. *getBindingContext()* → 이 행이 연결된 데이터 위치 정보(Context)를 가져오는 함수
2. *setBindingContext()* → 가져온 데이터 위치 정보를 다른 컨트롤에 연결하는 함수
3. *getPath()* → 데이터가 모델 안에서 어디에 있는지 경로를 알려주는 함수
4. *bindElement()* → 특정 데이터 경로를 직접 지정해서 컨트롤에 연결하는 함수
5. *getBinding()* → 이 컨트롤의 집합(items 등)이 어떤 데이터와 연결되어 있는지 가져오는 함수

```
let oBindingContext = oSelectedItem.getBindingContext("customer");
```

```
let sPath = oBindingContext.getPath();
```

```
oControl.setBindingContext(oBindingContext, "모델명")/;
```

```
this.getView().bindElement("/CarrierSet('AA')");
```

```
this.byId("idBookings").setBindingContext(oBindingContext, "customer");
```

```
this.byId("idProductsTable").getBinding("items").filter(aFilter);
this.byId("idProductsTable").getBinding("items").sort(aSorter);
```

01 데이터의 계층 구조 이해하기

1. EntitySet → 전체 데이터 모음 (배열)
2. Entity → 배열 안의 하나의 데이터
3. Complex Type (a) → Entity 안에 들어있는 객체 묶음
4. Field → 실제 값이 들어있는 속성

```
let entities = this.getView().getModel("model").getProperty("/EntitySet"); // 배열
let entity = entities[0];      // 첫 번째 Entity
let a = entity.a;              // Complex Type
let fieldValue = a.Field;      // 속성 값
```

```
{
  "EntitySet": [
    {
      "Name": "홍길동",
      "a": {
        "Field": "서울"
      }
    },
    {
      "Name": "박길동",
      "a": {
        "Field": "부산"
      }
    }
  ]
}
```

01 sap.m.Table (Item 기반 접근)

→ 데이터가 많지 않을 때 사용하며 선택된 행 객체 자체를 바로 가져올 수 있음

1) 선택된 단일 아이템에서 데이터 추출할 때 (테이블에 접근)

```
let oData = this.byId("idMobileTable").getSelectedItem().getBindingContext().getObject();
```

→ 테이블 객체를 가져와서 선택된 행의 객체를 가져와서 그 행이 연결된 위치 정보 가져와서 실제 데이터 객체 가져오기

2) 이벤트에서 직접 꺼내기

```
let oData = oEvent.getParameter("listItem").getBindingContext().getObject();
```

→ 클릭된 행의 객체를 가져와서 그 행이 연결된 위치 정보 가져와서 실제 데이터 객체 가져오기

3) 행 안 버튼 클릭 시

```
let oData = oEvent.getSource().getBindingContext().getObject();
```

→ 클릭된 버튼(버튼은 이미 테이블 안에 바인딩)을 가져와서 버튼이 속한 행의 데이터 위치 정보를 가져와서 실제 데이터 객체 가져오기

01 sap.ui.table.Table (Index 기반 접근)

→ 대량 데이터용으로 사용하며 몇 번째 줄인지 번호(Index)로 접근해야 함

1) rowContext로 데이터 바로 가져오기 (이벤트에서 꺼내기)

```
let oData = oEvent.getParameter("rowContext").getObject();
```

→ 클릭된 행의 데이터 위치 정보(rowContext는 클릭된 행이 가리키는 데이터 주소)를 가져와서 실제 데이터 객체 가져오기

2) rowIndex로 접근 (테이블에 접근)

```
let oData = this.byId("idEmployeeTable").getContextByIndex(this.byId("idEmployeeTable").getSelectedIndex()).getObject();
```

→ 테이블 객체를 가져와서 선택된 줄 번호 알아내 그 번호의 데이터 위치 정보를 가져와서 실제 데이터 객체 가져오기

3) 행 안 버튼 클릭 시

```
let oData = oEvent.getSource().getBindingContext().getObject();
```

→ 클릭된 버튼(버튼은 이미 테이블 안에 바인딩)을 가져와서 버튼이 속한 행의 데이터 위치 정보를 가져와서 실제 데이터 객체 가져오기

01 Filter (조건으로 걸러내기)

→ 테이블/리스트 데이터 중 조건에 맞는 항목만 화면에 표시하는 기능

Filter Operator

EQ → 같다
NE → 다르다
GT / LT → 크다 / 작다
GE / LE → 크거나 같다 / 작거나 같다
BT → 두 값 사이
Contains → 문자열 포함

- js 파일

```
new Filter("필드명", FilterOperator.연산자, 값1, 값2);
```

```
let oTable = this.byId("idTable");
let aFilters = [
    new Filter("Name", FilterOperator.Contains, "김")
];
oTable.getBinding("items").filter(aFilters);
```

- xml 파일

```
items="{
  path: '모델>/EntitySet',
  filters: [{ path:'필드', operator:'연산자', value1:'값' }]
}"
```

```
<Table items="{
    path: 'people>/People',
    filters: [
        { path: 'Name', operator: 'Contains', value1: '김' }
    ]
}">
```

01 Sorter (정렬하기)

→ 데이터를 특정 필드 기준으로 순서를 정렬하는 기능

false → 오름차순 (작은 값 → 큰 값)
true → 내림차순 (큰 값 → 작은 값)

- js 파일

```
new Sorter("필드명", true/false);
```

```
let oTable = this.byId("idTable");
let aSorters = [
    new Sorter("Age", true)   // true = 내림차순
];
oTable.getBinding("items").sort(aSorters);
```

- xml 파일

```
items="{
  path:'모델>/EntitySet',
  sorter:[{ path:'필드', descending:true }]
}"
```

```
<Table items="{
    path: 'people>/People',
    sorter: [
        { path: 'Age', descending: true }
    ]
}">
```

01 Formatter (포매터 함수)

→ 바인딩된 값이 화면에 표시되기 전에 데이터를 가공해서 보여주는 함수 (값 → 변환 → 화면 출력)

→ 파라미터에 undefined 이런 애들도 들어오기 때문에 if문 사용

```
<Text text="{path: '모델명>속성', formatter: '.포매터함수명'}" />
```

- view.xml 파일

```
<Text text="{
    path: 'customer>Age',
    formatter: '.fnFormatter'
}" />
```

- controller.js 파일

```
fnFormatter: function(sValue) {
    let result;
    // formatter 사용시 if문으로 value가 있는 경우를 체크하여
    // 처리하는게 좋다. value 값만 들어오는게 아니기 때문
    if(sValue) {
        let oModel = this.getView().getModel("customer");

        if(sValue >= 20) result = "성인";
        else if(sValue >= 17) result = "중학생";
        else if(sValue >= 14) result = "초등학생";
        else if(sValue >= 8) result = "초등학생";
        else result = "아동";
        return result;
    }
}
```

01 oModel.read() (데이터 조회)

→ OData 서버에서 데이터를 조회하는 함수

단 건 조회 → 결과가 객체 하나로 반환
다 건 조회 → results 배열 안에 여러 객체가 들어 있음

```
oModel.read("EntitySet", {
    filters: [oFilter],              // 조건
    urlParameters: { "$expand": "Relation" }, // 관련 데이터 확장
    success: function(oData) {},    // 조회 성공 시 실행
    error: function(oError) {}      // 조회 실패 시 실행
});
```

01 oModel.createKey() (엔티티 경로 생성)

→ OData 엔티티의 고유 주소(Key Path)를 만들어주는 함수

oModel.createKey(sPath, oParameters);

단일 키 조회 → /EntitySet(key=값) 또는 키가 하나일 경우 /EntitySet(값) 형태로 조회
복합 키 조회 → /EntitySet(key1=값,key2=값) 형태로 모든 키를 함께 지정해야 조회 가능

```
var oModel = this.getView().getModel();

var sKeyPath = oModel.createKey("/Products", {
   ProductID: 1
});

console.log(sKeyPath);
// 출력: "/Products(ProductID=1)"
```

01 라우터 - manifest.json

name → navTo 함수에서 호출할 라우터의 별명

pattern → 브라우저 URL 뒤에 붙는 경로

"" (빈 문자열): 앱 실행 시 처음 보여줄 메인 화면 (Root)
"CreateItem": 고정된 주소
"CreateItem/{pa1}": 필수 파라미터. {pa1} 자리에 반드시 값이 있어야 이동
"CreateItem/:pa1:": 선택 파라미터. 값이 없어도 해당 화면으로 이동이 가능

target: 패턴이 일치할 때 실제 띄워줄 목적지(targets)의 ID

id: 생성된 뷰 객체의 고유 ID
name: 실제 XML 뷰 파일의 이름

```
"routes": [
  {
    "name": "RouteMain",
    "pattern": ":?query:",
    "target": [
      "TargetMain"
    ]
  },
  {
    "name": "RouteCreateItem",
    "pattern": "CreateItem/{pa1}",
    "target": [
      "TargetCreateItem"
    ]
  }
],
"targets": {
  "TargetMain": {
    "id": "Main",
    "name": "Main"
  },
  "TargetCreateItem": {
    "id": "CreateItem",
    "name": "CreateItem"
  }
```

01 라우터 - 화면 이동 실행 (Controller - navTo)

getRouter() → manifest.json에 정의된 모든 길 안내 규칙을 가진 라우터 객체를 가져옴

navTo("RouteName", { ... }) → 지정된 이름의 라우트로 이동하며, { } 안의 파라미터 데이터를 URL에 담아 전달

```
onPress: function() {
    let oRouter = this.getOwnerComponent().getRouter();
    let pa1 = this.getView().byId("idHeaderTable").getSelectedItem().getBindingContext("oHeaderModel").getObject().Carrid;
    oRouter.navTo("RouteCreateItem", { pa1: pa1 });
}
```

01 라우터 - 도착 확인 및 파라미터 추출

getRoute("RouteCreateItem") → manifest.json에서 정의했던 특정 경로(RouteCreateItem) 객체를 얻어옴

.attachPatternMatched(...) → 이 화면의 URL 패턴과 일치하는 이동이 발생하면,
특정 함수(_attachPatternMatched)를 실행해라고 예약

this._attachPatternMatched → URL이 맞을 때 실행할 함수
this → 그 함수를 실행할 때 기준이 되는 Controller

```
onInit() {
    const oRouter = this.getOwnerComponent().getRouter();
    oRouter.getRoute("RouteCreateItem").attachPatternMatched(this._attachPatternMatched, this);
},
_attachPatternMatched: function(oEvent) {
    let gv_carrid = oEvent.getParameters().arguments.pa1;
```

01 라우터 - 메인 화면으로 복귀

showNavButton → 페이지 왼쪽 상단에 뒤로가기 화살표 버튼 여부
navButtonPres → 뒤로가기 버튼을 클릭시 동작

```
<Page id="page2" showNavButton="true" navButtonPress="onNavButtonPress" title="{i18n>title}">
```

navTo("RouteMain") → 메인화면으로 이동

```
onNavButtonPress: function() {
    let oRouter = this.getOwnerComponent().getRouter();
    oRouter.navTo("RouteMain");
},
```

01 라우터 - 메인 화면 복귀 감지 (Main 화면)

```
onInit() {
    const oRouter = this.getOwnerComponent().getRouter();
    oRouter.getRoute("RouteMain").attachPatternMatched(this._MainMatched, this);
```

```
_MainMatched: function() {
    sap.m.MessageToast.show("컴백 !");
},
```

01 Dialog 에서의 컨트롤 가져오기

```
let sKey = sap.ui.getCore().byId("idOption").getSelectedKey();
```

```
Fragment.load({
    id: "myDlg",
    name: "zuimemberb03.view.fragment.oDialog",
    controller: this
}).then(function(oDialog){
    this._oDialog = oDialog;
    this._oDialog.open();
}.bind(this));

var sKey = this._oDialog.byId("idOption").getSelectedKey();
```

02 라우팅 실습

- 3개의 페이지 구현
- Main 페이지에 A, B 페이지로 이동하는 버튼 생성
- 이동할 때 필수 파라미터로 Main에서 A로 또는 Main에서 B로 문자열 넘기기
- A나 B화면에서 받은 문자열을 byId()를 통해 TEXT 형식으로 출력
- 뒤로가기 버튼 클릭시 MAIN화면으로 이동하도록 구현