

네이버 Book Api연결해서 책 정보 받아오기



상단에 Application항목 선택한 후 “애플리케이션 등록”

애플리케이션 이름	<input type="text" value="BookSearch"/>
카테고리	<input type="text" value="도서/참고자료"/> ▼
이용 목적 ?	<input type="checkbox"/> 로그인 오픈API (네이버 아이디로 로그인) ? <input checked="" type="checkbox"/> 비로그인 오픈API ?
<div><div>! 오픈API 이용 권한 신청 ?</div><div><input type="checkbox"/> 지도 <input type="checkbox"/> 음성인식 <input checked="" type="checkbox"/> 검색 <input type="checkbox"/> 음성합성 <input type="checkbox"/> 기계번</div></div>	

등록 화면에 다음과 같이 체크

비로그인 오픈API 서비스 환경	OpenAPI를 적용할 서비스 URL을 입력
	안드로이드 앱 Package명 com.example.myPackage
	iOS Bundle ID

우리 프로젝트의 패키지명 입력

담당자 연락처	휴대폰 +82 10-7***-***3 수정 이메일 j6****@n*****.com 수정 연락처 인증은 최초 애플리케이션 등록시 1회만 하며, 휴대폰 인증만 하
이용약관	<b>제 1 조 목적</b> 이 약관은 네이버주식회사(이하 '회사'라 합니다)와 이용 고 에 회사가 제공하는 네이버 (www.naver.com, 이하 '네이버' p://developers.naver.com) 웹페이지에 기술된 Open API 서 니다)의 이용과 관련하여 "회사"와 "이용자"의 권리, 의무 및 타 필요한 사항을 구체적으로 규정함을 목적으로 합니다. <input checked="" type="checkbox"/> 해당 약관의 내용을 숙지하였으며, 이에 동의합니다. <input checked="" type="checkbox"/> 유해, 불법적인 내용과 관련되거나, 신청하지 않은 애플리케이션 등에
등록하기 취소	

최초 1회 휴대폰 인증 받고, 앱 등록!!

## 내 애플리케이션

Book Search

Test

애플리케이션 등록

제휴 신청

기존 API키 조회

## Book Search

개요

설정



## Book Search

개발중 ?

### Client ID

IutY6eYZKanfHEIdM6K

### Client Secret

TlPhbNz\_Ru

생성된 애플리케이션의 ID와 Secret복사

개 개발가이드 Open Source

네이버 아이디로 로그인

네이버 공유하기

지도

검색

블로그

개발 가이드 -> 검색 -> 책 항목으로 이동

## 2. API 기본 정보

메서드	인증	요청 URL	출력 포맷	설명
GET	-	https://openapi.naver.com/v1/search/book.xml	복사 해서 추후에 활용	책 기본 검색
GET	-	https://openapi.naver.com/v1/search/book_adv.xml	XML	책 상세 검색

## 3. 요청 변수

요청 변수	설명	비고
query	query는 검색할 단어로써, 반드시 UTF-8형태로 인코딩 해서 전송하도록 한다. 검색을 원하는 문자열로서 UTF-8로 인코딩한다.	-
display	10(기본값), 100(최대) 검색 결과 출력 건수 지정	-

## NaverActivity생성

### activity\_naver.xml정의

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    tools:context=".NaverActivity" >
```

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
```

```
<EditText
    android:id="@+id/search"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:hint="search" />
```

```
<Button
    android:id="@+id/search_btn"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="start" />
</LinearLayout>
```

```
<ListView
    android:id="@+id/myListView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

```
</LinearLayout>
```

## book\_item.xml

layout폴더에서 마우스 우클릭 하여 **book\_item.xml**파일 생성  
이놈은 위의 리스트 뷰에 추가될 형식

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
```

```
<ImageView
    android:id="@+id/book_img"
    android:layout_width="100dp"
    android:layout_height="200dp"/>
```

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:layout_gravity="center"
    android:paddingLeft="20dp">
```

```
<TextView
```

```

        android:id="@+id/book_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

<TextView
    android:id="@+id/book_author"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

<TextView
    android:id="@+id/book_price"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
</LinearLayout>
</LinearLayout>

```

**BookModel**클래스 생성 및 정의(언어올 정보들을 객체화 할 클래스)

```

public class BookModel {
    //언어올 정보들을 저장할 변수들 지정
    //책 제목, 저자, 가격, 이미지
    private String b_title, b_author, b_price, b_img;

    //셋터겟터 작성
    public String getB_title() {
        return b_title;
    }

    public void setB_title(String b_title) {
        this.b_title = b_title;
    }

    public String getB_author() {
        return b_author;
    }

    public void setB_author(String b_author) {
        this.b_author = b_author;
    }
}

```

```

    }

    public String getB_price() {
        return b_price;
    }

    public void setB_price(String b_price) {
        this.b_price = b_price;
    }

    public String getB_img() {
        return b_img;
    }

    public void setB_img(String b_img) {
        this.b_img = b_img;
    }
}

```

## Parser클래스 생성 및 정의

(웹에서 요소(책제목, 저자, 가격 등)를 검색하여 준비된 vo객체(BookModel)에 저장하여 추후에 액티비티의 imageView나 TextView에 적용하기 위한 준비를 하는 클래스)

```

public class Parser {

    BookModel vo;
    String myQuery = ""; // 검색어

    public ArrayList<BookModel> connectNaver(
        int start, ArrayList<BookModel> list){
        //검색 시작위치, arrayList

        // 생성되는 요소들을 ArrayList에 적재한다.
        try {
            myQuery = URLEncoder.encode(
                NaverActivity.search.getText().toString(), "UTF8");

```

```

// ↑ ↑ 잠시 후NaverActivity에 상수로 추가할 editText의 값

int count = 10; //검색결과 10건 표시
String urlStr =
"https://openapi.naver.com/v1/search/book.xml?query=" + myQuery +
"&start=" + start + "&display="+count;

//URL클래스를 생성하여 접근할 경로 설정
URL url = new URL(urlStr);

//url클래스의 연결 정보를 connection에 전달
URLConnection connection =
(HttpURLConnection)url.openConnection();

// ■ GET은 주소줄에 값이 ?뒤에 쌍으로 이어붙고 POST는 숨겨져서(body안에) 보내진다.
// ■ GET은 URL에 이어붙기 때문에 길이제한이 있어서 많은양의 데이터는 보내기 어렵고
// POST는 많은 양의 보내기에도 적합하다.(역시 용량제한은 있지만)

//네이버 API는 기본적으로 GET방식을 지원한다.
connection.setRequestMethod("GET");

//발급받은ID
connection.setRequestProperty(
    "X-Naver-Client-Id", "vuvmNrTZXP2o7acEZMnD");

//발급받은 secret
connection.setRequestProperty(
    "X-Naver-Client-Secret", "C2hz1bNLyw");

connection.setRequestProperty(
    "Content-Type", "application/xml");

// 위의 URL을 수행하여 받을 자원을
// 파싱(xml에 대입)할 객체를 준비시킨다.
XmlPullParserFactory factory =

```



```

        XmlPullParserFactory.newInstance();

// 위에서 생성된 factory를 가지고 파서객체를 생성한다.
XmlPullParser parser = factory.newPullParser();

// connection을 통해 파싱을 한다.
parser.setInput(connection.getInputStream(), null);

//이제 파서를 통하여 각 요소(Element)들을 반복 수행처리한다.
int parserEvent = parser.getEventType();

while(parserEvent != XmlPullParser.END_DOCUMENT){
    // XML문서의 끝이 아닐 때만 반복한다.
    // 시작태그일 때만 태그의 이름을 알아낸다.
    if(parserEvent == XmlPullParser.START_TAG){
        String tagName = parser.getName();

        if(tagName.equalsIgnoreCase("title")){
            vo = new BookModel();
            String title = parser.nextText();

            // 네이버는 검색어가 들어간 단어에
            // <b>태그가 들어가 있다.
            // 이것을 제거하기 위해 정규표현식을
            // 사용한다.
            Pattern pattern
                = Pattern.compile("<.*?>");

            Matcher matcher
                = pattern.matcher(title);

            if(matcher.find()){
                String s_title
                    = matcher.replaceAll("");
            }
        }
    }
}

```

```

        vo.setB_title(s_title);
    }else{
        vo.setB_title(title);
    }
} //title if()
else if(tagName.equalsIgnoreCase("image")){
    String img = parser.nextText();
    vo.setB_img(img);
} //image if()

else if(tagName.equalsIgnoreCase("author")){
    String author = parser.nextText();
    vo.setB_author(author);
} //author if()

else if(tagName.equalsIgnoreCase("price")){
    String price = parser.nextText();
    vo.setB_price(price);
    list.add(vo);
} //price if()
} //XmlPullParser.START_TAG

parserEvent = parser.next(); //다음요소

} //while의 끝!!

} catch (Exception e) {
    e.printStackTrace();
}

//현재 영역에 오면 서버로부터 받은 XML자원이
// 모두 Ex1BookModel객체로 생성되어
// ArrayList에 적재되어 있다.
// 이것을 반환!!

```

```

        return list;
    } //connectNaver()

}

```

스크롤이 가장 아래로 내려가면 10개씩 추가로 로드하는 코드를 구현할 건데, “추가로 로딩합니다”라는 문장을 리스트 뷰 하단에 붙여서 화면에 출력해줄 것이다. 이를 footer라고 하는데 footer를 구현하기 위한 레이아웃을 생성해준다.

res -> layout폴더에 footer.xml파일 생성 및 내용 정의

```

<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:background="#E6FFFF">

    <ProgressBar
        android:layout_width="30dp"
        android:layout_height="30dp"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="로드중..."
        android:layout_marginLeft="10dp"/>
</LinearLayout>

```

↑ ↑ 이녀석은 잠시후에 리스트뷰에 붙여주도록 하자.

NaverActivity에 내용 추가

```

public class NaverActivity extends AppCompatActivity {

```

```
private ListView myListView;
static EditText search; //검색어(스텍 선언)
private Button search_btn;
Parser parser;
ArrayList<BookModel> list;
ViewModelAdapter adapter;

int start = 1; //검색을 시작할 인덱스 번호

ProgressDialog dialog_progress; //로딩을 위한 다이얼로그

//스크롤링을 통한 추가 로드를 위해 필요한 변수
LayoutInflater inflater;
View footerView;
boolean mLockListView = true;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_naver);

    //다이얼로그 객체 준비
    dialog_progress = new ProgressDialog(this);
    dialog_progress.setMessage("Loading...");

    //객체검색
    myListView = (ListView)findViewById(R.id.myListView);

    //리스트뷰의 오버스크롤 이펙트 제거
    myListView.setOverScrollMode(View.OVER_SCROLL_NEVER);

    search = (EditText)findViewById(R.id.search);
    search_btn = (Button)findViewById(R.id.search_btn);
```

```
search_btn.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
```

```
    public void onClick(View v) {
```

```
        // 서버 연결과 결과를 받는 싱크 클래스 만들기
```

```
        //아래의 두줄은 NaverAsync클래스 작성후에 객체생성
```

```
        //한글자 이상 입력받은 경우에만 검색
```

```
        if(search.getText().toString().trim().length() > 0) {
```

```
            dialog_progress.show(); //로딩시작
```

```
            list = new ArrayList<>();
```

```
            adapter = null;
```

```
            start = 1;
```

```
            //new NaverAsync().execute("안", "녕");
```

```
            //↑ ↑이런식으로 값 집어넣어서도 한번 보여주자.
```

```
            new NaverAsync().execute();
```

```
        }
```

```
    } //onClick
```

```
});
```

```
parser = new Parser();
```

```
//로딩을 표시하기 위해 미리 만들어둔 footer를 등록하기 위한 준비
```

```
mInflater = (LayoutInflater)
```

```
    getSystemService(Context.LAYOUT_INFLATER_SERVICE);
```

```
footerView = mInflater.inflate(R.layout.footer, null);
```

```
} //onCreate()
```

```
//AsyncTask클래스는 생성시에 세개의 제네릭타입을 가진다.
```

```

class NaverAsync extends
    AsyncTask<String, String, ArrayList<BookModel>> {

    // 객체가 생성될 때 두번째로 호출되는 함수!!
    // doInBackground요놈은 필수!!!!
    // 각종 반복이나 제어등 주된 처리 로직을 담당
    @Override
    protected ArrayList<BookModel> doInBackground(
        String... params) {
        return parser.connectNaver(start, list);
    }

    @Override
    protected void onPostExecute(ArrayList<BookModel> result) {
        //doInBackground에서 통신을 마친 결과가
        //onPostExecute의 result로 넘어온다. 이것을 이용해서 리스트
        뷰에 화면을 갱신한다.
        if(adapter == null) {
            adapter = new ViewModelAdapter(
                NaverActivity.this, R.layout.book_item, result);

            // 리스트 뷰에 스크롤 리스너를 등록합니다.
            myListView.setOnScrollListener(scrollListener);

            //리스트뷰에 footer등록!! setAdapter 이전에 해야 합니다.
            myListView.addFooterView/footerView);

            //리스트뷰에 어댑터 세팅
            myListView.setAdapter(adapter);
        }

        //리스트뷰의 변경사항이 있다면 갱신
        adapter.notifyDataSetChanged();
        mLockListView = false;
    }
}

```

```
        dialog_progress.dismiss(); //로딩종료
    }
} //NaverAsync 클래스의 끝
```

//리스트 뷰의 스크롤 감지자

```
AbsListView.OnScrollListener scrollListener =
    new AbsListView.OnScrollListener() {
```

```
    @Override
```

```
    public void onScrollStateChanged(
        AbsListView absListView, int i) {
```

```
        //onScrollStateChanged 는 현재 리스트뷰의 상태를 알려줍니다.
        // scrollState 으로 넘어오는 상태값은 다음과 같은 3 가지입니다.
```

```
        //SCROLL_STATE_FLING ( 2 ) :
```

```
        //터치 후 손을 떼 상태에서 아직 스크롤 되고 있는 상태입니다.
```

```
        //SCROLL_STATE_IDLE ( 0 ) :
```

```
        //스크롤이 종료되어 어떠한 애니메이션도 발생하지 않는 상태입니다.
```

```
        //SCROLL_STATE_TOUCH_SCROLL ( 1 ) :
```

```
        //스크린에 터치를 한 상태에서 스크롤하는 상태입니다.
```

```
    }
```

```
    @Override
```

```
    public void onScroll(AbsListView absListView,
        int firstVisibleItem,
        int visibleItemCount,
        int totalItemCount) {
```

```
        //onScroll() 메서드는 스크롤이 발생하는 동안 지속적으로 호출되는
        //메서드로 현재 보여지는 리스트뷰에서 상단에 보여지는 항목의
        //index 와 현재 리스트뷰에서 보여지는 항목의 수, 그리고 리스트뷰의
        //총 항목의 수를 알려줍니다.
```

```
        // 현재 가장 처음에 보이는 항목index와 보여지는 셀번호를 더한값이
```

```

// 전체의 숫자와 동일해지면 가장 아래로 스크롤 되었다고 가정합니다.
int count = totalItemCount - visibleItemCount;

if(firstVisibleItem >= count &&
    totalItemCount != 0 && mLockListView == false){

    mLockListView = true;

    //1. 총 1000개까지 로드(네이버에서 제공하는 최대 start값이 1000임)
    //할것이므로 +10을 했을 때를 생각하여, 990보다
    //작을 때 까지만 start를 10씩 증가시킨다.

    //2. 한번에 10개씩 검사하므로, list.size()가 10보다 작을 경우에는
    //스크롤을 만들 필요가 없다고 판단해도 좋다.
    //if( ( start < 총 개수 - 한번에 검색하는 수 ) && ( list.size() >= 한번에 검색하는 수 ) ){ }
    if(start < 1000 - 10 && list.size() >= 10) {

        if(start >= list.size()){
            /*list.size() -> 검색결과가 15개라고 가정해보면. 검색이 완료 되었음에도 불구하고 15는
            990보다 작기 때문에 start += 10을 반복하며 if문을 계속 수행하게 된다.
            이것을 방지하기 위해 start를 강제로 990으로 만들어서 더 이상 if문으로 접근해 통신을 수
            행하지 않도록 한다*/

            start = 1000 - 10; //총 갯수 - 한번에 검색하는 수
        }

        start += 10; //열개씩 추가 로드할 예정

        //통신!
        //만약 footerView.setOnClickListener를 사용할 예정이라면 주석처리 하면 된다.
        new NaverAsync().execute();

    }else{
        Toast.makeText(getApplicationContext(),
            "더 불러올 내용이 없습니다" ,
            Toast.LENGTH_SHORT).show();
    }
}

```



```

        //footer 제거
        myListView.removeFooterView(footerView);
    }
}
} //onScroll()
};
} //class end

```

## ViewModelAdapter 생성 및 정의

// 배열을 참조하는 어댑터 클래스

// xml파일의 각 뷰들에게 Parser를 통해 반환된 BookModel정보를 전달하는 클래스다.

```
public class ViewModelAdapter extends ArrayAdapter<BookModel> {
```

```
    Context context;
```

```
    private ArrayList<BookModel> list;
```

```
    private BookModel vo;
```

```
    private int resource;
```

//생성자 정의

```
public ViewModelAdapter(
    Context context, int resource,
    ArrayList<BookModel> objects) {
    super( context, resource, objects );
```

```
    list = objects;
```

```
    this.context = context;
```

```
    this.resource = resource;
```

```
} //생성자
```

@Override

```
public View getView( int position,
    View convertView, ViewGroup parent ) {
```

```
    LayoutInflater linf =
```

```
(LayoutInflater)context.getSystemService(
    Context.LAYOUT_INFLATER_SERVICE);

if(convertView == null) {

    convertView = linf.inflate(resource, null);

    vo = list.get(position); // 표현할 책정보객체

    if(vo != null){
        // 값들을 표현할 TextView들을 검색!
        TextView title =
            (TextView)convertView.findViewById(R.id.book_title);

        TextView author =
            (TextView)convertView.findViewById(R.id.book_author);

        TextView price =
            (TextView)convertView.findViewById(R.id.book_price);

        ImageView img =
            (ImageView)convertView.findViewById(R.id.book_img);

        // 검색된 TextView에 문자 값을
        // vo에서 찾아 넣어준다.
        if(title != null)
            title.setText( "Title: " + vo.getB_title() );

        if(author != null)
            author.setText( "Author: " + vo.getB_author() );

        if(price != null)
            price.setText( "Price: " + vo.getB_price() );
```

```

        //이미지 로드 작업!!
        //이미지를 로드 하는 Async생성과 실행
        //제일 마지막에 실행!!
        new ImgAsync( img, vo ).execute();
    }
}
return convertView;
} //getView()
}

```

위로 올라가서 NaverActivity의 onPostExecute에 내용 추가  
 퍼미션!!! 인터넷으로 꼭 추가해주고 일단 컴파일.  
 이미지를 제외한 책 정보들을 잘 보일 것이다.  
 이제 이미지를 로드해보자.

ViewModelAdapter에 내용 추가하기!!!!!!!!!!

(추가된 부분은 파란색으로 표시함)

```
public class ViewModelAdapter extends ArrayAdapter<BookModel>{
```

```
.....
```

```

    public ViewModelAdapter(Context context, int resource,
        ArrayList<BookModel> objects) {
        super(context, resource, objects);
        .....
    }

```

```

@Override
public View getView(int position,
    View convertView, ViewGroup parent) {
    .....

```

ViewModelAdapter아래에 클래스 추가하고

```

        제일 마지막으로 아래 ↓↓↓코드 실행!!!
        new ImgAsync(img, vo).execute();
    }
    return convertView;
} //getView()

```

//getView() 아래ImgAsync 클래스 추가.

```

class ImgAsync extends AsyncTask<String, String, Bitmap> {

    Bitmap bm;
    ImageView mImg;
    BookModel model;

    public ImgAsync(ImageView mImg, BookModel model) {
        this.mImg = mImg;
        this.model = model;
    }

    @Override
    protected Bitmap doInBackground(String... params) {
        try{
            URL img_url =
                new URL(model.getB_img());

            // 위의 경로를 가지고 (스트림을 얻어)
            // 로드작업을 수행한다.
            BufferedInputStream bis =
                new BufferedInputStream(
                    img_url.openStream());

            // (스트림으로부터) Bitmap생성
            bm = BitmapFactory.decodeStream(bis);
            bis.close();// 스트림 닫기
        }
    }
}

```

```

        }catch(Exception e){
            e.printStackTrace();
        }

        //onPostExecute로 bitmap반환
        if(bm != null) {
            return bm;

        }else{
            //이미지가 없는 경우에는 기본 이미지를 반환
            Bitmap bitmap = BitmapFactory.decodeResource(
                context.getResources(), R.mipmap.ic_launcher);
            return bitmap;
        }

    }//doInBackground()

    @Override
    protected void onPostExecute(Bitmap result) {
        mImg.setImageBitmap(result);
    }

} //ImgAsync

} //class end

```

끝!! 다시 컴파일 하면 이미지도 나올것임

마지막으로 리스트뷰 각 항목의 클릭이벤트.

ViewModelAdapter 클래스에 내용 추가.

(추가된부분 파란색으로 표시. 01부터 번호 순서대로 접근)

```

public class ViewModelAdapter extends
    ArrayAdapter<BookModel> implements OnItemClickListener{ //01

```

```
Context context;
private ArrayList<BookModel> list;
private BookModel vo;
```

### 03. 생성자에 내용 추가.

```
//생성자 정의
public ViewModelAdapter(Context context, int textViewResourceId,
    ArrayList<BookModel> objects, ListView myListView) {
    super(context, textViewResourceId, objects);

    list = objects;
    this.context = context;
```

### 04. 리스트뷰에 아이템클릭 감지자 등록

```
myListView.setOnItemClickListener\(this\);
}
```

```
@Override
public View getView(int position,
    View convertView, ViewGroup parent) {
    .....
    return convertView;
} //getView()
```

### 02. 아이템클릭 이벤트 감지자 오버라이딩

//아이템클릭 이벤트 감지자를 구현하면서 생긴 onItemClick메서드.

```
@Override
public void onItemClick(AdapterView<?> parent,
    View v, int i, long id) {

    String bookImg = list.get(i).getB_img();

    //서버에서 가져온 이미지의 경로에서 이미지 이름만 추출
    String bookId =
        bookImg.substring(bookImg.lastIndexOf('/') + 1,
            bookImg.lastIndexOf(".jpg"));
```

```

        //추출한 이미지 이름을 통해 상세정보 페이지로 연결할 url준비
        String bookLink =
            "http://book.naver.com/bookdb/book_detail.nhn?bid=" +
                                                    bookId;

        Intent intent = new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse(bookLink));
        context.startActivity(intent);//상세보기 페이지로 이동
    }
} //ViewModelAdapter class end
하단 싱크클래스는 생략....

```

마지막으로 NaverActivity에서 어댑터 클래스로 listView를 넘겨주도록 하자.  
NaverActivity의 NaverAsync클래스 onPostExecute에 내용 추가하기.

```

class NaverAsync extends
    AsyncTask<String, String, ArrayList<BookModel>>{

    @Override
    protected ArrayList<BookModel> doInBackground(
        String... params) {
        .....
    }

    @Override
    protected void onPostExecute(ArrayList<BookModel> result) {

        if(adapter == null) {
            adapter = new ViewModelAdapter(
                NaverActivity.this, R.layout.book_item,
                result, myListView);
            .....
            myListView.setAdapter(adapter);
        }
        .....
        dialog_progress.dismiss();//로딩종료
    }
}

```

```
        } //onPostExecute()
    } //NaverAsync
```

실행해서 리스트뷰 클릭으로 내용표시가 잘 되는지 확인해보자!!!

마지막으로 가상키보드의 엔터키를 검색기능으로 바꿔서 버튼을 누르는 대신 키보드의 엔터키를 눌러도 검색이 가능하도록 해보자.

activity\_naver.xml의 에디텍스트에 속성 추가

```
<EditText
    android:id="@+id/search"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"
    android:hint="search"
    //엔터대신 검색으로 키보드 모양변경
    android:imeOptions="actionSearch"
    android:inputType="text"/>
```

NaverActivity에 내용 추가.

```
public class NaverActivity extends AppCompatActivity {

    .....

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_naver);

        .....

        search = (EditText)findViewById(R.id.search);
        search_btn = (Button)findViewById(R.id.search_btn);
```



```

search_btn.setOnClickListener(new View.OnClickListener() {

    @Override
    public void onClick(View v) {
        // 서버 연결과 결과를 받는 싱크 클래스 만들기
        //아래의 두줄은 NaverAsync클래스 작성후에 객체생성

        //한글자 이상 입력받은 경우에만 검색
        if(search.getText().toString().trim().length() > 0) {

            dialog_progress.show();//로딩시작

            list = new ArrayList<>();
            adapter = null;
            start = 1;

            //new NaverAsync().execute("안", "녕");
            //↑ ↑ 이런식으로 값 집어넣어서도 한번 보여주자.
            new NaverAsync().execute();

            //가상키보드 감추기
            InputMethodManager imm =
(InputMethodManager)getSystemService(INPUT_METHOD_SERVICE);
imm.hideSoftInputFromWindow(search.getWindowToken(), 0);
        }
    } //onClick
});

parser = new Parser();

//로딩을 표시할 푸터를 등록하기위한 준비
mInflater = (LayoutInflater)
getSystemService(Context.LAYOUT_INFLATER_SERVICE);
footerView = mInflater.inflate(R.layout.footer, null);

```

```
//에디트텍스트에서 가상키보드의 검색버튼이 클릭되었는지를 감지
search.setOnEditorActionListener(
    new TextView.OnEditorActionListener() {
        @Override
        public boolean onEditorAction(
            TextView v, int actionId, KeyEvent event) {

            switch (actionId) {
                case EditorInfo.IME_ACTION_SEARCH:
                    //에뮬레이터는 가상키보드가 없기 때문에 해당영역이 실행되지 않는다.
                    //디바이스에서는 엔터키가 돋보기 모양으로 바뀌어 있는것을 확인할 수 있다.
                    //Toast.makeText(getApplicationContext(),
                        "검색", Toast.LENGTH_LONG).show();

                    //search_btn을 클릭했을 때와 같은 일을 하면 된다.
                    //search_btn의 클릭 이벤트를 강제로 호출하기.
                    search_btn.performClick();
                    break;

                default:
                    //에뮬레이터로 키보드의 엔터값을 받아 검색하고 싶다면 여기에 작성하면 된다.
                    //Toast.makeText(getApplicationContext(),
                        "기본", Toast.LENGTH_LONG).show();
                    search_btn.performClick();
                    return false;
            }
        }
    });
}

//onCreate()

class NaverAsync extends
    AsyncTask<String, String, ArrayList<BookModel>> {
```

```

@Override
protected ArrayList<BookModel> doInBackground(
    String... params) {
    return parser.connectNaver(start, list);
}

@Override
protected void onPostExecute(ArrayList<BookModel> result) {
    if(adapter == null) {
        adapter = new ViewModelAdapter(
            NaverActivity.this, R.layout.book_item,
            result, myListView);
        .....
    }

    if(result.size() == 0) { //가져온 결과가 없을 때
        Toast.makeText(getApplicationContext(),
            "검색 결과가 없습니다.", Toast.LENGTH_LONG).show();
        //footer 제거
        myListView.removeFooterView(footerView);
        dialog_progress.dismiss();
        return;
    }

    //리스트뷰의 변경사항이 있다면 갱신
    adapter.notifyDataSetChanged();
    mLockListView = false;
    dialog_progress.dismiss();//로딩종료
}
} //NaverAsync
.....
} //class end

```

엔터키로 검색되는지, 검색결과가 없을 때 메시지가 잘 나오는

지, 결과 확인하고 마무리