

43.HTTP 통신

- GET vs POST
- HTTP 통신 방법
 - **HttpURLConnection**
 - Apache HttpClient
 - SDK 23부터 사용 중지됨.
 - Volley
 - 2013년에 발표된 라이브러리
 - 현재는 개발 중지됨. 사용은 가능.
 - **Glide**
 - Queue + Thread Pool을 이용한 **동시 병행 요청 라이브러리**
 - JSON / XML / **Image** 등 다양한 Response 형식 처리 가능
 - JSON를 모델 객체로 변환
 - Fresco
 - facebook에서 공개한 이미지라이브러리입니다.



HTTP 통신

- HTTP 통신
 - HTTP는 인터넷에서 데이터를 주고 받기 위한 프로토콜
- 퍼미션을 추가
 - `AndroidManifest.xml`에 권한 추가
 - `android.permission.INTERNET`
 - `android.permission.ACCESS_NETWORK_STATE`
 - 네트워크 연결 상태를 확인
- 네트워크 연결은 스레드 내에서 진행
 - `AsyncTask` 사용.



HTTP 통신

- HTTP 프로토콜



- HTTP 요청/응답 메시지 포맷





HTTP 통신

- 응답 메시지 포맷(상태 코드)

분류	상태 코드	내용	비고
Information	1xx (100~199)	서버에 클라이언트의 요청이 접수되었고 현재 작업을 처리 중이라는 의미	
Success	2xx (200 ~299)	클라이언트의 요청이 성공적으로 처리 되었음을 의미.	
Redirection	3xx (300~399)	파일이 이동되었을 의미. 이동하는 위치를 응답에 포함시킨다.	
Client Error	4xx (400~499)	클라이언트의 요청이 불완전하며, 클라이언트 요청을 성고 하려면 추가적인 다른 정보가 필요하다는 것을 의미.	
Server Error	5xx (500~599)	서버가 에러를 발생시켰으며, 클라이언트 요구를 처리할 수 없는 경우를 의미.	

상태 코드	내용
200	성공. 클라이언트 요청을 서버가 제대로 처리한 경우이다.
403	서버가 요청을 거부한 경우에 해당한다.
404	서버에 존재하지 않는 페이지에 대한 요청이 있을 경우에 발생하는 상태 코드이다.
500	서버에 오류가 발생하여 요청을 수행할 수 없는 경우이다.
503	서버의 문제로 인하여 사용할 수 없는 경우에 발생하는 상태 코드이다.



HTTP 통신

- HTTP 요청 방법

항목	설명	안전	캐시화
GET	클라이언트가 서버에 정보를 요청하기 위해서 사용.	X	X
POST	클라이언트가 서버에 정보를 전달하기 위해서 사용.		
PUT	서버의 정보를 업데이트하기 위해서 사용.		
DELETE	서버의 정보를 삭제하기 위해서 사용.		
TRACE	클라이언트의 요청을 그대로 반환. 서버 상태를 확인하기 위한 목적으로 사용.		
HEAD	HEAD 정보만 요청한다. 해당 자원의 존재 유무 및 서버 상태 확인.	X	X
OPTIONS	서버가 지원하는 METHOD의 종류를 요청할 때 사용.		

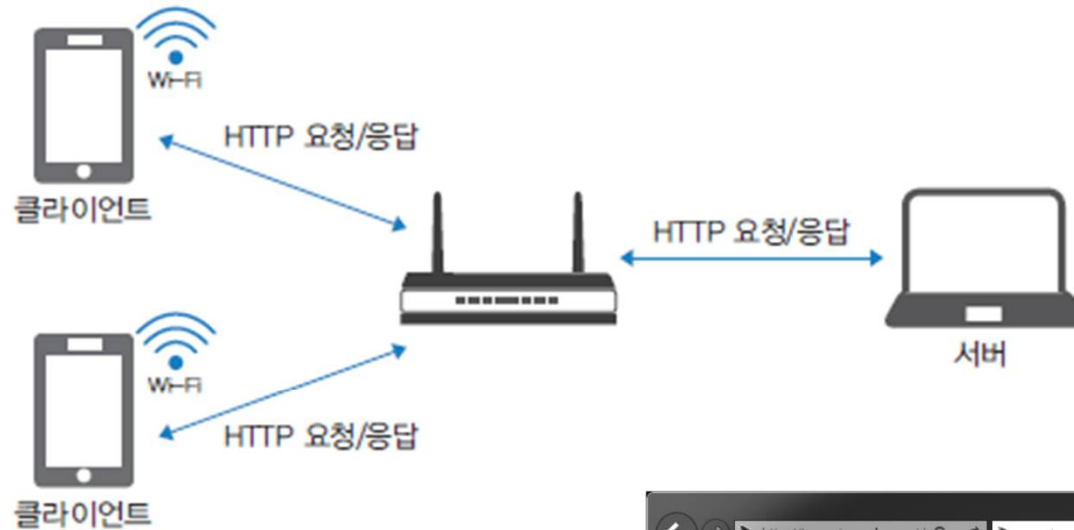
- GET vs POST

항목	GET 방식	POST 방식
사용 용도	서버 정보를 가져오는데 사용	서버의 정보를 추가 및 수정하는데 사용
전달 방식	url에 정보 노출됨	Body 안에 있기 때문에 노출 되지 않음.
용량 제한	문자열로 용량 제한 있음. window에서는 최대 256자를 넘을 수 없음.	GET 방식보다 많은 데이터를 전달 가능함. 용량 제한이 있음
전송 데이터	문자열	문자열, 파일
전송 속도	POST 방식보다 빠르다.	GET 방식 보다 느리다.

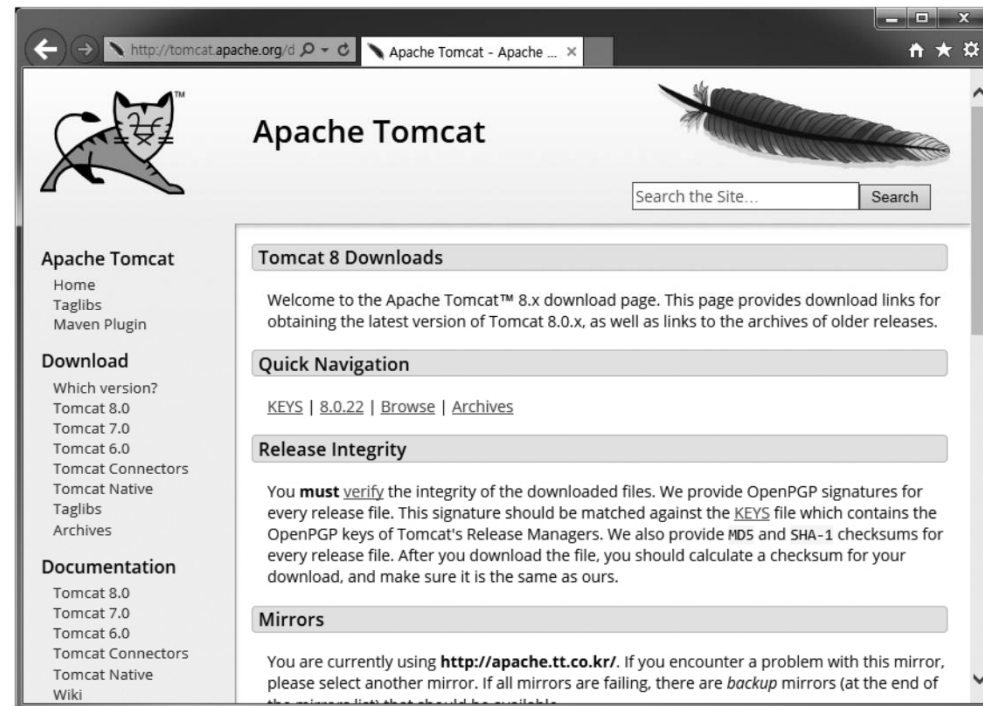


HTTP 통신을 위한 테스트 환경 만들기

- Wi-Fi를 이용한 테스트 환경 구성



- JDK 및 아파치 톰캣 설치





JSON , GSON 추가

- AndroidManifest.xml 권한 추가

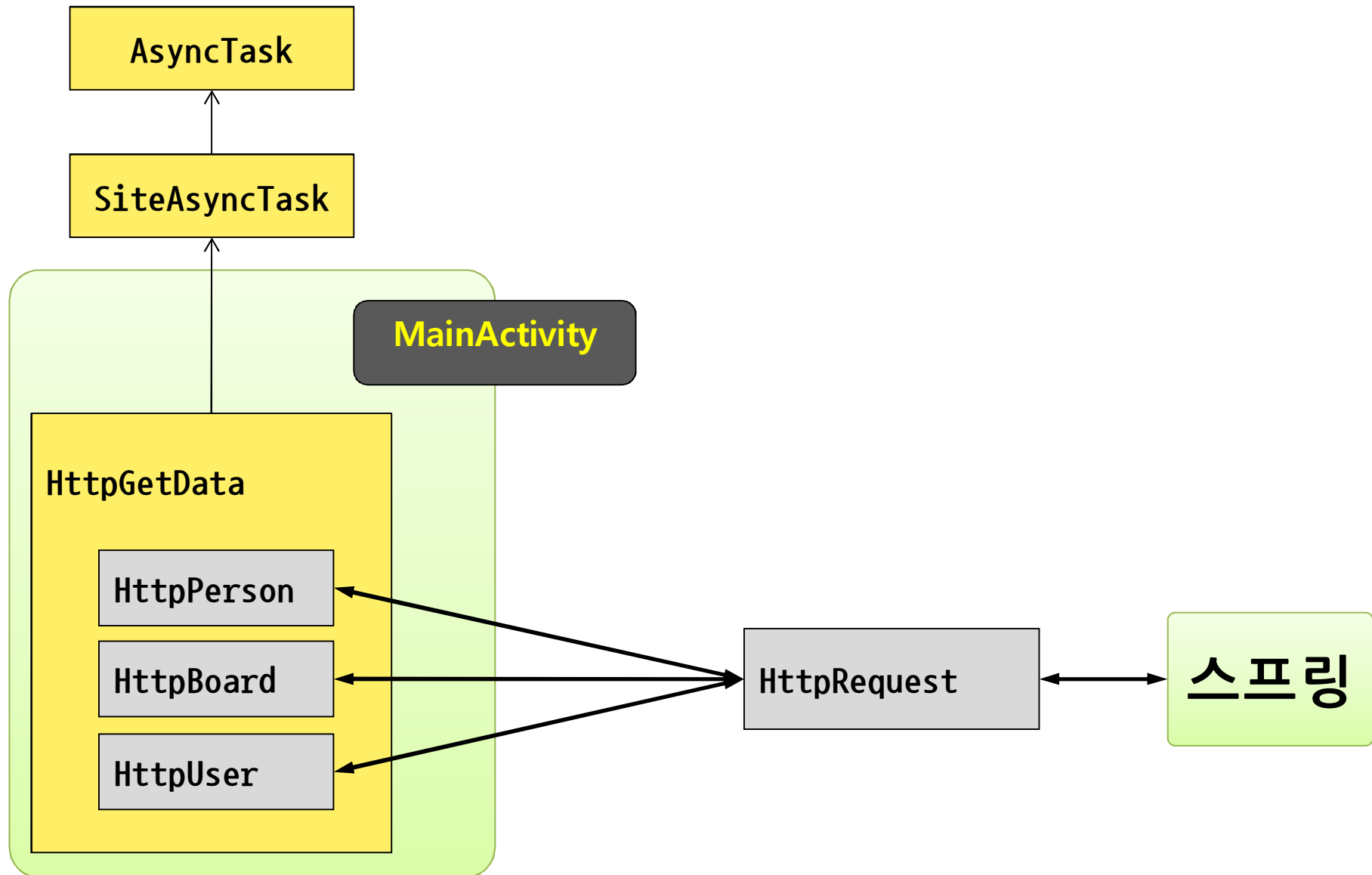
```
<uses-permission android:name="android.permission.INTERNET" />
```

- build.gradle 에 라이브러리 추가

```
android {  
    // ... 이어서  
    testOptions {  
        unitTests.returnDefaultValues = true  
    }  
}  
  
dependencies {  
    // ... 이어서  
    compile 'org.apache.commons:commons-lang3:3.4'  
    compile 'commons-codec:commons-codec:1.10'  
  
    compile 'org.json:json:20170516'  
    compile 'com.google.code.gson:gson:2.8.0'  
}
```

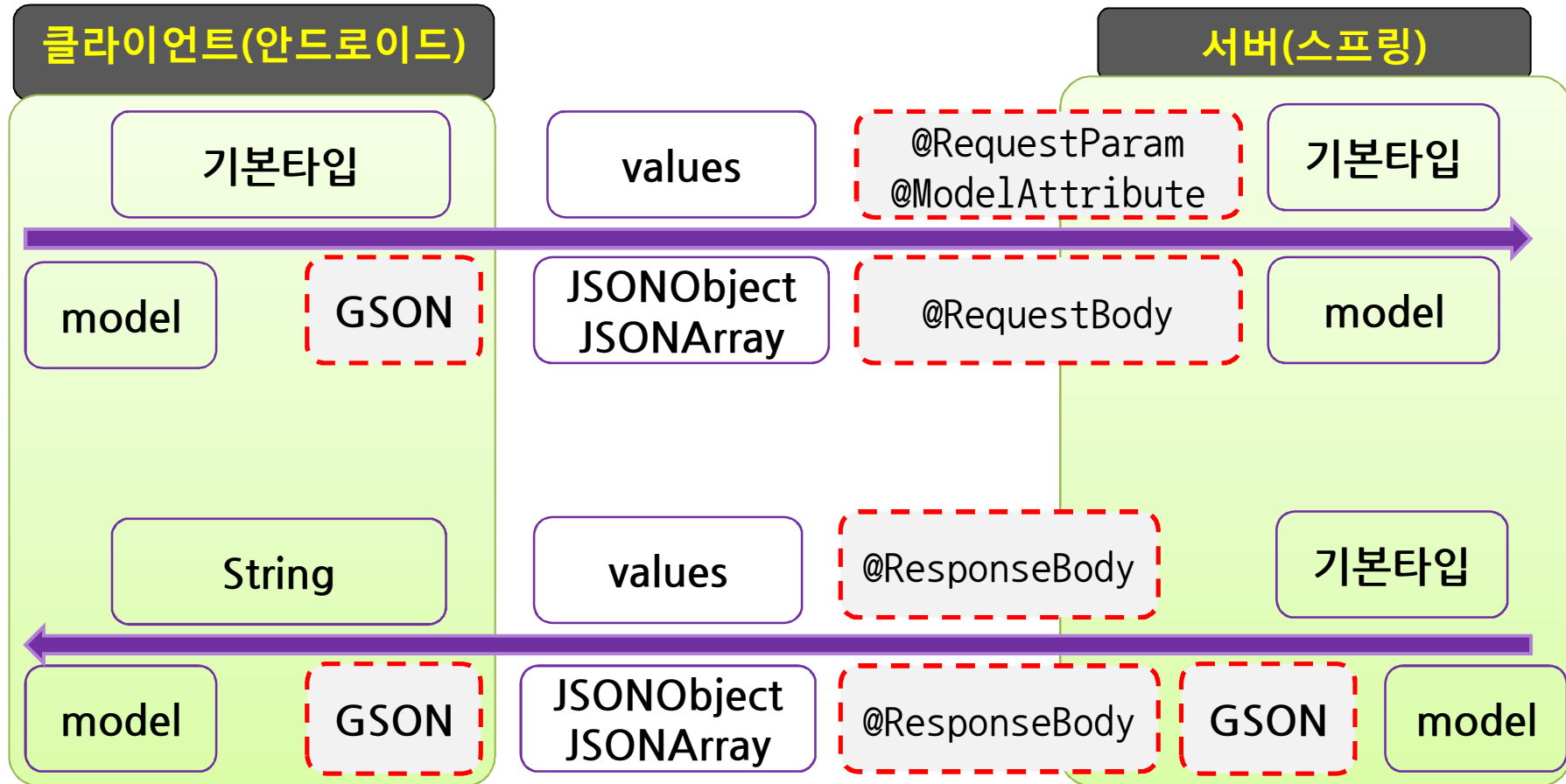


AVD Network Address Space





json 과 model간 변환



- GSON이란:
 - ORM 매핑 구글 라이브러리



JSON <--> 자바 Model 인스턴스 변환

- **Convert Java object to JSON : toJson()**

```
ModelPerson obj = new ModelPerson("valid", "valpw", "valname", "valemail");  
String jsonString = new Gson().toJson(obj);
```

- **java List 를 JSON으로 변환 : toJson()**

```
List<ModelPerson> list = new ArrayList<ModelPerson>();  
for( int i=0; i<10; i++){  
    String t = String.valueOf(i);  
    ModelPerson obj = new ModelPerson("id"+t, "pw"+t, "name"+t, "email"+t);  
    list.add(obj);  
}  
String data = new Gson().toJson(list);
```

- **Convert JSON to Java Object : fromJson()**

```
ModelPerson person = new Gson().fromJson(jsonInString, ModelPerson.class);
```

- **Convert a JSON Array to a java List, using TypeToken : fromJson()**

```
String json = "[{'name':'mkyong'}, {'name':'laplap'}]";  
List<ModelPerson> list = new Gson().fromJson(json, new  
    TypeToken<List<ModelPerson>>().getType());
```



AVD Network Address Space

- Each instance of the emulator runs behind your development machine's.
- An emulated device can not see your development machine or other emulator instances on the network.
- An emulated device sees only that it is connected through Ethernet to a router/firewall.
- The virtual router for each instance manages the 10.0.2/24 network
- Network Address Description
 - 10.0.2.1 Router/gateway address
 - 10.0.2.2 Special alias to your host loopback interface (i.e., 127.0.0.1 on your development machine)
 - 10.0.2.3 First DNS server
 - 10.0.2.4 / 10.0.2.5 / 10.0.2.6 Optional second, third and fourth DNS server (if any)
 - 10.0.2.15 The emulated device's own network/ethernet interface
 - 127.0.0.1 The emulated device's own loopback interface



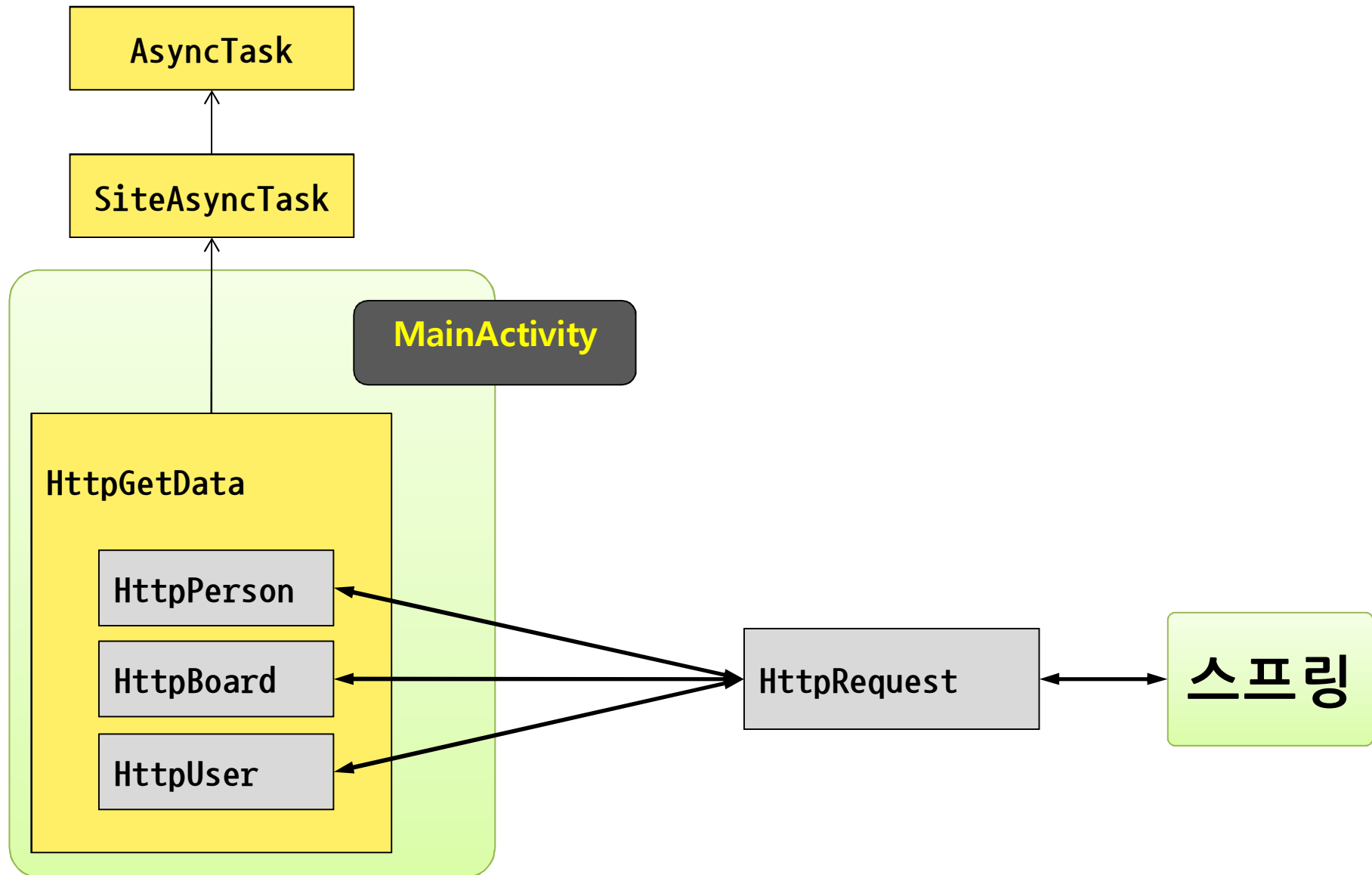
Http 통신을 위한 작업 순서

1. 스프링 프로젝트에서 REST 서비스 만들기
2. 스프링 프로젝트 실행

1. 안드로이드 프로젝트 생성
2. build.gradle 에 라이브러리 추가
`compile 'com.google.code.gson:gson:2.8.0'`
3. AndroidManifest.xml 에 권한 추가
`<uses-permission android:name="android.permission.INTERNET" />`
4. SiteAsyncTask 복사 <-- Http 통신하는 동안 시계 표시를 위한 클래스
5. HttpRequest 복사 <-- Http 통신을 위한 클래스
6. MainActivity 나 Fragment에서 Thread 생성을 위한 내부 클래스 추가
`private class HttpGetData extends SiteAsyncTask< ... > { }`
7. HttpPerson, HttpUser, HttpBoard 클래스 만들기
--> HttpGetData 에서 스프링 데이터를 요청하기 위해 사용된다.
8. 안드로이드 프로젝트 실행



AVD Network Address Space



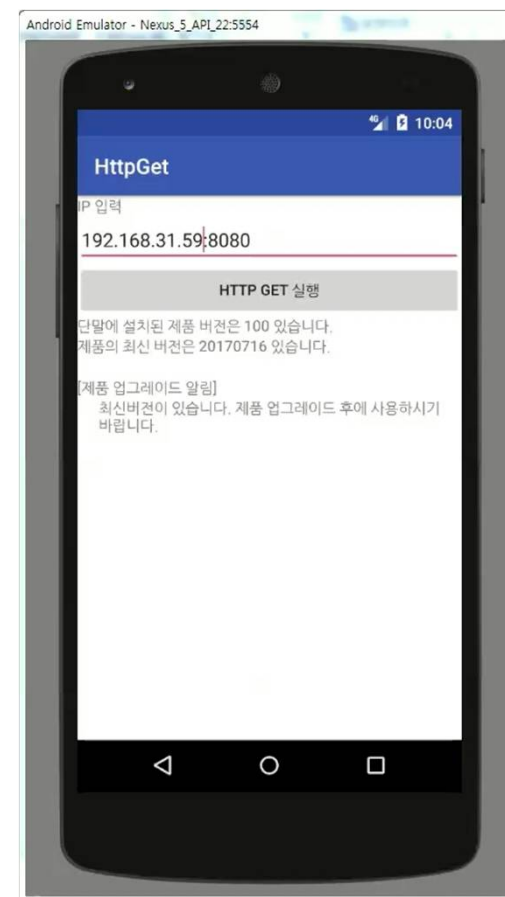
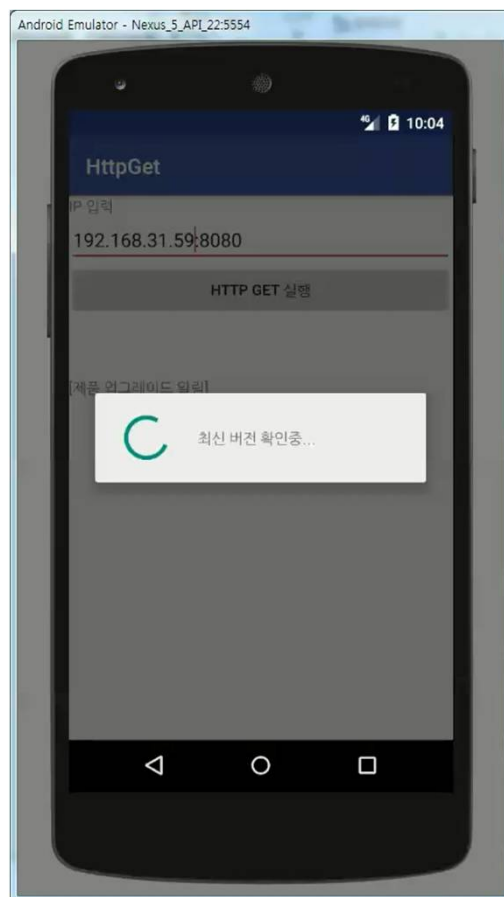


ad43-01

st1current version

GET 방식을 이용한 "제품 버전 확인"

```
<uses-permission android:name="android.permission.INTERNET" />
```





ad43-01

st3 insert person

POST를 이용한 사용자 추가.

Insert person

User ID:

Password:

Name:

Email:


Insert person

User ID:

Password:

Name:

Email:

 최신 버전 확인중...

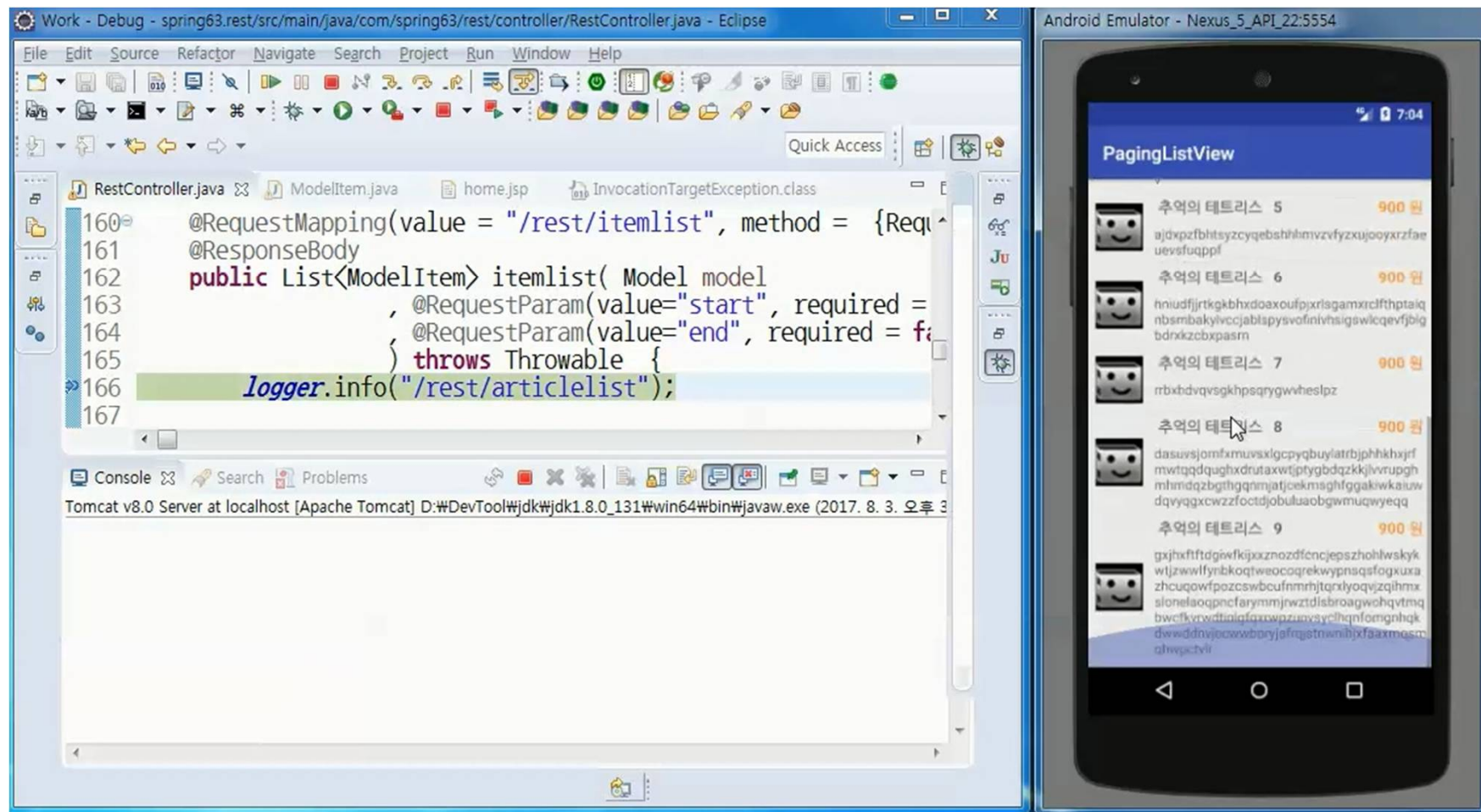


ad43-01

st4 Paging List View

리스트 뷰와 스프링 연동하기

1. build.gradle 에 dependency 추가
2. AndroidManifest.xml 권한 추가

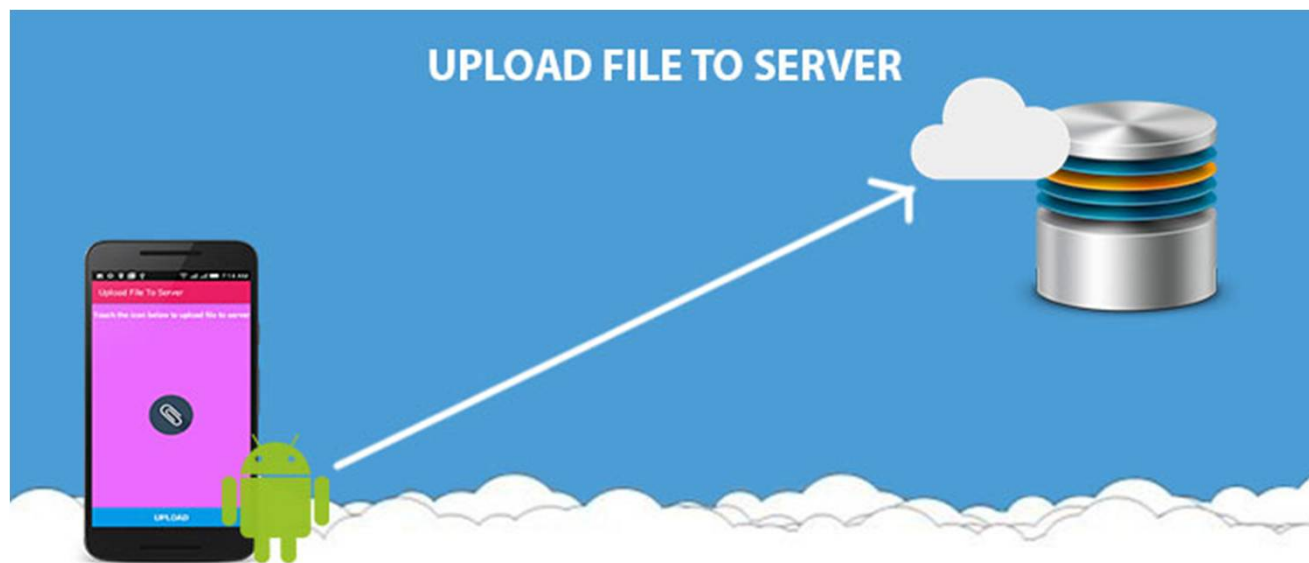




ad43-01

st5 file upload

Multipart form-data 를 이용하여 파일과 텍스트를 동시에 웹 서버로 전송하는 어플을 만들어 보자



<https://github.com/vamsitallapudi/AndroidUploadFileToServer>

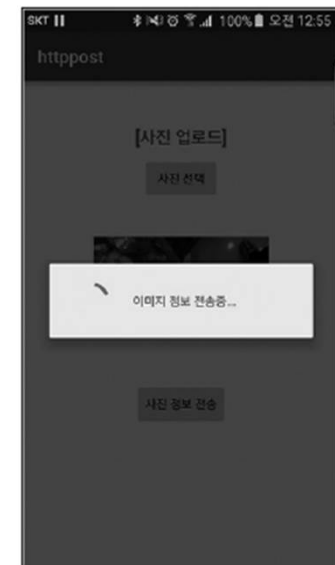
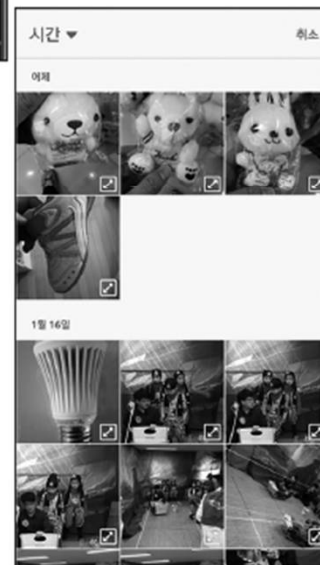
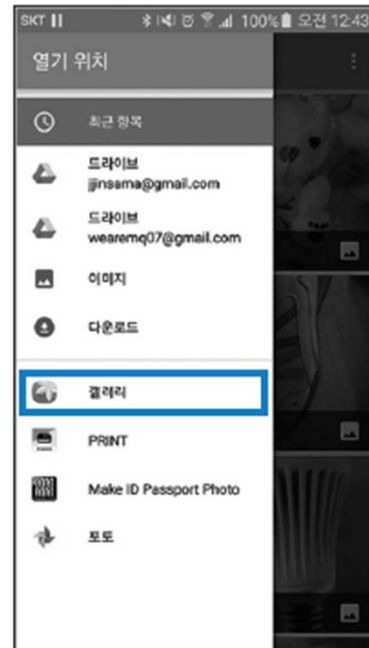
<https://github.com/gotev/android-upload-service>



ad43-01

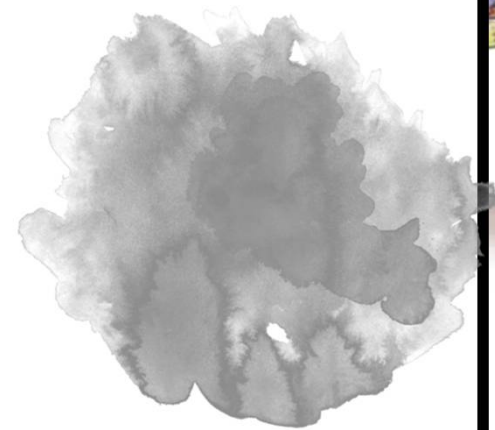
st6 image upload

POST를 이용한 "이미지 정보 전송"



이미지 로딩 라이브러리

- Universal Image Loader
- Picasso
- Glide
- Fresco





이미지 로딩 라이브러리 비교표

	(+)	(-)
Fresco	<ul style="list-style-type: none"> - Pretty fast image loader (for small && medium images) - A lot of functionality(streaming, drawing tools, memory management, etc) - Possibility to setup directly in xml (for example round corners) - GIF support - WebP support 	<ul style="list-style-type: none"> - Huge size of library - No Callback with View, Bitmap parameters - SimpleDraweeView doesn't support wrap_content - Huge size of cache
Glide ★	<ul style="list-style-type: none"> - Tinny size of cache - Simple in use - GIF support - WebP support - Fast loading big images from internet into ListView - UI is not freeze - BitmapPool to reuse memory and thus lesser GC events 	<ul style="list-style-type: none"> - Big size of library
	반복 스크롤 시 메모리를 많이 쓰는 만큼 빠른 속도와 안정성을 보장. 디스크 캐시에서 읽어오는 속도가 빨라 스크롤 시에도 이미지가 빠르게 뜬다.	
Picasso	<ul style="list-style-type: none"> - Tinny size of library - Small size of cache - Simple in use - UI is not freeze - WebP support 	<ul style="list-style-type: none"> - Slow loading big images from internet into ListView
Universal Image Loader	<ul style="list-style-type: none"> - Tinny size of library - Simple in use 	<ul style="list-style-type: none"> - Limited functionality(limited image processin) - GIF not supported - Project support has stopped since 27.11.2015
	메모리 캐시 히팅률이 낮음. 스크롤 시 이전 이미지가 보여지는 현상 발생.	



glide 사용법

- glide 사용법
 - Glide 클래스는 빌더 패턴으로 구현되어 있고, 3개의 필수 파라미터를 요구한다.
 - with(Context context) : 안드로이드의 많은 API를 이용하기 위해 필요
 - load(String imageUrl) : 이미지 경로
 - into(ImageView targetImageView) : 다운 받은 이미지를 보여줄 이미지 뷰

```
// 웹 URL
ImageView target = (ImageView) findViewById(R.id.imageview);
String url = "http://www.example.com/icon.png";
Glide.with(context).load(url).into(target);
```

```
// 리소스 ID
int resourceId = R.mipmap.ic_launcher;
Glide.with(context).load(resourceId).into(target);
```

```
// 로컬 파일
File file = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES),
"Example.jpg");
Glide.with(context).load(file).into(target);
```