챗봇 과 웹 서비스

챗봇

❖ 챗봇 데이터

✓ Korpora(Korean Corpora Archives)는 한국어 데이터셋이 있는 오픈소스 파이썬 패키지로 챗봇, 댓글, 혐오. 국민 청원, 위키, 신문, 메신저 등 다양한 한글 말뭉치가 있음

말뭉치 이름	설명	링크
korean_chatbot_data	챗봇 트레이닝용 문답 페어	https://github.com/songys/Chatbot_data
kcbert	KcBERT 모델 훈련용 댓글 데이터	https://github.com/Beomi/KcBERT
korean_hate_speech	한국어 혐오 데이터셋	https://github.com/kocohub/korean-hate-speech
korean_petitions	청와대 국민 청원	https://github.com/lovit/petitions_archive
kornli	Korean NLI	https://github.com/kakaobrain/KorNLUDatasets
korsts	Korean STS	https://github.com/kakaobrain/KorNLUDatasets
namuwikitext	나무위키 텍스트	https://github.com/lovit/namuwikitext
naver_changwon_ner	네이버 × 창원대 개체명 인식 데이터셋	https://github.com/naver/nlp-challenge/tree/master/missions/ner
nsmc	NAVER Sentiment Movie Corpus	https://github.com/e9t/nsmc
question_pair	한국어 질문쌍 데이터셋	https://github.com/songys/Question_pair
modu_news	모두의 말뭉치: 신문	https://corpus.korean.go.kr
modu_messenger	모두의 말뭉치: 메신저	https://corpus.korean.go.kr
modu_mp	모두의 말뭉치: 형태 분석	https://corpus.korean.go.kr
modu_ne	모두의 말뭉치: 개체명 분석	https://corpus.korean.go.kr
modu_spoken	모두의 말뭉치: 구어	https://corpus.korean.go.kr
modu_web	모두의 말뭉치: 웹	https://corpus.korean.go.kr
modu_written	모두의 말뭉치: 문어	https://corpus.korean.go.kr
aihub_translation	한국어-영어 번역 말뭉치	https://aihub.or.kr/aidata/87
open_substitles	영화 자막 한영 병렬 말뭉치	http://opus.nlpl.eu/OpenSubtitles-v2018.php
korean_parallel_koen_news	한국어-영어 병렬 말뭉치	https://github.com/jungyeul/korean-parallel-corpora

챗봇

- 챗봇 데이터
  - ✓ 데이터 구조

    ,Q,A,label

    0,12시 땡!,하루가 또 가네요.,0

    1,1지망 학교 떨어졌어,위로해 드립니다.,0

    - ❑ 일련번호 – 없어도 가능
    - ❑ 질문
    - ❑ 답변
    - ❑ 레이블 – 없어도 가능

챗봇

- 챗봇 데이터
  - ✓ 훈련 시 주의 사항
    - ❑ 전체 데이터셋 크기는 11,823쌍인데 구글 코랩 무료 환경을 기준으로 1.1만 개의 쌍을 훈련하게 되면 다음과 같은 메시지가 나타나면서 코랩이 재시작 될 수 있는데 그림의 메시지는 램 용량 초과로 나타나는 현상으로 큰 데이터셋을 다룰 때 종종 나타남

    - ❑ 무료로 제공되는 RAM 용량은 12G
    - ❑ 이 문제를 해결하기 위해는 코랩 Pro 버전(25G, 유료)을 사용하거나 로컬 환경에서 하는 방법이 있음
    - ❑ 로컬에 파일을 저장하고 하는 경우도 메모리 크기에 유의

챗봇

❖ 챗봇 구현

✓ 데이터를 메모리로 가져오기

```
import pandas as pd

#corpus =
pd.read_csv('https://raw.githubusercontent.com/songys/Chatbot_data/master/ChatbotData.csv')
corpus = pd.read_csv('./ChatbotData.csv')

# 2,000개 데이터 셋만 활용 (Google Colab 일 경우 3,000개에서는 메모리 오버되는 현상 발생)
texts = []
pairs = []
for i, (text, pair) in enumerate(zip(corpus['Q'], corpus['A'])):
    texts.append(text)
    pairs.append(pair)

    #메모리가 부족하면 데이터 개수 조절
    if i >= 2000:
        break
```

챗봇

❖ 챗봇 구현

✓ 데이터 확인

```
# question와 answer 데이터 확인
list(zip(texts, pairs))[1995:2000]
```

```
[('배 아프다', '약이 필요하면 도움을 받아보세요.'),
 ('배 터지겠네', '위를 좀 쉬게 해주세요.'),
 ('배 터지겠다.', '산책 좀 해야겠네여.'),
 ('배가 너무 고파', '뭐 좀 챙겨드세요.'),
 ('배가 넘넘 고파', '저도 밥 먹고 싶어요')]
```

챗봇

❖ 챗봇 구현

✓ 데이터 전처리

```
import re
def clean_sentence(sentence):
    # 한글, 숫자를 제외한 모든 문자는 제거합니다.
    sentence = re.sub(r'[^0-9ㄱ-ㅎㅏ-ㅣ가-힣 ]',r'', sentence)
    return sentence

# 전처리 함수 테스트
print(clean_sentence('안녕하세요~:)'))
print(clean_sentence('TensorFlow^@^%#@!'))
```

```
안녕하세요
TensorFlow
```

챗봇

- 챗봇 구현
  - ✓ 데이터 전처리

```
#한글 형태소 분석
from konlpy.tag import Okt
okt = Okt()
def process_morph(sentence):
    return ' '.join(okt.morphs(sentence))
```

챗봇

- 챗봇 구현
  - ✓ 데이터 전처리
    - ❑ 챗봇 모델을 훈련하기 위해서 필요한 3가지 데이터
      - question : 인코더에 입력할 데이터 (질문 전체)
      - answer_input : 디코더에 입력할 데이터(답변의 시작)로 <START> 토큰을 문장 처음에 추가
      - answer_output : 디코더의 출력 데이터(답변의 끝)로 〈END〉토큰을 문장 마지막에 추가

챗봇

- 챗봇 구현
  - ✓ 데이터 전처리

```
#챗봇을 위한 데이터 생성
def clean_and_morph(sentence, is_question=True):
    # 한글 문장 전처리
    sentence = clean_sentence(sentence)
    # 형태소 변환
    sentence = process_morph(sentence)
    # Question 인 경우, Answer인 경우를 분기하여 처리
    # Answer에는 시작 과 종료 기호 추가
    if is_question:
        return sentence
    else:
        # START 토큰은 decoder input에 END 토큰은 decoder output에 추가합니다.
        return ('<START> ' + sentence, sentence + ' <END>')
```

챗봇

❖ 챗봇 구현

✓ 데이터 전처리

```
#챗봇을 위한 데이터 생성
def preprocess(texts, pairs):
    questions = []
    answer_in = []
    answer_out = []

    # 질의에 대한 전처리
    for text in texts:
        # 전처리와 morph 수행
        question = clean_and_morph(text, is_question=True)
        questions.append(question)

    # 답변에 대한 전처리
    for pair in pairs:
        # 전처리와 morph 수행
        in_, out_ = clean_and_morph(pair, is_question=False)
        answer_in.append(in_)
        answer_out.append(out_)

    return questions, answer_in, answer_out
```

챗봇

❖ 챗봇 구현

✓ 데이터 전처리

```
#챗봇 훈련에 필요한 데이터 생성 및 확인
questions, answer_in, answer_out = preprocess(texts, pairs)

print(questions[:2])
print(answer_in[:2])
print(answer_out[:2])
```

```
['12시 땡', '1 지망 학교 떨어졌어']
['<START> 하루 가 또 가네요', '<START> 위로 해 드립니다']
['하루 가 또 가네요 <END>', '위로 해 드립니다 <END>']
```

챗봇

❖ 챗봇 구현

✓ 데이터 전처리

```
# 전체 문장을 하나의 문장으로 생성
all_sentences = questions + answer_in + answer_out
```

챗봇

- 챗봇 구현
  - ✓ 데이터 전처리

```
#토크나이저 와 수치화 및 패딩
import numpy as np
import warnings
import tensorflow as tf

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# WARNING 무시
warnings.filterwarnings('ignore')
```

챗봇

❖ 챗봇 구현

✓ 데이터 전처리

```
# 토큰화
tokenizer = Tokenizer(filters='', lower=False, oov_token='<OOV>')
tokenizer.fit_on_texts(all_sentences)
```

챗봇

- 챗봇 구현
  - ✓ 데이터 전처리

```
# 텍스트를 시퀀스로 인코딩 (texts_to_sequences)
question_sequence = tokenizer.texts_to_sequences(questions)
answer_in_sequence = tokenizer.texts_to_sequences(answer_in)
answer_out_sequence = tokenizer.texts_to_sequences(answer_out)
```

챗봇

- 챗봇 구현
  - ✓ 데이터 전처리

```
# 문장의 길이 맞추기 (pad_sequences)
MAX_LENGTH = 30
question_padded = pad_sequences(question_sequence,
                                maxlen=MAX_LENGTH,
                                truncating='post',
                                padding='post')
answer_in_padded = pad_sequences(answer_in_sequence,
                                 maxlen=MAX_LENGTH,
                                 truncating='post',
                                 padding='post')
answer_out_padded = pad_sequences(answer_out_sequence,
                                  maxlen=MAX_LENGTH,
                                  truncating='post',
                                  padding='post')
```

챗봇

❖ 챗봇 구현

✓ 데이터 전처리

```
# 단어 사전 확인
for word, idx in tokenizer.word_index.items():
    print(f'{word}₩t -> ₩t{idx}')
    if idx > 10:
        break
```

```
<OOV>    ->    1
<START>  ->    2
<END>    ->    3
이       ->    4
을       ->    5
거       ->    6
가       ->    7
예요     ->    8
사람     ->    9
요       ->    10
에       ->    11
```

챗봇

❖ 챗봇 구현

✓ 데이터 전처리

```
# 토큰 개수 확인
len(tokenizer.word_index)
```

12637

챗봇

❖ 챗봇 구현

✓ 데이터 전처리

#시퀀스 확인
question_padded.shape, answer_in_padded.shape, answer_out_padded.shape

((11823, 30), (11823, 30), (11823, 30))

챗봇

❖ 챗봇 구현

✓ 데이터 전처리

```
#원핫 인코딩
VOCAB_SIZE = len(tokenizer.word_index)+1

#원핫 인코딩을 위한 함수
def convert_to_one_hot(padded):
    # 원핫인코딩 초기화
    one_hot_vector = np.zeros((len(answer_out_padded),
                               MAX_LENGTH,
                               VOCAB_SIZE))

    # 학습시 입력은 인덱스이지만 출력은 원핫 인코딩 형식임
    for i, sequence in enumerate(answer_out_padded):
        for j, index in enumerate(sequence):
            one_hot_vector[i, j, index] = 1

    return one_hot_vector

answer_in_one_hot = convert_to_one_hot(answer_in_padded)
answer_out_one_hot = convert_to_one_hot(answer_out_padded)
answer_in_one_hot[0].shape, answer_in_one_hot[0].shape
```

((30, 3605), (30, 3605))

챗봇

❖ 챗봇 구현

✓ 데이터 전처리

```
# 변환된 index를 다시 단어로 변환
def convert_index_to_text(indexs, end_token):

    sentence = ''

    # 모든 문장에 대해서 반복
    for index in indexs:
        if index == end_token:
            # 끝 단어이므로 예측 중비
            break;
        # 사전에 존재하는 단어의 경우 단어 추가
        if index > 0 and tokenizer.index_word[index] is not None:
            sentence += tokenizer.index_word[index]
        else:
        # 사전에 없는 인덱스면 빈 문자열 추가
            sentence += ''

        # 빈칸 추가
        sentence += ' '
    return sentence
```

챗봇

❖ 챗봇 구현

✓ 모델 생성을 위한 클래스 구현

```
# 라이브러리 로드
from tensorflow.keras.layers import Input, Embedding, LSTM, Dense, Dropout, Attention
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import TensorBoard, ModelCheckpoint
from tensorflow.keras.utils import plot_model
```

챗봇

❖ 챗봇 구현

✓ 모델 생성을 위한 클래스(Encoder) - 어텐션 가중치를 구하기 위해서 return_sequences = True 옵션을 주가 설정

```
class Encoder(tf.keras.Model):
    def __init__(self, units, vocab_size, embedding_dim, time_steps):
        super(Encoder, self).__init__()
        self.embedding = Embedding(vocab_size,
                                   embedding_dim,
                                   input_length=time_steps,
                                   name='Embedding')
        self.dropout = Dropout(0.2, name='Dropout')
        # (attention) return_sequences=True 추가
        self.lstm = LSTM(units,
                         return_state=True,
                         return_sequences=True,
                         name='LSTM')

    def call(self, inputs):
        x = self.embedding(inputs)
        x = self.dropout(x)
        x, hidden_state, cell_state = self.lstm(x)
        # (attention) x return 추가
        return x, [hidden_state, cell_state]
```

챗봇

❖ 챗봇 구현

✓ 모델 생성을 위한 클래스(Decoder) - 어텐션 레이어를 추가해 어텐션 매커니즘을 구현

```
class ChatModel(tf.keras.Model):
    def __init__(self, units, vocab_size, embedding_dim, time_steps, start_token,
end_token):
        super(ChatModel, self).__init__()
        self.start_token = start_token
        self.end_token = end_token
        self.time_steps = time_steps

        self.encoder = Encoder(units, vocab_size, embedding_dim, time_steps)
        self.decoder = Decoder(units, vocab_size, embedding_dim, time_steps)
```

챗봇

❖ 챗봇 구현

✓ 모델 생성을 위한 클래스(Decoder) - 어텐션 레이어를 추가해 어텐션 매커니즘을 구현

```
def call(self, inputs, training=True):
    if training:
        encoder_inputs, decoder_inputs = inputs
        # (attention) encoder 출력 값 수정
        encoder_outputs, context_vector = self.encoder(encoder_inputs)
        # (attention) decoder 입력 값 수정
        decoder_outputs, _, _ = self.decoder((encoder_outputs, decoder_inputs),
                                             initial_state=context_vector)
        return decoder_outputs
    else:
        x = inputs
        # (attention) encoder 출력 값 수정
        encoder_outputs, context_vector = self.encoder(x)
        target_seq = tf.constant([[self.start_token]], dtype=tf.float32)
        results = tf.TensorArray(tf.int32, self.time_steps)
```

챗봇

❖ 챗봇 구현

✓ 모델 생성을 위한 클래스(Decoder) - 어텐션 레이어를 추가해 어텐션 매커니즘을 구현

```
        for i in tf.range(self.time_steps):
            decoder_output, decoder_hidden, decoder_cell =
self.decoder((encoder_outputs, target_seq),
                                                    initial_state=context_vector)
            decoder_output = tf.cast(tf.argmax(decoder_output, axis=-1), dtype=tf.int32)
            decoder_output = tf.reshape(decoder_output, shape=(1, 1))
            results = results.write(i, decoder_output)

            if decoder_output == self.end_token:
                break

            target_seq = decoder_output
            context_vector = [decoder_hidden, decoder_cell]

        return tf.reshape(results.stack(), shape=(1, self.time_steps))
```

챗봇

❖ 챗봇 구현

✓ 모델 생성을 위한 클래스(Model)

```
class ChatModel(tf.keras.Model):
    def __init__(self, units, vocab_size, embedding_dim, time_steps, start_token,
end_token):
        super(ChatModel, self).__init__()
        self.start_token = start_token
        self.end_token = end_token
        self.time_steps = time_steps

        self.encoder = Encoder(units, vocab_size, embedding_dim, time_steps)
        self.decoder = Decoder(units, vocab_size, embedding_dim, time_steps)
```

챗봇

❖ 챗봇 구현

✓ 모델 생성을 위한 클래스(Model)

```
def call(self, inputs, training=True):
    if training:
        encoder_inputs, decoder_inputs = inputs
        # (attention) encoder 출력 값 수정
        encoder_outputs, context_vector = self.encoder(encoder_inputs)
        # (attention) decoder 입력 값 수정
        decoder_outputs, _, _ = self.decoder((encoder_outputs, decoder_inputs),
                                             initial_state=context_vector)
        return decoder_outputs
```

챗봇

❖ 챗봇 구현

✓ 모델 생성을 위한 클래스(Model)

```
        else:
            x = inputs
            # (attention) encoder 출력 값 수정
            encoder_outputs, context_vector = self.encoder(x)
            target_seq = tf.constant([[self.start_token]], dtype=tf.float32)
            results = tf.TensorArray(tf.int32, self.time_steps)

            for i in tf.range(self.time_steps):
                decoder_output, decoder_hidden, decoder_cell =
self.decoder((encoder_outputs, target_seq),
                                                    initial_state=context_vector)
                decoder_output = tf.cast(tf.argmax(decoder_output, axis=-1), dtype=tf.int32)
                decoder_output = tf.reshape(decoder_output, shape=(1, 1))
                results = results.write(i, decoder_output)

                if decoder_output == self.end_token:
                    break

                target_seq = decoder_output
                context_vector = [decoder_hidden, decoder_cell]

            return tf.reshape(results.stack(), shape=(1, self.time_steps))
```

챗봇

❖ 챗봇 구현

✓ 모델 생성

```
# 하이퍼 파라미터 설정
BUFFER_SIZE = 1000
BATCH_SIZE = 16
EMBEDDING_DIM = 100
TIME_STEPS = MAX_LENGTH
START_TOKEN = tokenizer.word_index['<START>']
END_TOKEN = tokenizer.word_index['<END>']

UNITS = 128

VOCAB_SIZE = len(tokenizer.word_index)+1
DATA_LENGTH = len(questions)
SAMPLE_SIZE = 3
NUM_EPOCHS = 20
```

챗봇

❖ 챗봇 구현

✓ 모델 생성

```
# 체크 포인트 생성
checkpoint_path = 'model/chatmodel-attention-checkpoint.ckpt'
checkpoint = ModelCheckpoint(filepath=checkpoint_path,
                             save_weights_only=True,
                             save_best_only=True,
                             monitor='loss',
                             verbose=1
                            )
```

챗봇

❖ 챗봇 구현

✓ 모델 생성

```
#모델 생성
chat = ChatModel(UNITS,
                 VOCAB_SIZE,
                 EMBEDDING_DIM,
                 TIME_STEPS,
                 START_TOKEN,
                 END_TOKEN)

chat.compile(optimizer='adam',
             loss='categorical_crossentropy',
             metrics=['acc'])
```

챗봇

❖ 챗봇 구현

✓ 모델 생성

```
# 예측을 위한 함수
def make_prediction(model, question_inputs):
    results = model(inputs=question_inputs, training=False)
    # 변환된 인덱스를 문장으로 변환
    results = np.asarray(results).reshape(-1)
    return results
```

챗봇

❖ 챗봇 구현

✓ 모델 학습

```
for epoch in range(NUM_EPOCHS):
    print(f'processing epoch: {epoch * 10 + 1}...')
    chat.fit([question_padded, answer_in_padded],
             answer_out_one_hot,
             epochs=10,
             batch_size=BATCH_SIZE,
             callbacks=[checkpoint]
            )
    # 랜덤한 샘플 번호 추출
    samples = np.random.randint(DATA_LENGTH, size=SAMPLE_SIZE)
```

챗봇

❖ 챗봇 구현

✓ 모델 학습

```
# 예측 성능 테스트
for idx in samples:
    question_inputs = question_padded[idx]
    # 문장 예측
    results = make_prediction(chat, np.expand_dims(question_inputs, 0))

    # 변환된 인덱스를 문장으로 변환
    results = convert_index_to_text(results, END_TOKEN)

    print(f'Q: {questions[idx]}')
    print(f'A: {results}₩n')
    print()
```

챗봇

❖ 챗봇 배포

✓ 자연어 (질문 입력) 대한 전처리 함수

```
def make_question(sentence):
    sentence = clean_and_morph(sentence)
    question_sequence = tokenizer.texts_to_sequences([sentence])
    question_padded = pad_sequences(question_sequence, maxlen=MAX_LENGTH,
truncating='post', padding='post')
    return question_padded

make_question('오늘 날씨 어때?')
```

챗봇

❖ 챗봇 배포

✓ 질문에 대한 답변을 리턴해주는 함수

```
def run_chatbot(question):
    question_inputs = make_question(question)
    results = make_prediction(seq2seq, question_inputs)
    results = convert_index_to_text(results, END_TOKEN)
    return results
```

챗봇

❖ 챗봇 배포

✓ 콘솔에서 실행

```
while True:
    user_input = input('<< 말을 걸어 보세요!₩n')
    if user_input == 'q':
        break
    print('>> 챗봇 응답: {}'.format(run_chatbot(user_input)))
```

```
<< 말을 걸어 보세요!
밥은 먹었니
>> 챗봇 응답: 배고프지 않아요

<< 말을 걸어 보세요!
```

챗봇

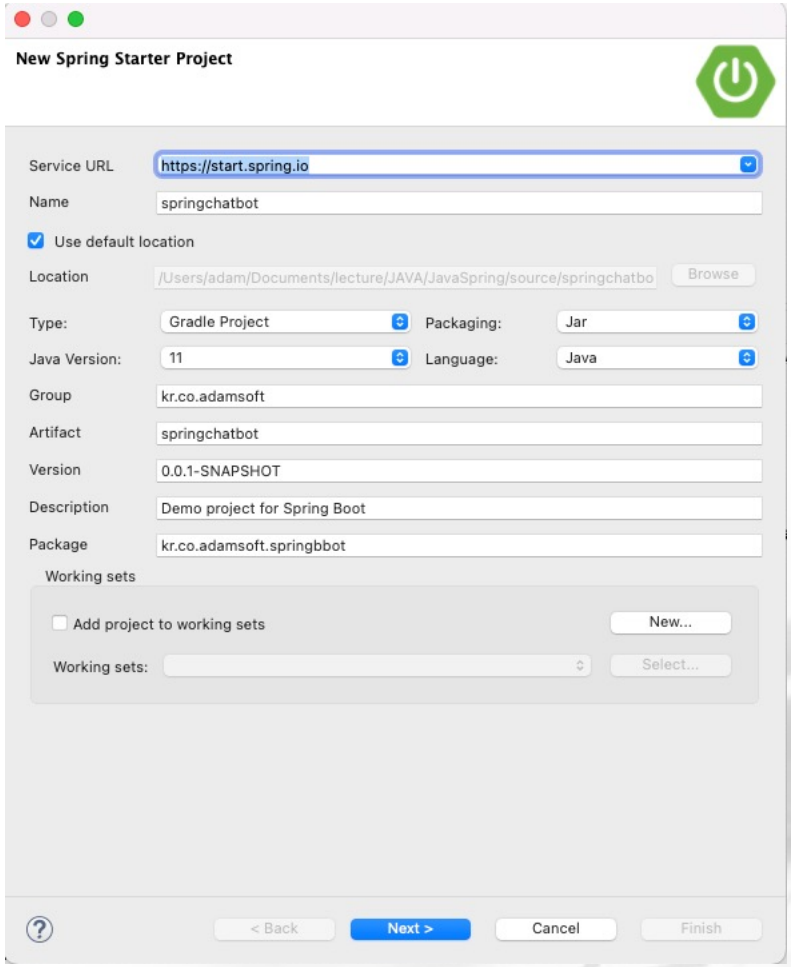
❖ 챗봇 배포

✓ 웹 서비스를 이용한 실행

```
# 웹 서비스를 위한 배포
from flask import Flask, request
from flask import jsonify

app = Flask(__name__)

#시작 요청이 왔을 때 아래 코드를 수행
@app.route('/', methods=['POST', 'GET'])
def main():
    return 'Hello Chatbot'

@app.route('/chatbot', methods=['POST', 'GET'])
def chatbot():
    print(request.args["question"])
    answer = run_chatbot(request.args["question"])
    print('>> 챗봇 응답: {}'.format(answer))
    response = {'answer': answer}
    return jsonify(response)

#서버 구동
app.run('0.0.0.0', port=9000, threaded=True)
```

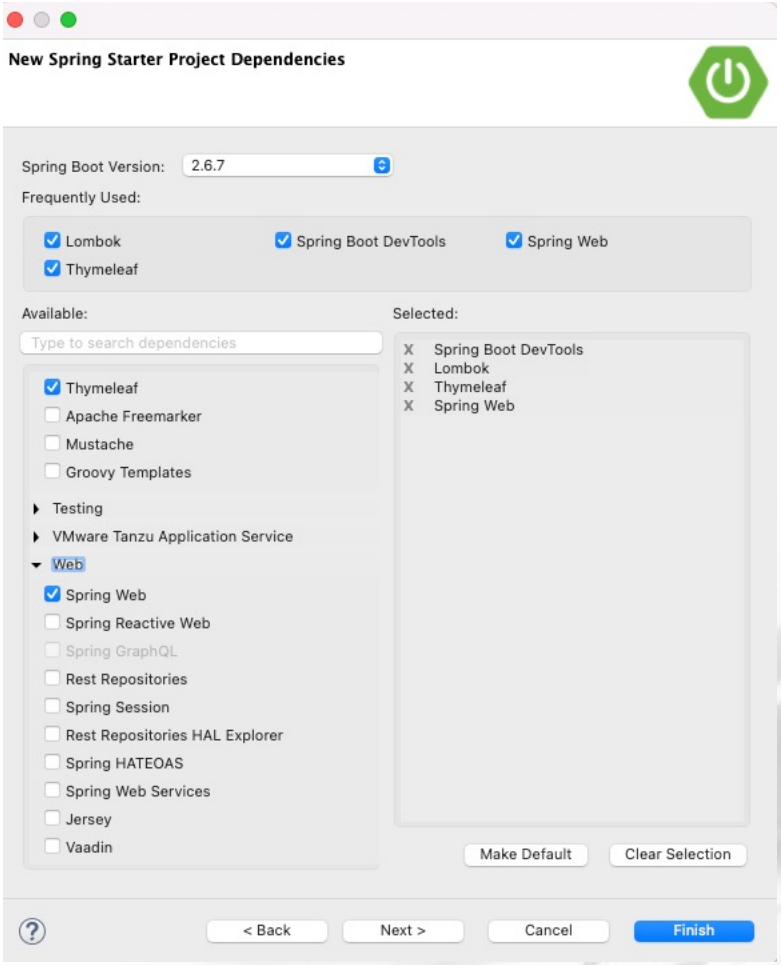
챗봇

- 챗봇 사용을 위한 Spring Boot Project
  - ✓ Spring Boot Project 생성



챗봇

- 챗봇 사용을 위한 Spring Boot Project
  - ✓ Spring Boot Project 생성



챗봇

❖ 챗봇 사용을 위한 Spring Boot Project

✓ 기본 패키지에 페이지 이동을 위한 Controller 생성

```
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class PageController {
        @GetMapping("/")
        public String main(Model model){
                return "index";
        }
}
```

챗봇

- 챗봇 사용을 위한 Spring Boot Project
  - ✓ src/main/resource 디렉토리의 template 디렉토리에 index.html 파일을 생성하고 작성

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>챗봇</title>
</head>
<body>
		<h3>질문 입력 영역</h3>
		<p>
			<!-- <a
href='http://localhost:9000/chatbot?questions=12%EC%8B%9C'>링크 이용</a>  -->
			<a
href='http://192.168.20.105:9000/chatbot?question=12%EC%8B%9C'>링크 이용</a>
		</p>
		<p>
			ajax 이용
			<input type="text" id="question">
			<input type="button" id="chatbotbtn" value="챗본 전송" />
		</p>
</body>
```

챗봇

❖ 챗봇 사용을 위한 Spring Boot Project

✓ src/main/resource 디렉토리의 template 디렉토리에 index.html 파일을 생성하고 작성

```
<script>
        window.addEventListener("load", function(e){
                var question = document.getElementById("question");
                var chatbotbtn =  document.getElementById("chatbotbtn");
                chatbotbtn.addEventListener('click', function(e){
                        var request;
                        request = new XMLHttpRequest();
                    var queryString = "chatbot?question=" + question.value;

                        request.open('GET', queryString, true);
                        request.send('');

                        request.addEventListener('load', function() {

        alert(JSON.parse(request.responseText).answer);
                        });

                });
        });
</script>
</html>
```

챗봇

❖ 챗봇 사용을 위한 Spring Boot Project

✓ 패키지에 ajax 요청을 proxy 형태로 처리해 줄 서비스 인터페이스 생성

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public interface ChatBotService {
        public String download(HttpServletRequest request, HttpServletResponse
response);

}
```

챗봇

❖ 챗봇 사용을 위한 Spring Boot Project

✓ 패키지에 ajax 요청을 proxy 형태로 처리해 줄 서비스 클래스 생성

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Service;
```

챗봇

❖ 챗봇 사용을 위한 Spring Boot Project

✓ 패키지에 ajax 요청을 proxy 형태로 처리해 줄 서비스 클래스 생성

```
@Service
public class ChatBotServiceImpl implements ChatBotService {

		@Override
		public String download(HttpServletRequest request, HttpServletResponse
response) {
			StringBuilder sb = new StringBuilder();
			try {
				String question =
URLEncoder.encode(request.getParameter("question"), "UTF-8");

				String addr =
"http://192.168.20.105:9000/chatbot?question=" + question;
				URL url = new URL(addr);
				HttpURLConnection conn = (HttpURLConnection)
url.openConnection();
```

챗봇

❖ 챗봇 사용을 위한 Spring Boot Project

✓ 패키지에 ajax 요청을 proxy 형태로 처리해 줄 서비스 클래스 생성

```
                    if (conn != null) {
                        conn.setConnectTimeout(20000);
                        conn.setUseCaches(false);
                        if (conn.getResponseCode() ==
HttpURLConnection.HTTP_OK) {
                            InputStreamReader isr = new
InputStreamReader(conn.getInputStream());
                            BufferedReader br = new
BufferedReader(isr);
                            while (true) {
                                String line = br.readLine();
                                if (line == null) {
                                    break;
                                }
                                sb.append(line + "₩n");
                            }
                            br.close();
                            conn.disconnect();
                        }
                    }
                }
```

챗봇

❖ 챗봇 사용을 위한 Spring Boot Project

✓ 패키지에 ajax 요청을 proxy 형태로 처리해 줄 서비스 클래스 생성

```
                catch (Exception e) {
                        System.out.println("가져오기 실패:" + e.getMessage());
                }
                return sb.toString();
        }
}
```

챗봇

❖ 챗봇 사용을 위한 Spring Boot Project

✓ 패키지에 ajax 요청을 처리해줄 Controller 클래스 생성하고 작성

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class JsonController {
		@Autowired
		private ChatBotService chatBotService;

		@GetMapping("/chatbot")
		public String chatbot(HttpServletRequest request, HttpServletResponse
response){
			String result = chatBotService.download(request, response);
			return result;
		}

}
```

챗봇

❖ 챗봇 사용을 위한 Spring Boot Project

✓ 프로젝트 실행 후 localhost:8080 에서 버튼 클릭