

Task Description

1、Background

When you submit code changes to a code repository such as GitHub using a version control system such as Git, it gets organized into a commit with a commit message (usually a sentence or two that explains the content and intent of the commit).

After the code is successfully submitted to GitHub, you can see the commit details in your GitHub repo. In particular, diff refers to a change between a set of files or code. In GitHub, when you view the difference between two different commits, you will see the diff showing the file changes. diff shows, in rows, which rows have been added, removed, or modified.

Here's a concrete example to help you understand:

```
✓ Fix DefaultDataBuffer#getNativeBuffer() to set correct limit: → commit message
Closes gh-30967

injae-kim committed 5 days ago commit d357085237c91e1511ed44a6f19404a08454d249

spring-core/src/main/java/org/springframework/core/io/buffer/DefaultDataBuffer.java diff

@@ -37,6 +37,7 @@
 37 37 * @author Arjen Poutsma
 38 38 * @author Juergen Hoeller
 39 39 * @author Brian Clozel
 40 + * @author Injae Kim
 40 41 * @since 5.0
 41 42 * @see DefaultDataBufferFactory
 42 43 */

@@ -81,12 +82,12 @@ static DefaultDataBuffer fromEmptyByteBuffer(DefaultDataBufferFactory dataBuffer
 81 82 /**
 82 83 * Directly exposes the native {@code ByteBuffer} that this buffer is based
 83 84 * on also updating the {@code ByteBuffer's} position and limit to match
 84 - * the current {@link #readPosition()} and {@link #readableByteCount()}.
 85 + * the current {@link #readPosition()} and {@link #writePosition()}.
 85 86 * @return the wrapped byte buffer
 86 87 */
 87 88 public ByteBuffer getNativeBuffer() {
 88 89     this.byteBuffer.position(this.readPosition);
 89 -    this.byteBuffer.limit(readableByteCount());
 90 +    this.byteBuffer.limit(this.writePosition);
```

As you can see from the figure above, the text "Fix DefaultDataBuffer#getNativeBuffer() to set correct limit" at the top is the commit message, indicating the content and intent of the code changes. At the bottom is the diff, which shows the difference between the new version and the previous version, with green lines (beginning with a "+") indicating new lines of code and red lines (beginning with a "-") indicating removed lines.

2、Task Details

Each participant will label 2500 samples independently. Participants need to first read code changes and their corresponding commit messages, and then classify the commit messages based on their specific semantics (identified as explicit and implicit commit messages). Below, we give the definitions of explicit and implicit commit messages:

Explicit commit message: A superficial summary of the code changes. That is, it summarizes only the content of the code changes without taking into account the nature of the change logic or the intent behind it. This perspective of commit messages summarizing code changes is relatively partial and narrow.

Example1:

2		...n/java/org/springframework/boot/autoconfigure/freemarker/FreeMarkerAutoConfiguration.java	
↑...	@@ -29,6 +29,7 @@		
29	29	import org.springframework.boot.context.properties.EnableConfigurationProperties;	
30	30	import org.springframework.context.ApplicationContext;	
31	31	import org.springframework.context.annotation.Import;	
32	32	+ import org.springframework.context.annotation.ImportRuntimeHints;	
32	33	import org.springframework.ui.freemarker.FreeMarkerConfigurationFactory;	
33	34		
34	35	/**	
↑...	@@ -44,6 +45,7 @@		
44	45	@EnableConfigurationProperties(FreeMarkerProperties.class)	
45	46	@Import({ FreeMarkerServletWebConfiguration.class, FreeMarkerReactiveWebConfiguration.class,	
46	47	FreeMarkerNonWebConfiguration.class })	
48	48	+ @ImportRuntimeHints(FreeMarkerRuntimeHints.class)	
47	49	public class FreeMarkerAutoConfiguration {	
48	50		
49	51	private static final Log logger = LoggerFactory.getLog(FreeMarkerAutoConfiguration.class);	

commit message: Add runtime hints for freemarker

Implicit commit message: considering the context information of the changed code, it explains and summarizes the code changes from a relatively global perspective, which can often reflect the nature of the problem or the purpose of the code changes.

Example2:

23		...configure/src/main/java/org/springframework/boot/autoconfigure/amqp/RabbitProperties.java	
↑...	@@ -1127,17 +1127,20 @@	private String trimPrefix(String input) {	
1127	1127	}	
1128	1128		
1129	1129	private String parseUsernameAndPassword(String input) {	
1130	1130	if (input.contains("@")) {	
1131	1131	String[] split = StringUtils.split(input, "@");	
1132	1132	String creds = split[0];	
1133	1133	input = split[1];	
1134	1134	split = StringUtils.split(creds, ":");	
1135	1135	this.username = split[0];	
1136	1136	if (split.length > 0) {	
1137	1137	this.password = split[1];	
1138	1138	}	
1130	1130	+ String[] splitInput = StringUtils.split(input, "@");	
1131	1131	+ if (splitInput == null) {	
1132	1132	+ return input;	
1139	1133	}	
1140	1140	return input;	
1134	1134	+ String credentials = splitInput[0];	
1135	1135	+ String[] splitCredentials = StringUtils.split(credentials, ":");	
1136	1136	+ if (splitCredentials == null) {	
1137	1137	+ this.username = credentials;	
1138	1138	+ }	
1139	1139	+ else {	
1140	1140	+ this.username = splitCredentials[0];	
1141	1141	+ this.password = splitCredentials[1];	
1142	1142	+ }	
1143	1143	+ return splitInput[1];	

commit message: Fix NPE in RabbitProperties if user is given but password is not

After careful analysis, if a commit message is considered to be an explicit commit message, it is labeled as 0 on "isTarget"; Otherwise, the implicit commit message is labeled as 1.