CodeArts Agent

Best Practices

Issue	01
Date	2026-08-03

HUAWEI

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 AI Drawing Logic Construction Practice Based on Custom Rules

Application Scenarios

In the practical application of AI-assisted drawing, due to the lack of unified rules and constraints, AI still faces significant uncertainties in understanding user prompts, constructing image structures, and processing logical relationships between elements. This randomness often leads to deviations between the generated results and user expectations, affecting both efficiency and creative precision. Therefore, users often need to continuously optimize prompts and make iterative adjustments to gradually approach the desired effect, thereby improving the usability and reliability of AI-assisted drawing in practical scenarios.

This practice uses Huawei Cloud CodeArts Agent IDE as an example to describe how to configure rules to standardize and constrain the process of AI-assisted drawing of sin(x) function graphs. This ensures precise graph coordinates, consistent formatting, and controllable results.

Procedure

Table 1-1 Procedure for AI drawing logic construction practice based on custom rules

No.	Step	Description
1	**Preparations**	1. Create a project for storing files. 2. Enable the auto approval function to reduce manual intervention.
2	**Creating an Irregular Sin(x) Graph**	Create an irregular sin(x) function graph.
3	**Creating a Regular Sin(x) Graph**	1. Compile a drawing specification rule file. 2. Generate a sin(x) graph based on the rule file.

Preparations

Before using CodeArts Agent, create a project to store various files in the project.

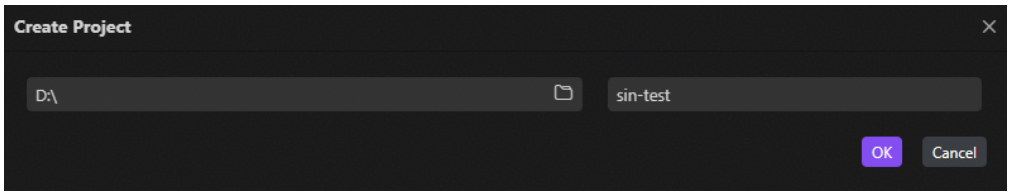
Step 1 Log in to CodeArts Agent by referring to **Quick Start**.

Step 2 Create a project for storing files.

1. On the top menu bar of the IDE, choose **File** > **New** > **Create Project** to go to the page for creating a project.
2. Select a path for storing the project, enter the project name (for example, **sin-test**), and click **OK**.

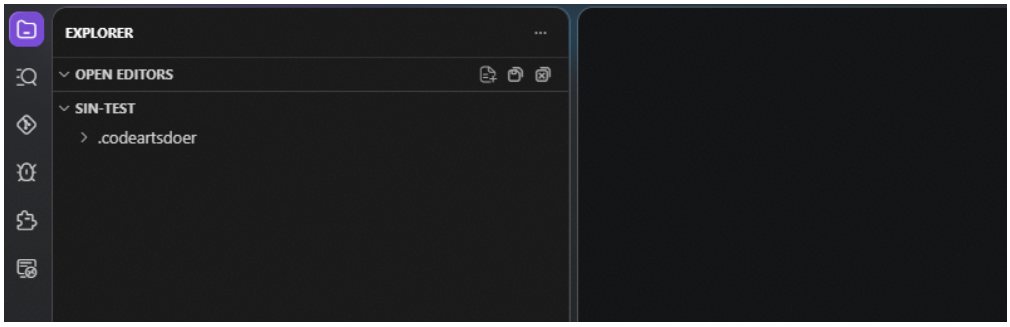
   The project name must start with a letter, and can contain a maximum of 64 characters. Letters, digits, hyphens (-), and underscores (_) are allowed.

   Figure 1-1 Creating a project

   After the project is created, you can view it under **EXPLORER**.

   Figure 1-2 Viewing the created project

Step 3 (Optional) Select the agent running mode and authorize automatic operations.

1. Click  in the upper right corner of CodeArts Agent IDE to go to the **Settings** page.
2. Choose **Chats** > **Agents** > **Terminal Command Running Mode**, select the running policy for the agent to execute terminal commands. This section uses the default running policy (**Running in Sandbox**).
3. Choose **Chats** > **Auto-approve**, and click  to enable the required items.

   After authorization, tasks for generating complex project-level code are executed automatically. Without authorization, some operations will need your manual confirmation when you code with CodeArts Agent.

> **CAUTION**
>
> Enabling auto approval can lead to operational risks. Assess these risks carefully first and only enable this feature in a secure and trustworthy environment.

Table 1-2 Parameters for auto approval

Parameter	Description	Example
Edit	Allows the agent to call tools like edit, write, and deleteFile to edit files on your computer.	Enabled
Browser Access	Allows the agent to access websites in the browser.	Enabled
Web Crawler	Allows the agent to access and capture specified web page content.	Enabled

4. Exit the current page to complete the authorization.

----End

Creating an Irregular Sin(x) Graph

To clearly demonstrate the effect of rule-based drawing, the following shows the drawing result when **no rules are applied**:

Step 1 In the input box, enter the following instruction and click  :

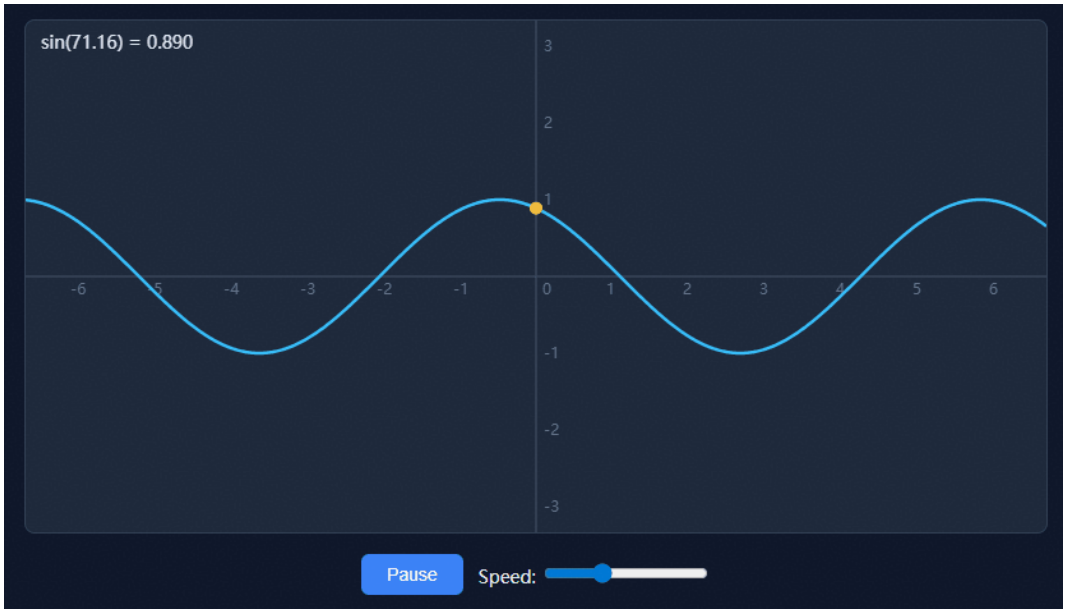
```
Draw a dynamic sin(x) function graph on the web page.
```

After the execution is complete, a file named ***.html** (for example, **index.html**) is generated in the **SIN-TEST** directory under **EXPLORER**.

Step 2 Right-click **index.html** and choose **Preview in Browser** from the shortcut menu to view the graph.

The graph shown here is for reference only. The actual output may vary.

Figure 1-3 Irregular sin(x) function graph



----End

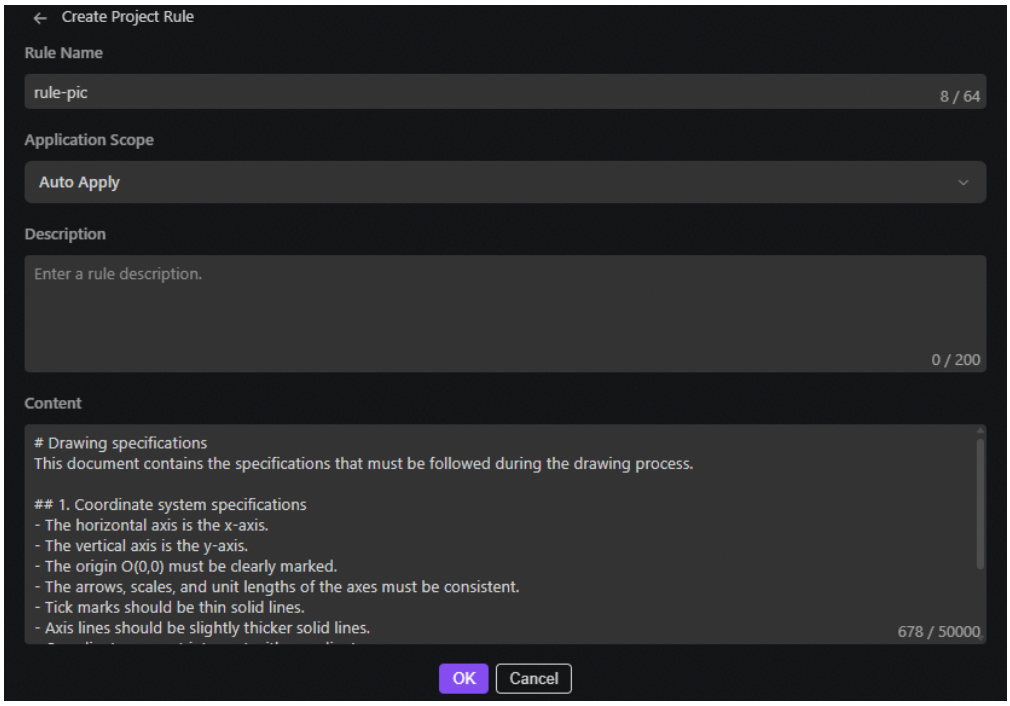
Creating a Regular Sin(x) Graph

The preceding steps describe how to draw a sin(x) graph without any rule constraints. The steps below describe how to draw a sin(x) graph based on the drawing specifications. Perform the following steps to create a project, compile a rule file, and draw a graph:

Step 1 Compile a drawing specification rule file.

1. In the upper left corner of the IDE, choose **File** > **New** > **Create Project**. Then, create a project named **sin-test-rule**, and open it in a new window.
2. Click  in the upper right corner of CodeArts Agent IDE to go to the **Settings** page.
3. Choose **Skills and Rules** and click  next to **Rules** on the **Project** tab.
4. Set **Rule Name** to **rule-pic**, retain the default option **Auto Apply** for **Application Scope**, enter the drawing rule of sin(x) in the text box under **Content**, and click **OK**.

Figure 1-4 Creating a rule-pic rule



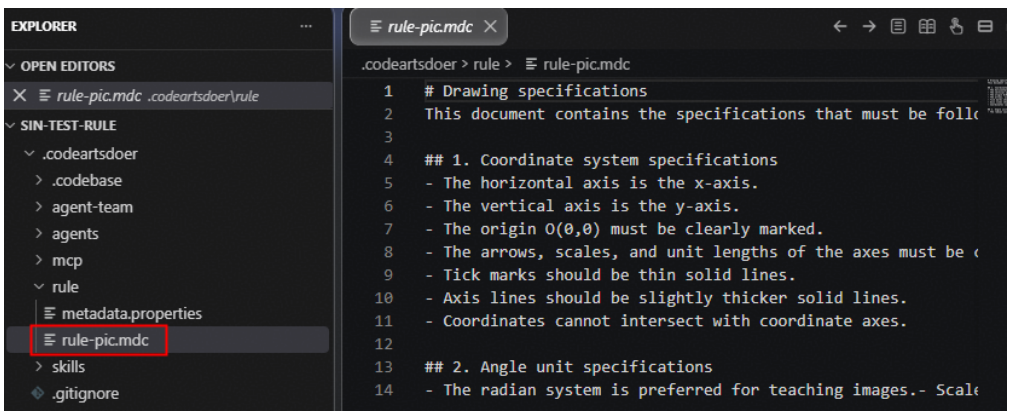

```
# Drawing specifications
This document contains the specifications that must be followed during the drawing process.

## 1. Coordinate system specifications
- The horizontal axis is the x-axis.
- The vertical axis is the y-axis.
- The origin O(0,0) must be clearly marked.
- The arrows, scales, and unit lengths of the axes must be consistent.
- Tick marks should be thin solid lines.
- Axis lines should be slightly thicker solid lines.
- Coordinates cannot intersect with coordinate axes.

## 2. Angle unit specifications
- The radian system is preferred for teaching images.
- Scales to be marked on the coordinate axes: 0, $\frac{\pi}{2}$, $\pi$, $\frac{3\pi}{2}$, and $2\pi$
```

After the rule is created, you can view the created rule file **rule-pic.mdc** in the **.codeartsdoer/rule** directory of the current project (SIN-TEST-RULE).

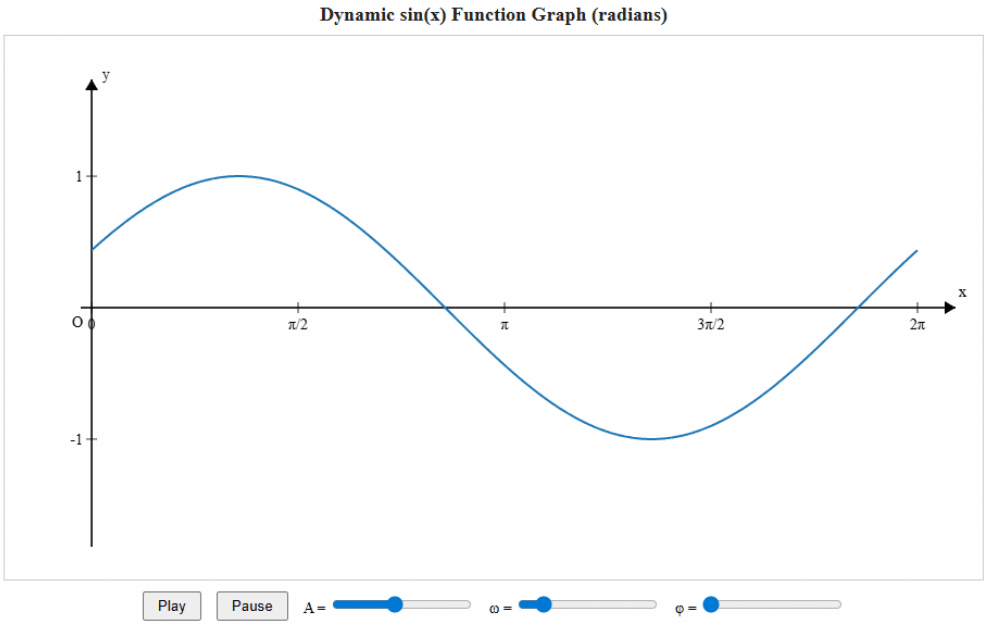
Figure 1-5 Viewing the rule file rule-pic.mdc



Step 2 Generate a sin(x) graph based on the rule file.

1. In the input box, enter the following instruction and click :
   ```
   Draw a dynamic sin(x) function graph on the web page.
   ```
   After the execution is complete, a file named ***.html** (for example, **index.html**) is generated in the **SIN-TEST-RULE** directory under **EXPLORER**.
2. Right-click **index.html** and choose **Preview in Browser** from the shortcut menu to view the graph.
   The graph shown here is for reference only. The actual output may vary.

Figure 1-6 Regular sin(x) function graph



----End

Summary

By comparing the sin(x) function graphs drawn with and without rule constraints, it is clear that the introduction of rules significantly improves the coordinate specifications and mathematical expression accuracy of the function graphs, effectively avoiding issues such as axis confusion. This fully demonstrates the important role of rules in unifying standards, constraining behavior, and ensuring output quality and stability, making the results more standard, reliable, and predictable.

This practice helps you understand **rules**. In addition to rules, CodeArts Agent also provides **skills** and **MCP** to build diverse capabilities of agents. **Table 1-3** compares the differences between the functions.

Table 1-3 Comparison between skills and other functions

Dimension	Rule	Skill	MCP Server
Definition	Constrains the behavior of an agent.	Describes how to complete a specific task.	Invokes external tools.
Loading method	Is loaded into the context at the beginning of a chat for continuous inference.	Is loaded on demand to reduce context usage.	Is not involved in the inference process; calls external interfaces as required.
Application scenario	Defines code specifications, output formats, team collaboration agreements, etc. Example: Code must comply with PEP 8 specifications.	Encapsulates testing processes, development tasks, complex business logic, etc. Example: executing UI automated testing.	Connects to external systems to perform specific operations. Example: controlling browser-related operations.
Key differences	Controls how an agent should behave, defining the behavior boundary.	Specifies how to complete a task, serving as a process guide.	Provides **the capability to invoke tools**, representing the execution dimension.

2 Using Skill Creator to Generate a Skill for Splitting PDF Documents by Chapter

In daily office work, large PDF documents such as manuals, reports, theses, and bidding documents are commonly processed. Manually splitting chapters and renaming the resulting files one by one is time-consuming and affects document processing efficiency. To address this pain point, you can use Skill Creator provided in **Marketplace** of CodeArts Agent to quickly generate a dedicated **PDF-splitting skill**.

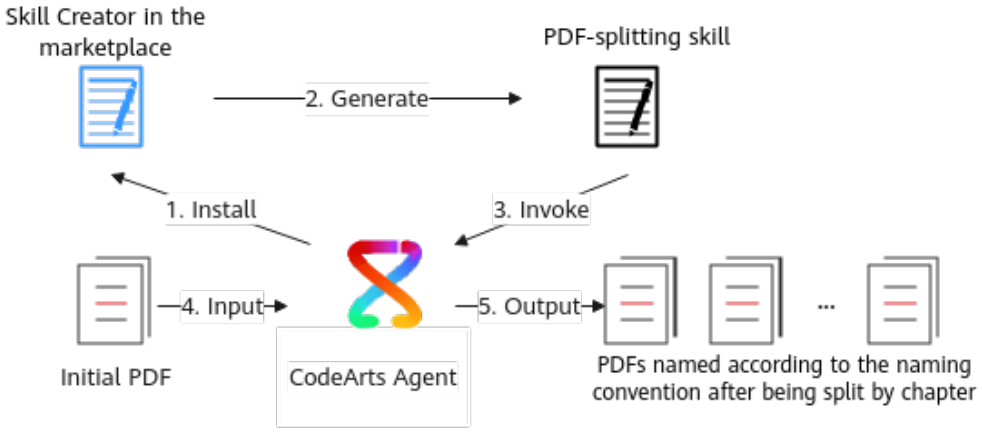
You can find Skill Creator in **Marketplace** and use it after installation. When you need a skill, simply say, "Help me create a skill", and specify its desired functions. CodeArts Agent will then produce the necessary skill package.

The **PDF-splitting skill** can identify chapter titles in a document, split the document into multiple independent PDF files by chapter, and name the files based on the chapter names. This eliminates the need for manual pagination and renaming, significantly simplifying the process of handling large documents.

- **Large document splitting**: A manual or report with hundreds of pages can be split into single-chapter files for easy distribution and viewing.
- **Standard archiving**: After splitting, the files are automatically named based on the chapter names and can be directly imported into the database with just one click.
- **Targeted distribution**: Only a specific chapter is distributed, eliminating the need to send the entire large document, which is more secure and saves bandwidth.

Figure 2-1 shows the entire process.

Figure 2-1 Practice process



Preparations

Before using CodeArts Agent, create a project to store various files in the project. After the project is created, enable the auto approval function. In this way, AI will automatically perform related operations without manual intervention.

Step 1 Log in to CodeArts Agent by referring to **Quick Start**.

Step 2 Create a project for storing files.

1. On the top menu bar of the IDE, choose **File** > **New** > **Create Project** to go to the page for creating a project.
2. Select a path for storing the project, enter the project name (for example, **pdf-skill-demo**), and click **OK**.

   The project name must start with a letter, and can contain a maximum of 64 characters. Letters, digits, hyphens (-), and underscores (_) are allowed.

   After the project is created, you can view the created **PDF-SKILL-DEMO** project under **EXPLORER**.

Step 3 (Optional) Select the agent running mode and authorize automatic operations.

1. Click  in the upper right corner of CodeArts Agent IDE to go to the **Settings** page.
2. Choose **Chats** > **Agents** > **Terminal Command Running Mode**, select the running policy for the agent to execute terminal commands. This section uses the default running policy (**Running in Sandbox**).
3. Choose **Chats** > **Auto-approve**, and click  to enable the required items.

   After authorization, tasks for generating complex project-level code are executed automatically. Without authorization, some operations will need your manual confirmation when you code with CodeArts Agent.

> **CAUTION**
>
> Enabling auto approval can lead to operational risks. Assess these risks carefully first and only enable this feature in a secure and trustworthy environment.

Table 2-1 Parameters for auto approval

Parameter	Description	Example
Edit	Allows the agent to call tools like edit, write, and deleteFile to edit files on your computer.	Enabled
Browser Access	Allows the agent to access websites in the browser.	Enabled
Web Crawler	Allows the agent to access and capture specified web page content.	Enabled

4. Exit the current page to complete the authorization.

----End

Installing Skill Creator in the Marketplace

Step 1 Click  in the upper right corner of CodeArts Agent IDE to go to the **Settings** page.

Step 2 In the navigation pane on the left, choose **Skills and Rules**.

Step 3 Click **Marketplace**. On the displayed page, locate **Skill Creator**.

Step 4 After specifying the skill, click  , set the installation location to **Project**, and click **OK**.

After the installation is successful,  next to the skill changes to **Installed**.

----End

Generating a PDF-Splitting Skill

Step 1 In the input box on the CodeArts Agent chat panel, enter the following prompts and click  :

```
Invoke Skill Creator to generate a professional PDF-splitting skill by chapter based on the following rules:
[Triggering rules]
- Enable the skill as long as a user says "Split the PDF by chapter", "Split and name the PDF by chapter title", or "Split the PDF chapters."
[Execution rules]
- Automatically identifies chapter titles and boundaries in PDF files.
- Splits the original PDF into multiple independent PDFs by chapter.
- Names each split file with the corresponding chapter name.
```

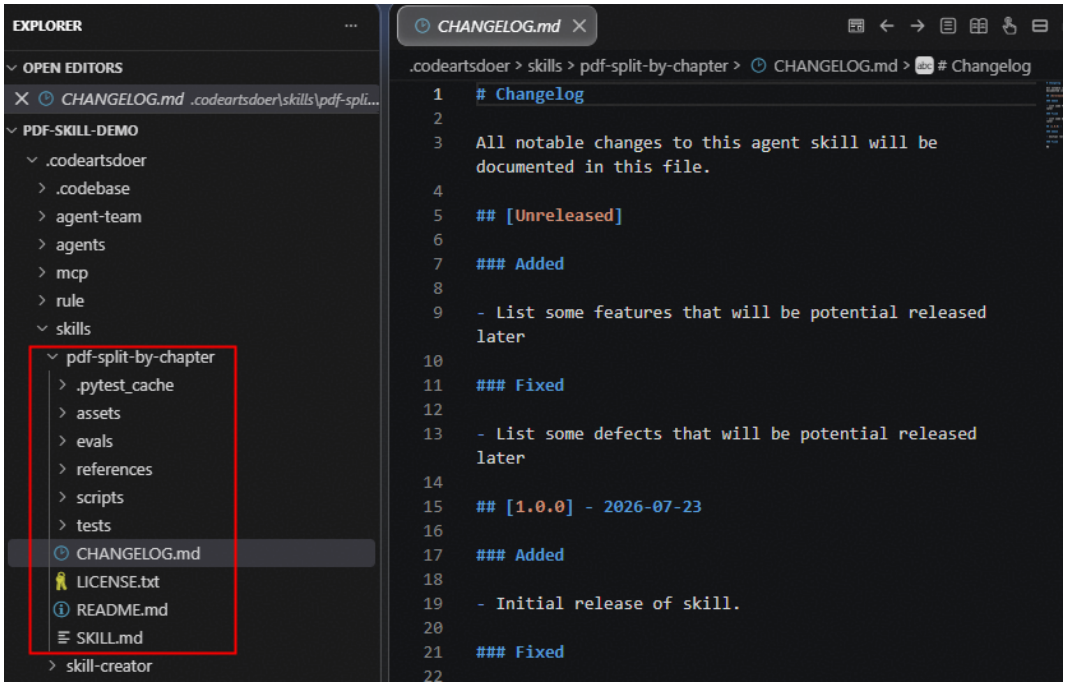


```
[Delivery requirements] No page loss or incorrect splitting. Return all split files after the operation is complete.
```

Step 2 Confirm the creation information, select **"Proceed with creation" – Create the skill with the above name and path**, click **Submit**, and wait until the skill is successfully created.

After the skill is created, the **pdf-split-by-chapters** skill package is generated in the **PDF-SKILL-DEMO** directory under **EXPLORER**.

Figure 2-2 Viewing the generated skill package



Step 3 Place the PDF to be split in the current project directory and enter the following prompt in the text box on the chat panel:

```
Invoke pdf-chapter-splitter to split the PDF of the CodeArts Agent user guide by chapter.
```

Click . CodeArts Agent will help you split the PDF.

Figure 2-3 Splitting completed

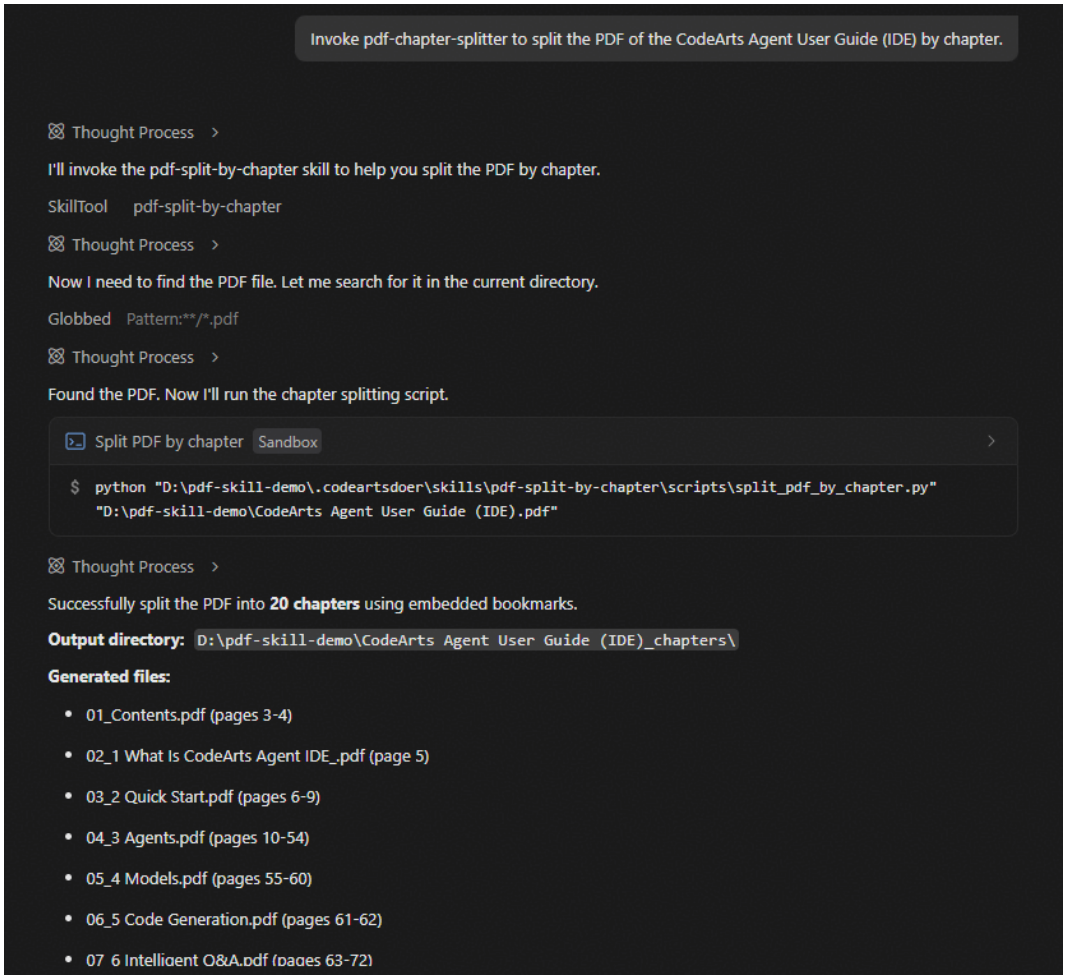
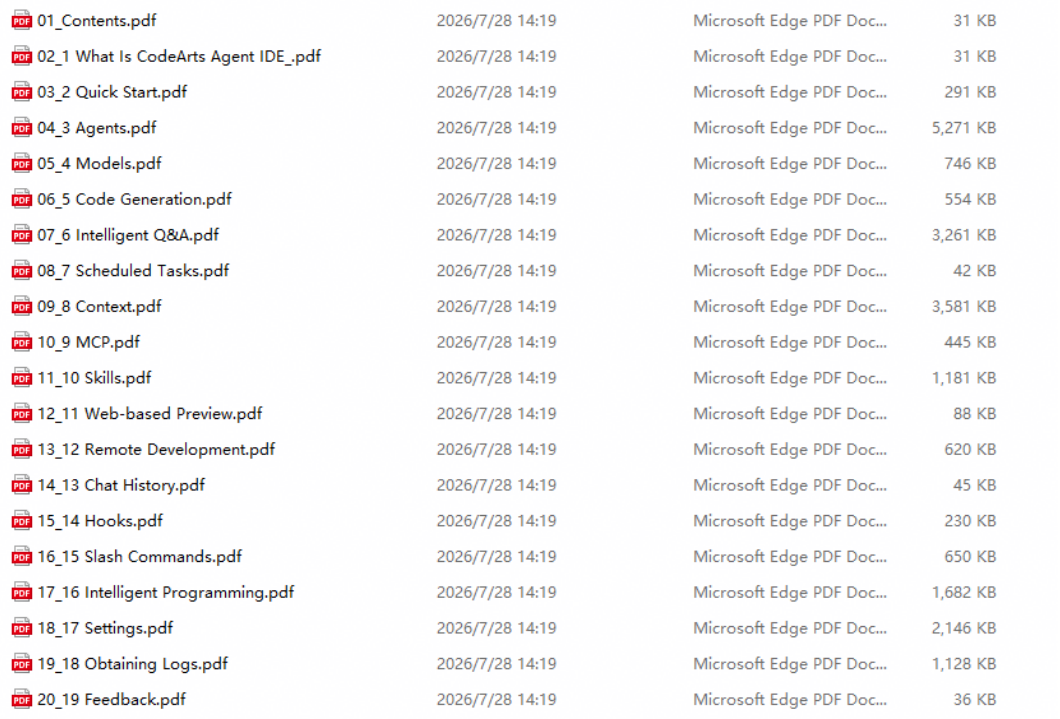


Figure 2-4 Viewing the splitting result in the corresponding directory



NOTE

The graph shown here is for reference only. The actual output may vary.

----End

Prompt Optimization

In the previous section, the prompts for generating skills provide basic core functions, which can only be used to perform simple splitting. There are no unified standards for chapter identification, naming, or exception handling. When the content of a PDF document is complex or batch processing is required, issues such as decreased recognition accuracy, prolonged processing time, and naming exceptions often occur.

You can optimize the prompts, for example, specifying the chapter identification style and adding prerequisite file validation rules, splitting requirements, naming rules, and exception messages. This ensures that skills can be executed in a way that better meets your business needs and can run stably and reliably over the long term.

Prompt After Optimization

```
Invoke Skill Creator to generate a professional PDF-splitting skill by chapter based on the following rules:
[Triggering rules]
- Positive triggering: The skill is enabled when a user requests to split a PDF by chapter or title and automatically name the resulting files.
- Exclusive constraint: This skill does not perform other operations such as PDF merging, encryption/decryption, text extraction, or format conversion to avoid function conflicts.
[Standard handling process]
```


1. PDF file preprocessing: Check the PDF file status, identify encrypted files, and provide clear prompts and pop-up notifications to users in case of exceptions.
2. Chapter identification and parsing: Identify hierarchical headings, support common formats such as "Chapter *X*", "1.1", "Chapter 1", and "1", and automatically remove the cover and table of contents pages.
3. Precise document splitting: Split pages based on chapter boundaries to ensure that chapter content is complete and coherent, without missing pages, missing content, content truncation, or page disorder.
4. File naming after splitting: Use the corresponding chapter name as the name of the split file, automatically remove system-prohibited special characters such as / \ : * ? " < > | from the file name, and ensure compatibility with Windows and macOS.
5. Result receipt: After the splitting is complete, output the splitting details list, clearly mark the chapter name and start and end page numbers of each file, and return all split files in an orderly manner.
[Quality requirements]
Ensure that the splitting result is accurate, the file names do not contain garbled characters, and the files are fully compatible with mainstream PDF formats, meeting the requirements of daily office work and batch production.

Summary of Prompt Optimization Tips

Issues such as skill execution deviation and file naming exceptions are often caused by vague prompt semantics and lack of rules. When writing prompts, follow the practical tips described in the table below to ensure standardization. This can comprehensively optimize the skill execution effect, reduce running deviations, and significantly improve the practicality and stability of the PDF chapter splitting function.

Table 2-2 Prompt optimization tips

Tip	Good Example	Bad Example
Specify chapter identification rules.	Identify hierarchical headings, support common formats such as "Chapter *X*", "1.1", "Chapter 1", and "1", and automatically remove the cover and table of contents pages.	Automatically identifies chapters in PDF files.
Specify exclusive rules to prevent foundation models from triggering other operations by mistake.	Do not perform other PDF operations such as merging, encryption, and text extraction to avoid conflicts.	No content is provided.
Consider exception scenarios.	• Remove special characters (/ \ : * ? " < > |) from file names to avoid errors. • If a chapter title cannot be identified, the system displays a message indicating that **no chapter is detected and the corresponding page number is marked**.	No content is provided.

3 Best Practices for Checkpoint Session Rollback

After the basic core version of a software system is developed, various extended functions are continuously added in iterations to enrich product capabilities and meet diversified service requirements. However, some non-core extended functions may not match service requirements, increase system performance loss, or cause redundant and complex interactions after actual verification, affecting overall stability and user experience. The session-level checkpoint mechanism is used to automatically archive the system status and accurately roll back the system to the initial basic version with one click, without affecting core services or leaving redundant code and configurations. This efficiently solves the problems caused by invalid iterations.

This practice uses a lightweight coffee ordering system to demonstrate checkpoint capabilities during chatting, including status archiving, precise rollback, and UI restoration.

Preparations

Before using CodeArts Agent, create a project to store various files in the project.

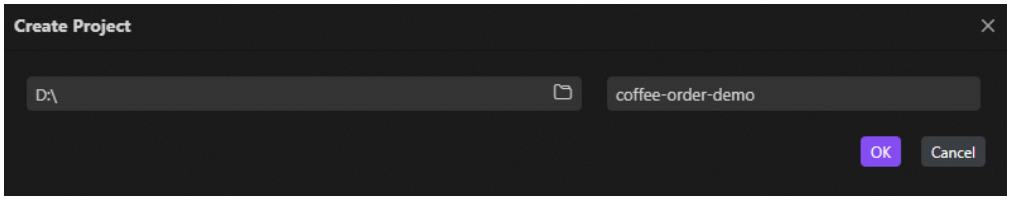
Step 1 Log in to CodeArts Agent by referring to **Quick Start**.

Step 2 Create a project for storing files.

1. On the top menu bar of the IDE, choose **File** > **New** > **Create Project** to go to the page for creating a project.
2. Select a path for storing the project, enter the project name (for example, **coffee-order-demo**), and click **OK**.

   The project name must start with a letter, and can contain a maximum of 64 characters. Letters, digits, hyphens (-), and underscores (_) are allowed.

Figure 3-1 Creating a project



After the project is created, you can view the created **COFFEE-ORDER-DEMO** project under **EXPLORER**.

Step 3 (Optional) Select the agent running mode and authorize automatic operations.

1. Click  in the upper right corner of CodeArts Agent IDE to go to the **Settings** page.
2. Choose **Chats** > **Agents** > **Terminal Command Running Mode**, select the running policy for the agent to execute terminal commands. This section uses the default running policy (**Running in Sandbox**).
3. Choose **Chats** > **Auto-approve**, and click  to enable the required items.

   After authorization, tasks for generating complex project-level code are executed automatically. Without authorization, some operations will need your manual confirmation when you code with CodeArts Agent.

> **CAUTION**
>
> Enabling auto approval can lead to operational risks. Assess these risks carefully first and only enable this feature in a secure and trustworthy environment.

Table 3-1 Parameters for auto approval

Parameter	Description	Example
Edit	Allows the agent to call tools like edit, write, and deleteFile to edit files on your computer.	Enabled
Browser Access	Allows the agent to access websites in the browser.	Enabled
Web Crawler	Allows the agent to access and capture specified web page content.	Enabled

4. Exit the current page to complete the authorization.

----End

Generating a Basic Coffee Ordering System

The initial phase of the ordering system is built in a lightweight manner, focusing on the core ordering service process. Non-essential marketing and value-added functions are not expanded to ensure that the core link is simple, available, and reliable.

Step 1 In the input box of the chat window, enter the following prompts to create a coffee ordering system:

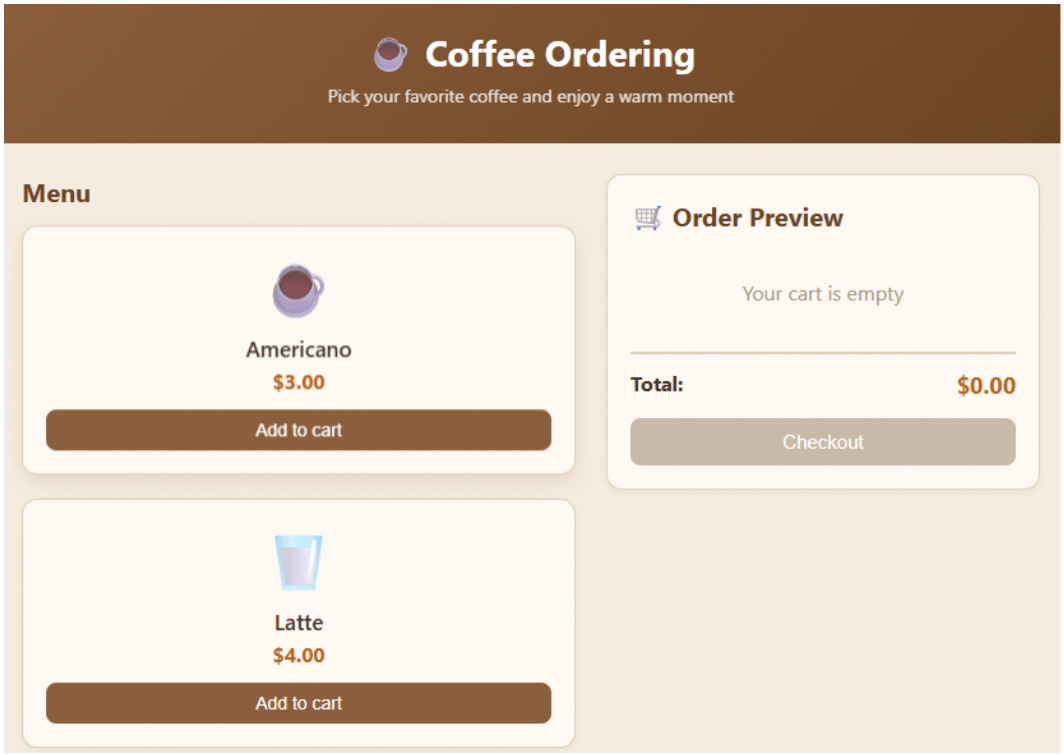
```
Use HTML, CSS, and JavaScript to create a basic coffee ordering system.
Function requirements:
- Menu display: A static list containing at least three types of coffee (such as Americano, Latte, and Cappuccino) is displayed. Each type of coffee is marked with its name, price, and a general icon.
- Add to cart: There is an "Add to cart" button next to each coffee.
- Order preview: There is an area on the page that displays the items, quantities, and calculated total price in the current shopping cart in real time.
- Style: The UI should be simple, clean, and warm in color.
```

After the execution is complete, a file named **index.html** is generated in the **COFFEE-ORDER-DEMO** directory under **EXPLORER**.

Step 2 Right-click **index.html** and choose **Preview in Browser** from the shortcut menu to view the website.

The graph shown here is for reference only. The actual output may vary.

Figure 3-2 Coffee ordering system



----End

Adding the Personalized Recommendation Function

The team proposed an idea: to add a "Daily Special" recommendation function for phased marketing activities to attract customers to place orders. This type of function is not a core function of the system. After the activity period ends, the function will be taken offline and the original GUI logic will be restored.

Step 1 In the input box of the chat window, enter the following prompts to optimize the coffee ordering system:

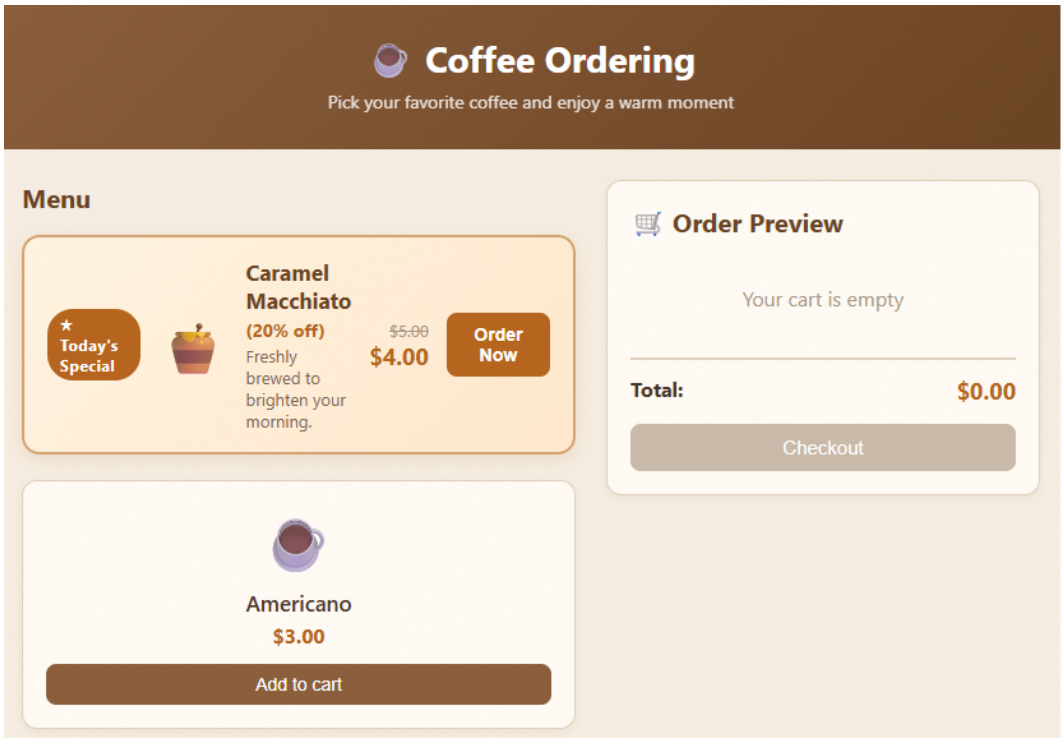
```
Add a "Daily Special" recommendation function to the coffee ordering system.
Function requirements:
- Add a "Today's Special" area to the top of the menu.
- This area dynamically displays a type of coffee as a special offer, along with a simple recommendation reason.
- For example, "Today's Special: Iced Lemon Tea (20% off)" with a separate "Order Now" button.
```

After the execution is complete, the **index.html** file in the **COFFEE-ORDER-DEMO** directory under **EXPLORER** will be updated accordingly.

Step 2 Right-click **index.html** and choose **Preview in Browser** from the shortcut menu to view the website.

The graph shown here is for reference only. The actual output may vary.

Figure 3-3 Adding daily special recommendations



----End

Rolling Back to the Basic Version

After team discussion and evaluation, it was found that the "Daily Special" recommendation module causes strong visual interference, has redundant

functions, and incurs high maintenance costs. It is more harmful than beneficial to the website at this stage. Therefore, this function is not implemented for the time being, and the simple and basic core ordering version is used.

Step 1 On the chat panel, roll back a chat.

Hover over the chat to be rolled back and click . A confirmation dialog box is displayed, showing the name of the file to be restored.

Figure 3-4 Clicking the rollback icon

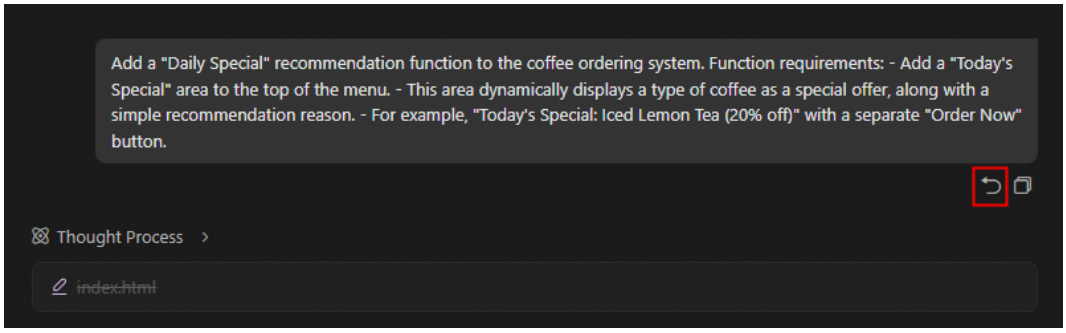
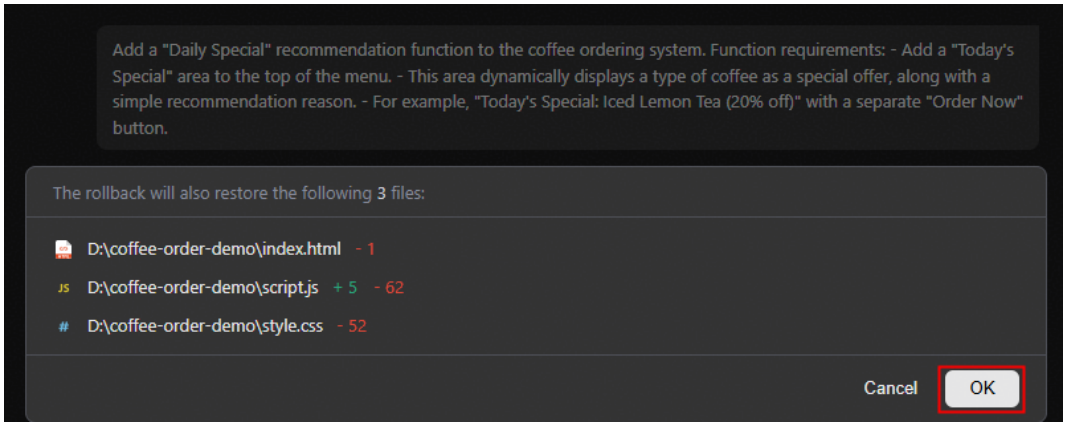


Figure 3-5 Confirmation dialog box



Step 2 Click **OK** to roll back the chat.

Files will be rolled back to their historical states, and the messages at and after this timestamp in the current chat will disappear. The coffee ordering system is rolled back to the basic version synchronously.

----End

Summary

Through the preceding practices, you can clearly see the core value of checkpoints:

- **Version security assurance**: A restorable baseline version is provided for the basic system generated by CodeArts Agent.
- **Quick cancellation of invalid iterations**: If an added function does not meet expectations, you can roll back the code with just one click, without the need to manually delete the code.

- **Reduced trial-and-error costs**: Free function expansion is allowed without compromising the initial stable system.
- **Improved O&M efficiency**: You can quickly restore a project to the version that best meets your requirements without regenerating a project.

4 Local Development and Static Product Verification of Tic-Tac-Toe

Frontend development often encounters issues such as non-standard TypeScript types, time-consuming UI style debugging, and error-prone project configurations. This practice uses the local development of a Tic-Tac-Toe single-page application (SPA) built with Next.js, React, and TypeScript as an example to help developers complete project initialization, code generation, and full local function verification using CodeArts Agent, improving frontend R&D efficiency and code quality.

Preparations

Before using CodeArts Agent, create a project to store various files in the project. After the project is created, enable the auto approval function. In this way, AI will automatically perform related operations without manual intervention.

Step 1 Log in to CodeArts Agent by referring to **Quick Start**.

Step 2 Create a project for storing files.

1. On the top menu bar of the IDE, choose **File** > **New** > **Create Project** to go to the page for creating a project.
2. Select a path for storing the project, enter the project name (for example, **Tic-Tac-Toe_Demo**), and click **OK**.

   The project name must start with a letter, and can contain a maximum of 64 characters. Letters, digits, hyphens (-), and underscores (_) are allowed.

   After the project is created, you can view the created **TIC-TAC-TOE_DEMO** project under **EXPLORER**.

Step 3 (Optional) Select the agent running mode and authorize automatic operations.

1. Click  in the upper right corner of CodeArts Agent IDE to go to the **Settings** page.
2. Choose **Chats** > **Agents** > **Terminal Command Running Mode**, select the running policy for the agent to execute terminal commands. This practice has many dependencies, so the manual mode is recommended.
3. Choose **Chats** > **Auto-approve**, and click  to enable the required items.

After authorization, tasks for generating complex project-level code are executed automatically. Without authorization, some operations will need your manual confirmation when you code with CodeArts Agent.

CAUTION

Enabling auto approval can lead to operational risks. Assess these risks carefully first and only enable this feature in a secure and trustworthy environment.

Table 4-1 Parameters for auto approval

Parameter	Description	Example
Edit	Allows the agent to call tools like edit, write, and deleteFile to edit files on your computer.	Enabled
Browser Access	Allows the agent to access websites in the browser.	Enabled
Web Crawler	Allows the agent to access and capture specified web page content.	Enabled

4. Exit the current page to complete the authorization.

----End

Generating the Core Code and UI for Tic-Tac-Toe

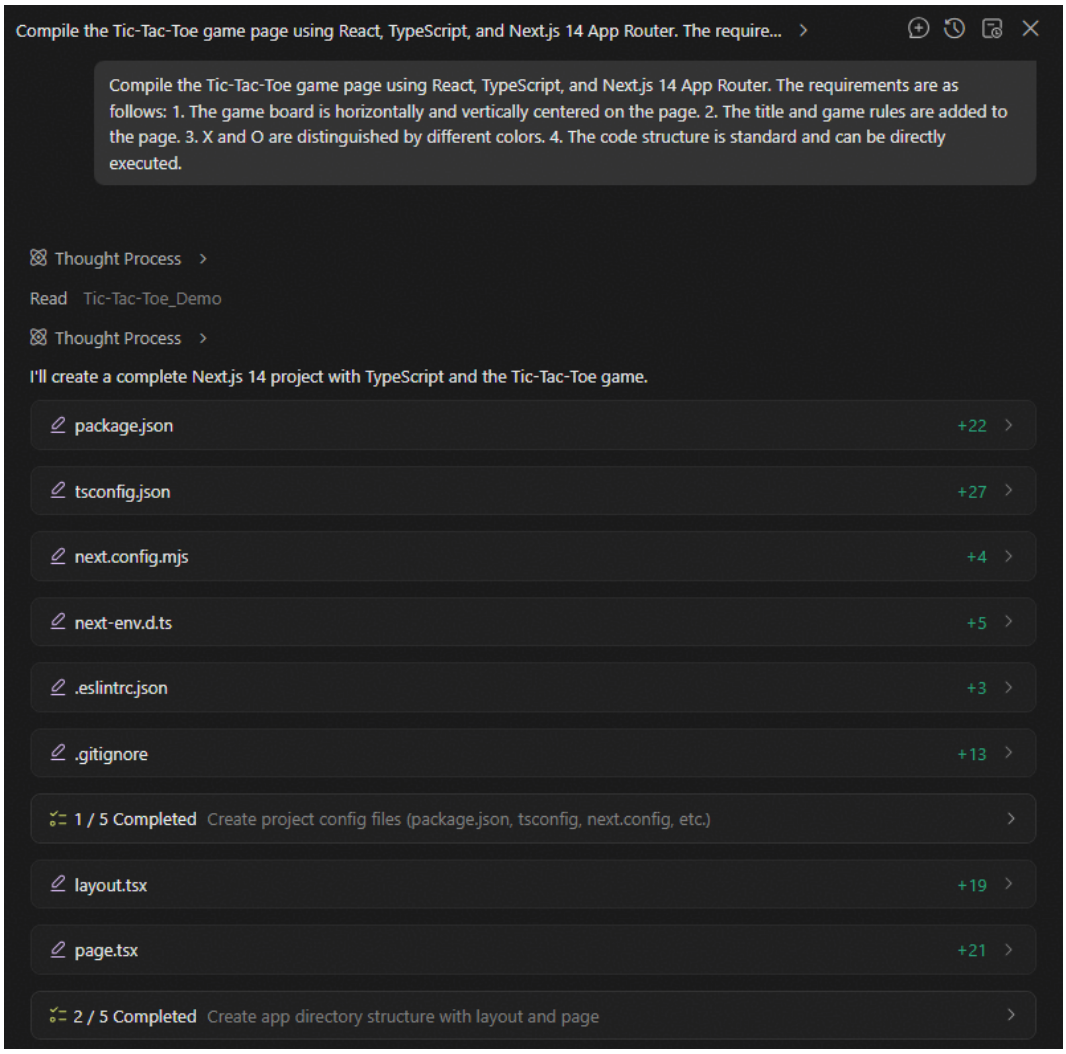
Step 1 In the input box on the CodeArts Agent chat panel, enter the following prompts and click :

```
Compile the Tic-Tac-Toe game page using React, TypeScript, and Next.js 14 App Router. The requirements are as follows:
1. The game board is horizontally and vertically centered on the page.
2. The title and game rules are added to the page.
3. X and O are distinguished by different colors.
4. The code structure is standard and can be directly executed.
```

Step 2 During task execution, manually confirm all terminal operations and high-risk commands and wait until the project is successfully created.

The graph shown here is for reference only. The actual output may vary.

Figure 4-1 Project creation process



Step 3 Click **Accept All** to accept the current file modification.

----End

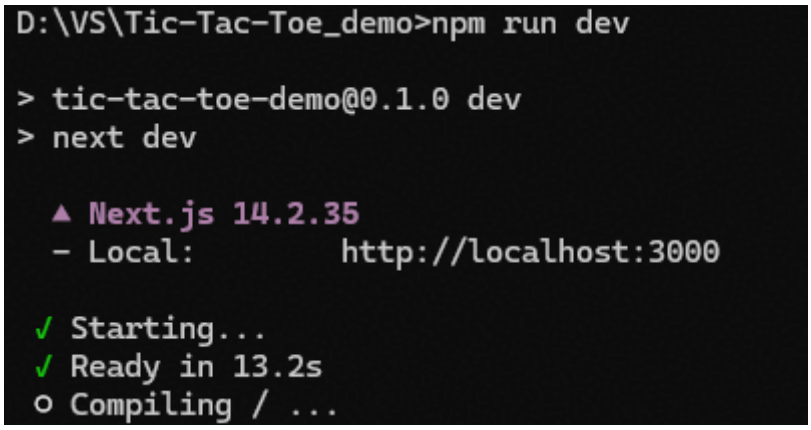
Debugging the Development Environment

After the code is generated, you can verify all functions in the development environment. After confirming that the functions are normal, perform packaging and configuration to avoid logic issues in advance.

Step 1 Start the Next.js local development server.

1. Open the CLI on your local device. The subsequent steps use Windows 11 as an example. Press **Win+R**, enter **cmd** in the **Run** dialog box, and press **Enter**.
2. Switch to the directory where the current project is located. The following directory is used as an example:
   ```
   D:
   cd \VS\Tic-Tac-Toe_Demo
   ```
3. Run the following command to start the Next.js local development server and wait until the compilation is complete:
   ```
   npm run dev
   ```


Figure 4-2 Server started



Step 2 Open a browser, enter **http://localhost:3000** in the address box, and verify the functions one by one.

- UI: The title, game rules, and board layout are displayed properly. X and O are clearly distinguished by color.
- Interaction: Clicking a cell places a piece. An occupied cell cannot be clicked again.
- Logic: Horizontal, vertical, and diagonal lines are correctly evaluated for wins, and a draw is properly detected when the board is full.

----End

Static Packaging and Verification

After the development environment passes the verification, configure packaging rules, perform compilation, and verify static resources.

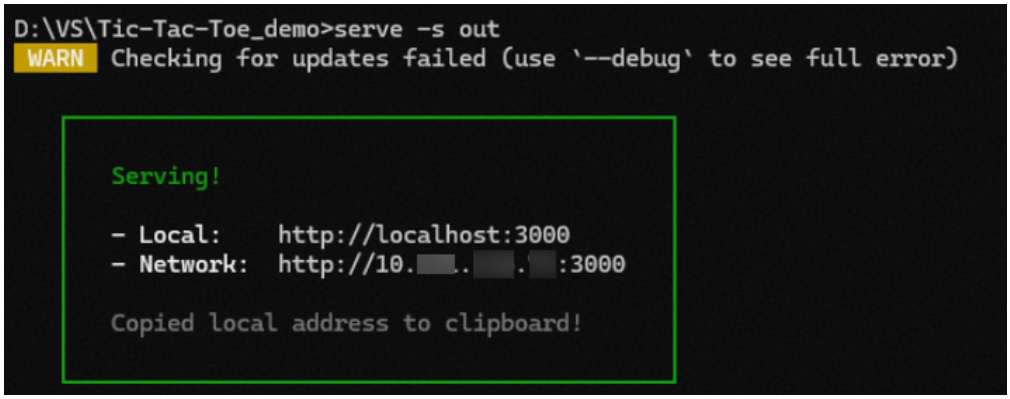
Step 1 In the input box on the CodeArts Agent chat panel, enter the following packaging rules and click :

```
Modify next.config.mjs to enable static export and set the packaging output directory to out. Ensure that the code does not contain errors and can be packaged properly.
```

Step 2 Package and build the project and preview the generated static resources.

1. Open the CLI on your local device. The subsequent steps use Windows 11 as an example. Press **Win+R**, enter **cmd** in the **Run** dialog box, and press **Enter**.
2. Switch to the directory where the current project is located. The following directory is used as an example:
   ```
   D:
   cd \VS\Tic-Tac-Toe_Demo
   ```
3. Run the startup command to package and build the project in the production environment, and wait until the full compilation and optimization are complete.
   ```
   npm run build
   ```
4. Run the following command to install the static file server globally (first-time setup only):
   ```
   npm install -g serve
   ```
5. Run the following command to start the static file server:
   ```
   serve -s out
   ```


Figure 4-3 Running the command for starting the static file server on the local terminal



Step 3 Access the page based on the output address, and verify the functions such as the UI, interaction, and winner determination again to ensure that the static product is consistent with the verification result in the development environment.

The graph shown here is for reference only. The actual output may vary.

----End

Common Issues and Solutions

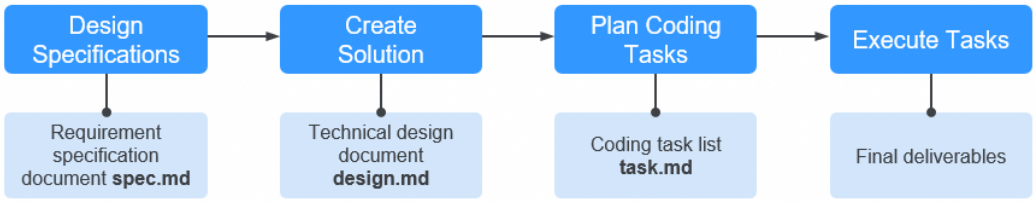
Symptom	Root Cause	Solution
The page cannot be accessed after **npm run dev** is executed.	Port occupation	Specify a new port. For example, if port 3000 is occupied, specify port 3001. Run **npm run dev --p 3001** (new port).
The development environment functions properly, but the **out** directory is missing after packaging.	Static export is not correctly configured.	Run the packaging configuration prompt again to modify the **next.config.mjs** file. Modify **next.config.mjs** to enable static export and set the packaging output directory to **out**. Ensure that the code does not contain errors and can be packaged properly.

5 Standard Workflow and Slash Command Practices in SDD

The Spec-Driven Development (SDD) mode of CodeArts Agent is a specification-centric software development method. After developers provide requirement descriptions, the agent generates specifications, design solutions, and task plans in sequence. This helps developers and the agent reach a consensus on the task objectives and implementation solutions before coding, ultimately producing high-quality code.

The SDD mode of CodeArts Agent uses a four-phase progressive process, as shown in **Figure 5-1**.

Figure 5-1 SDD procedure and outputs



Based on this, CodeArts Agent provides two usage modes:

- Standard SDD workflow: After the agent's dialog switches to the spec-driven mode, the agent takes the lead in the process and executes the tasks in the sequence shown in **Figure 5-1**. After the outputs of each phase are manually confirmed, the next phase is executed.
- Slash command mode: Developers can use slash commands shown in **Table 5-1** to skip or repeat any phase.

Table 5-1 SDD slash commands

Command	Function	Input	Output
/sdd-new	Creates requirement specifications.	Requirement description	spec.md

Command	Function	Input	Output
/sdd-design	Creates a technical design document.	Name of the folder where **spec.md** is located	design.md
/sdd-tasks	Creates coding tasks.	Name of the folder where **spec.md** and **design.md** are located	tasks.md
/sdd-apply	Executes coding tasks.	Name of the folder where **tasks.md** is located	Code file

This section uses the example of developing user registration and login functions for a web application to compare the differences and application scenarios of the two modes.

Preparations

Before using CodeArts Agent, create a project to store various files in the project. After the project is created, enable the auto approval function. In this way, AI will automatically perform related operations without manual intervention.

Step 1 Log in to CodeArts Agent by referring to **Quick Start**.

Step 2 Create a project for storing files.

1. On the top menu bar of the IDE, choose **File** > **New** > **Create Project** to go to the page for creating a project.
2. Select a path for storing the project, enter the project name (for example, **SDD_Demo**), and click **OK**.

   The project name must start with a letter, and can contain a maximum of 64 characters. Letters, digits, hyphens (-), and underscores (_) are allowed.

   After the project is created, you can view the created **SDD_DEMO** project under **EXPLORER**.

Step 3 (Optional) Select the agent running mode and authorize automatic operations.

1. Click  in the upper right corner of CodeArts Agent IDE to go to the **Settings** page.
2. Choose **Chats** > **Agents** > **Terminal Command Running Mode**, select the running policy for the agent to execute terminal commands. This section uses the default running policy (**Running in Sandbox**).
3. Choose **Chats** > **Auto-approve**, and click  to enable the required items.

After authorization, tasks for generating complex project-level code are executed automatically. Without authorization, some operations will need your manual confirmation when you code with CodeArts Agent.

CAUTION

Enabling auto approval can lead to operational risks. Assess these risks carefully first and only enable this feature in a secure and trustworthy environment.

Table 5-2 Parameters for auto approval

Parameter	Description	Example
Edit	Allows the agent to call tools like edit, write, and deleteFile to edit files on your computer.	Enabled
Browser Access	Allows the agent to access websites in the browser.	Enabled
Web Crawler	Allows the agent to access and capture specified web page content.	Enabled

4. Exit the current page to complete the authorization.

----End

Standard SDD Workflow

The standard SDD workflow is a built-in spec-driven mode of CodeArts Agent. After you select spec-driven on the chat panel and enter the requirement description, the agent will execute the preset workflow step by step. First, it generates the requirement specification document **spec.md**. After you confirm that the document is correct, the agent automatically creates a solution and generates the **design.md** document. Then, it generates the task list **tasks.md** and finally executes the coding. The entire process is driven by AI, and you only need to confirm the key nodes (accept or reject the generated documents).

Step 1 At the bottom of the input box on the chat panel, choose **Built-in** > **Agent** to switch to the agent mode. The selected model is displayed on the right. You can select a model as required from the drop-down list.

NOTE

If the chat panel of CodeArts Agent is not displayed, click  (**Expand AI Sidebar**) in the upper right corner of the top menu bar to open it.

Step 2 On the chat panel, click **Spec-driven**.

Step 3 In the input box on the CodeArts Agent chat panel, enter the following prompts and click  :


```
Develop a user login API that supports identity authentication using a username and password. After successful authentication, return a JWT token. If the login fails, record the number of failed attempts. If the login fails for five consecutive times, lock the account for 15 minutes.
```

Step 4 View the generated requirement specification document **spec.md**. Provide modification suggestions to enable CodeArts Agent to modify **spec.md** accordingly.

```
Resolve the following issues:
After a successful login, return the token and basic user information (nickname and role).
The lockout duration of 15 minutes is too long. Change it to 5 minutes.
Add rate limiting to the API: maximum 10 requests per minute per IP address.
Do not use the response code 200. Use the service code 20000 to indicate success.
```

Step 5 View the modified **spec.md** document again. After confirming that the requirements are correctly understood, click **Start Create Solution**.

Step 6 View the generated **design.md** document. After confirming that the technical solution is reasonable, click **Start Plan Coding Tasks**.

Step 7 View the generated **tasks.md** document. After confirming that the task breakdown is complete, click **Start Execute Tasks**.

Step 8 Wait until the task is complete. After it is complete, the complete code is generated, and three process documents are generated: **spec.md**, **design.md**, and **tasks.md**.

----End

Slash Command Mode

CodeArts Agent encapsulates underlying skills to define four standard slash commands. Developers can execute commands as needed based on the existing process deliverables, without the need to re-execute the entire process. For example, if you find that the encryption library is outdated while executing the **/sdd-apply** command, manually modify the **design.md** document, and then run the **/sdd-tasks** and **/sdd-apply** commands in sequence.

Step 1 At the bottom of the input box on the chat panel, choose **Built-in** > **Agent** to switch to the agent mode. The selected model is displayed on the right. You can select a model as required from the drop-down list.

NOTE

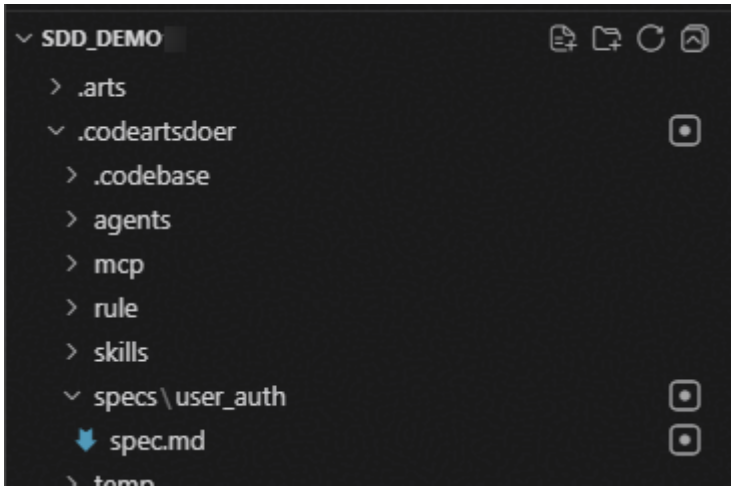
If the chat panel of CodeArts Agent is not displayed, click  (**Expand AI Sidebar**) in the upper right corner of the top menu bar to open it.

Step 2 On the chat panel, click **Vibe Coding**.

Step 3 Execute commands as needed to complete SDD.

- **/sdd-new: creates requirement specifications.**
  a. Enter a slash (/) in the dialog box and choose **/sdd-new** from the displayed menu. Enter the following requirement description to generate a **spec.md** document in a unified format:
  ```
  Help me develop a user registration and login module, including mobile number/email address registration, JWT-based login authentication, verification code sending, and password resetting.
  ```
  b. Go to *<project_root>***/.codeartsdoer/specs/***<output_folder>* to obtain the generated **spec.md**.

Figure 5-2 Directory of spec.md



c. Review the current requirement specification document. You can modify the **spec.md** document manually or using the agent until the requirement specification document meets the service expectations.

- **/sdd-design: creates a technical design document.**

  If you already have a **spec.md** document, you can directly execute **/sdd-design** and enter the output folder name (for example, **user_auth**) to generate a **design.md** document that contains the architecture and API description.

- **/sdd-tasks: creates a coding task list.**

  Enter a slash (/) in the dialog box, choose **/sdd-tasks** from the displayed menu, and enter the output folder name (for example, **user_auth**) to generate an executable task list **tasks.md**.

- **/sdd-apply: executes coding tasks.**

  Enter a slash (/) in the dialog box, choose **/sdd-apply** from the displayed menu, and enter the output folder name (for example, **user_auth**). CodeArts Agent executes coding tasks one by one based on the **tasks.md** document.

----End

Scenario-based Offerings

In actual development, the standard SDD workflow and slash command modes are not mutually exclusive. Instead, they can be flexibly switched based on different project phases and team roles. In the early stage of a new project, use the standard workflow to quickly set up the framework and establish a complete SDD documentation system. In the mid-term iteration, switch to the slash command mode and use the four commands for incremental development and partial refactoring.

Table 5-3 Core differences

Dimension	Standard SDD Workflow	Slash Command Mode
Execution mode	Enter requirements in the dialog box. The agent leads the process, and you can accept or reject the generated documents at key nodes.	Execute commands in the dialog box. You can independently drive the process.
Flexibility	The process is fixed and cannot be skipped.	Individual commands can be freely combined and repeatedly executed.
Application scenario	• New project initiation from scratch The team builds a complete SDD process from requirement analysis, system design, to the SDD document system. • Formal projects with high standardization requirements In scenarios with high requirements on consistency and verifiability, all development activities are carried out based on the specified specifications to ensure that the final functions and performance are highly consistent with the specifications. • Getting started with SDD Developers who are new to SDD can quickly establish a complete link through the step-by-step guidance of CodeArts Agent. • Projects with relatively stable requirements Requirements have been fully discussed, and the direction and solution will not be frequently adjusted during development.	• Projects with frequent requirement iterations or unclear requirements that need repeated confirmation If a product manager suddenly changes the login logic, the manager only needs to manually modify **spec.md**, and then executes **/sdd-design** and **/sdd-tasks** in sequence. There is no need to regenerate **spec.md**. • Multi-person asynchronous collaboration An architect creates **design.md** in the morning and submits it to Git. After a developer pulls the code in the afternoon, the developer only needs to execute **/sdd-tasks** and **/sdd-apply** in the chat box. • Partial documents available If you have manually written **spec.md** and want to start the design directly, simply type **/sdd-design** to proceed with the subsequent process.
Roles involved	Product managers quickly validate ideas. Independent developers work on small projects.	Architects review technical solutions. Team members collaborate to develop enterprise-level applications.

6 Code Check and Intelligent Defect Fixing Using CodeArts Check MCP

Overview

CodeArts Agent can connect to CodeArts through MCP for collaborative R&D. They can work together to create projects, set up code repositories, as well as commit and push code. Code check rules can be configured on CodeArts. CodeArts Agent receives check results through MCP, parses reports in a unified manner, and fixes code.

This practice focuses on CodeArts Agent and standardizes its interaction with CodeArts through MCP. The practice helps streamlines the entire process from code rule check, report analysis, to defect fixing, ensuring a unified and controllable R&D process.

6.1 Prerequisites

Before performing this practice, subscribe to a CodeArts package and configure permissions for each role.

This practice uses the Windows OS as an example.

Constraints

Table 6-1 Practice case restrictions

Category	Constraint
Region	This practice is applicable to: CodeArts (the AP-Singapore region is used as an example) CodeArts Agent (the CN-Hong Kong region is used as an example)
Tool	This practice supports only CodeArts Agent IDE.

Category	Constraint
OS	• Windows OS: **Windows 11 (x64) is recommended.** For Windows 10 (x64), the version must be 2019 or later, and upgrading to the latest stable version is advised. • macOS: macOS 11 or later, compatible with ARM64 (Apple Silicon). • Linux: Linux (ARM64) architecture, glibc version ≥ 2.31, glibcxx version ≥ 3.4.26.
Third-party software version	Node.js≥18.0.0

Purchasing a CodeArts Package

Step 1 **Sign up for a HUAWEI ID and enable Huawei Cloud services**.

Step 2 Use your HUAWEI ID to log in to the **Buy CodeArts package** page.

Step 3 Configure the CodeArts package parameters. In this practice, the **Free** package is used as an example.

Table 6-2 Purchasing a CodeArts package

Parameter	Value	Description
Region	AP-Singapore	The CodeArts package applies only in the region selected during purchase and cannot be used in other regions.
CodeArts Package	Select **Free**.	Edition of the CodeArts package
Users	50 (fixed value for the free edition)	Number of users included in the CodeArts package. The free edition has a fixed value of 50.
Required Duration	1 month	How long you will use the CodeArts package. The free edition has a fixed value of 1 month.

Parameter	Value	Description
Auto-renew	Enable	This parameter is optional. Enable it as needed. Once enabled, the package will be automatically renewed when it expires. • Required duration in months: The package will renew for 1 month each time. The number of renewal times is unlimited. • Required duration in years: The package will renew for 1 year each time. The number of renewal times is unlimited. For more details, see **Auto-Renewal Rules**.

Step 4 Click **Enable**.

After purchasing the free package with a HUAWEI ID, proceed with **Setting a Project Creator**.

----End

Setting a Project Creator

Step 1 Open the CodeArts homepage using your HUAWEI ID.

1. Log in to the **CodeArts console** using your HUAWEI ID, click , and select **AP-Singapore**.
2. Click **Go to Workspace** in the upper right corner.

Step 2 Click the username on the top navigation bar and choose **All Account Settings**.

Step 3 Choose **General** > **Project Creators**.

Step 4 Select **All members can create projects**. After that, all IAM users in your account can create projects.

After setting the project creator, proceed with **Creating a CodeArts Project and Setting CodeCheck Rules**.

----End

6.2 Creating a CodeArts Project and Setting CodeCheck Rules

Creating a Project and a Code Repository

Step 1 Log in to the **CodeArts console** using an account with the project creation permission and select **AP-Singapore**.

Step 2 Click **Go to Workspace** in the upper right corner to go to the CodeArts project homepage.

Step 3 Click **Create** > **Create Project**. The page for creating a project is displayed.

Step 4 On the CodeArts homepage, find the **Scrum** template and click **Select**.

Step 5 Enter a project name (for example, **req-demo**), and click **OK**. Keep the name under 128 characters.

The project is created, and the **Work Items** page of CodeArts Req is displayed. **By default, the project creator is the project manager.**

Step 6 In the navigation pane, choose **Code** > **Repo** to access CodeArts Repo.

Step 7 Click **Create Repository**.

Step 8 Select **Template** and click **Next**.

Step 9 Select the **Java Web Demo** template and click **Next**.

Step 10 Enter a repository name (for example, **repo-demo**). Start the name with a letter, digit, or underscore (_), and use letters, digits, hyphens (-), underscores (_), and periods (.). Do not end the name with **.git**, **.atom**, or a period.

Step 11 Select **Public** > **For project members** under **Visibility**, and click **OK**.

The repository is created, and its **Code** page is displayed.

Step 12 Choose **Code** > **Branches** and click **Create Branch**. In the displayed dialog box, set the following parameters and click **OK**.

A message is displayed, indicating that the operation is successful. The **Files** page is displayed, showing the **develop-dev** branch.

Table 6-3 Setting parameters for creating a branch

Configuration Item	Example	Description
Based On	master	Select an existing branch.
Branch Name	develop-dev	Name of the branch you are creating. The name cannot exceed 200 bytes. • Do not start with a hyphen (-), period (.), **refs/heads/**, or **refs/remotes/**. • Do not use spaces or these special characters: [\<~^:?*!()'"| • Do not end with a period (.), slash (/), or **.lock**.
CodeArts Req Work Items to Associate	/	Associate an existing work item with the branch. By default, no Req work item is associated.

Step 13 Click **Settings** on the top navigation bar, choose **Security Management** > **Permissions** in the left navigation pane, and enable **Use Project Permissions** to use project-level permissions.

----End

Configuring Code Check Rules

Step 1 Log in to the **CodeArts console** as a project manager, and select **AP-Singapore**.

Step 2 Click **Go to Workspace** in the upper right corner to go to the CodeArts project homepage.

Step 3 Click the **req-demo** project to go to the homepage of the project.

Step 4 In the navigation pane, choose **Code** > **Check**. The CodeArts Check homepage is displayed.

Step 5 On the **Tasks** tab, click ••• in the row where repo-demo-check is located, and click **Settings**. The **Basic Info** page of repo-demo-check is displayed.

Step 6 On the **Basic Info** page, set **Default Branch** to **develop-dev** and click **Save**.

Step 7 On the **Check Scope** page, set **Version Check** to **Last Commit** and click **Save**.

Step 8 On the **Integration Service** page, select **Triggered upon code commit** and click **Save**.

----End

6.3 Developing Code

Creating a Branch and Cloning a Repository

Developers need to develop code for new requirements based on the develop-dev branch.

Step 1 Open the **req-demo** project and choose **Code** > **Repo** from the navigation pane to access CodeArts Repo.

Step 2 Click the **repo-demo** repository.

Step 3 Open the local Git Bash, and run the following command to clone the develop-dev branch to the local PC:

```
git clone -b Branch name https:example.com
```

In the preceding command, *Branch name* indicates the name of the branch to be cloned, and *https:example.com* indicates the HTTPS address of the code repository.

If the following information is displayed, the develop-dev branch of the code repository repo-demo has been successfully cloned to the local PC:

```
$ git clone -b develop-dev https://test.com/bb5f2eb7ded44e0bbc0ff4bd3d1d244c/repo-demo.git
Cloning into 'repo-demo'...
remote: Enumerating objects: 36, done.
remote: Counting objects: 100% (36/36), done.
remote: Compressing objects: 100% (30/30), done.
```



```
remote: Total 36 (delta 0), reused 36 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (36/36), 301.74 KiB | 1.23 MiB/s, done.
```

----End

Coding

Step 1 In the upper left corner of CodeArts IDE, choose **File** > **Open Project** to open the created code repository **repo-demo**.

Step 2 In the input box, enter the following information and click  :

```
A path in a binary tree is defined as a node sequence, with an edge exists between every two adjacent nodes in the sequence. A node can appear only once in a path. The path must contain at least one node and does not necessarily pass through the root node. The path sum is the sum of the values of all nodes in the path. Based on this project, provide the root node of a binary tree, and return the maximum path sum. If you have any questions, feel free to ask me.
```

Step 3 After the code is generated, enter the following information, compile and run the verification script using Java and javac, and click the send icon  .

```
Write a verification script and use Java and javac to compile and run it.
```

Step 4 After the code is generated, choose **Terminal** > **New Terminal** in the navigation pane and run the following command:

```
powershell -ExecutionPolicy Bypass -File run-tests.ps1
```

In the preceding command, replace *run-tests.ps1* with the script file to be run. In this document, the script file name is **run-tests.ps1**.

If the following information is displayed, the code can run properly.

```
==========================================
  Maximum Path Sum in a Binary Tree - Test and Verification
==========================================

[PASS] example1 [1,2,3] => 6
[PASS] example2 [-10,9,20,null,null,15,7] => 42
[PASS] allNegative [-1,-2,-3] => -1
[PASS] leftSkewedTree [5,4,null,3,null,2] => 14
[PASS] negativeBranchesIgnored [2,-1,-2] => 2
[PASS] complexTree => 1

==========================================
  Results: 9 passed, 0 failed
==========================================

All tests are passed.
```

NOTE

The actual command output prevails.

----End

6.4 Checking Code

Committing code

Step 1 Right -click repo-demo code project and choose **Open in Terminal** from the shortcut menu. Run the following command in the terminal to verify that the develop-dev branch is modified:


```
git branch
```

If the following information is displayed, the current branch is correct:

```
* develop-dev
```

Step 2 After the verification is successful, run the following command in the terminal to view the local code modifications:

```
git status
```

The following information is displayed, in which the content in red indicates the modified files that are not saved. Confirm the code to be committed.

```
On branch develop-dev
Your branch is up to date with 'origin/develop-dev'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        run-tests.ps1
        src/main/java/com/huawei/codearts/controller/TreeController.java
        src/main/java/com/huawei/codearts/tree/
        src/test/
        target/

nothing added to commit but untracked files present (use "git add" to track)
```

Step 3 Run the following command in the terminal to add the modifications to the staging area:

```
git add .
```

If no command output is displayed, the modified files are successfully committed to the staging area.

Step 4 Run the following command in the terminal to commit the code locally:

```
git commit -m "feat: Add the code for the maximum path of the binary tree and the test execution script."
```

If the following information is displayed, the code is successfully committed:

```
[develop-dev e1256e8] feat: Add the code for the maximum path of the binary tree and the test execution
script.
 Committer: chenzhu (F) <test.com>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 10 files changed, 295 insertions(+)
 create mode 100644 run-tests.ps1
 create mode 100644 src/main/java/com/huawei/codearts/controller/TreeController.java
 create mode 100644 src/main/java/com/huawei/codearts/tree/MaxPathSum.java
 create mode 100644 src/main/java/com/huawei/codearts/tree/MaxPathSumRunner.java
 create mode 100644 src/main/java/com/huawei/codearts/tree/TreeNode.java
 create mode 100644 src/test/java/com/huawei/codearts/tree/MaxPathSumTest.java
 create mode 100644 target/test-classes/com/huawei/codearts/tree/MaxPathSum.class
 create mode 100644 target/test-classes/com/huawei/codearts/tree/MaxPathSumManualTest.class
 create mode 100644 target/test-classes/com/huawei/codearts/tree/MaxPathSumRunner.class
 create mode 100644 target/test-classes/com/huawei/codearts/tree/TreeNode.class
```

Step 5 Run the following command to push the code to the develop-dev branch on the remote cloud:

```
git push origin develop-dev
```


If the following information is displayed, the local code has been successfully committed to the cloud repository repo-demo.

```
Enumerating objects: 40, done.
Counting objects: 100% (40/40), done.
Delta compression using up to 16 threads
Compressing objects: 100% (18/18), done.
Writing objects: 100% (32/32), 7.24 KiB | 494.00 KiB/s, done.
Total 32 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Start Git Hooks Checking                                [PASSED]
remote:
remote: To create a merge request for develop-dev, visit:
remote:   https://test.com/codehub/2112135977/newmerge?branchName=develop-
dev&projectNamespace=bb5f2eb7ded44e0bbc0ff4bd3d1d244c/repo-demo
remote:
To https://test.com/bb5f2eb7ded44e0bbc0ff4bd3d1d244c/repo-demo.git
  2f31d8d..e1256e8  develop-dev -> develop-dev
```

After the code is successfully committed, CodeArts Check automatically triggers a check task.

----End

Viewing Check Results

Step 1 Log in to the **CodeArts console**, and select **AP-Singapore**.

Step 2 Click **Go to Workspace** in the upper right corner to go to the CodeArts homepage.

Step 3 Click the **req-demo** project to go to the homepage of the project.

Step 4 In the navigation pane, choose **Code** > **Check**. The CodeArts Check homepage is displayed.

Step 5 On the **Tasks** tab, click **repo-demo-check**. The **Overview** page of repo-demo-check is displayed.

Step 6 Click the branch name, click **Commit**, and select the commit record to be checked.

Step 7 Click **Issues** in the navigation bar.

As shown in the following figure, 72 issues are detected in the latest commit.

----End

6.5 Configuring CodeArts Check MCP

Step 1 In the upper left corner of IDE, choose **File** > **Open Project**, and open the **repo-demo** project.

Step 2 Click  in the upper right corner of CodeArts Agent IDE to open the settings page.

Step 3 In the navigation pane on the left, choose **MCP**.

Step 4 Download CodeArts Check MCP from **official download website** to the local PC and decompress it.

Step 5 Click **Configure MCP**, copy the following configuration information, and paste it to the **mcp_settings.json** file. For details about the parameters and their example values, see **Table 6-4**.

After the configuration is complete, press shortcut keys (**Ctrl+S** for Windows/Linux and **Command (⌘)+S** for macOS) to save the settings and wait until the tool is started.

```
{
  "mcpServers": {
    "CodeArts Check MCP": {
      "disabled": false,
      "timeout": 600000,
      "command": [
        "D:\\Program Files\\nodejs\\node.exe",
        "D:\\codeartscheck-ts-mcp-server-1.0.0\\package\\dist\\index.js"
      ],
      "environment": {
        "region": "ap-southeast-3",
        "ak": "your_ak",
        "sk": "your_sk"
      },
      "type": "local"
    }
  }
}
```

Table 6-4 Parameters

Parameter	Example Value	Description
mcpServers	See the configuration information below.	Top-level node, which contains all registered MCP servers.
CodeArts Check MCP	/	(Mandatory) A unique name (maximum 64 characters) used to distinguish between different MCP servers. **NOTE** If the server name exceeds 64 characters, the request may fail with one of the following errors. For details, refer to the definition of **Tools** under **Core Server Features**. • type: session.error info • length over limit, maximum is 64
disabled	false	(Mandatory) Whether to disable the MCP server. The default value is **false**. Value: • false: enabled • true: disabled
timeout	600000	Timeout interval (ms). After the timeout interval expires, the client reports an error.

Parameter	Example Value	Description
command	"command": ["D:\\Program Files\\nodejs\\node.exe", "D:\\codeartscheck-ts-mcp-server-1.0.0\\package\\dist\\index.js"],	(Mandatory) It consists of two parts: • "D:\\Program Files\\nodejs\\node.exe": path of the executable file for running the MCP server. • "D:\\codeartscheck-ts-mcp-server-1.0.0\\package\\dist\\index.js": path for storing the **index.js** file after the package obtained in **Step 4** is decompressed.

environment	{
	"region": "ap-southeast-3",
	"ak": "your_ak",
	"sk": "your_sk"
	}
		(Optional) Environment variables, including areas, AK, and SK, passed to the MCP server. These variables must be strings.
		The value of **environment.region** must be fixed to **ap-southeast-3**. The methods of obtaining **environment.ak** and **environment.sk** are as follows:
		1. Create access keys by referring to **Access Keys**.
		2. In the **Create Access Key** dialog box, select the agreement check box and click **Next**. Enter a description for the access key and click **OK**.
		After the access key is created, download and save it. Once the dialog box is closed, the key cannot be obtained.
		3. You can obtain the values of **environment.ak** and **environment.sk** from the **credentials.csv** file that you have downloaded and saved.

type	local	Communication protocol of the MCP server. The value is **local**.

Step 6 Click  in the upper right corner of CodeArts Agent IDE to open the settings page.

Step 7 In the navigation pane on the left, choose **MCP**.

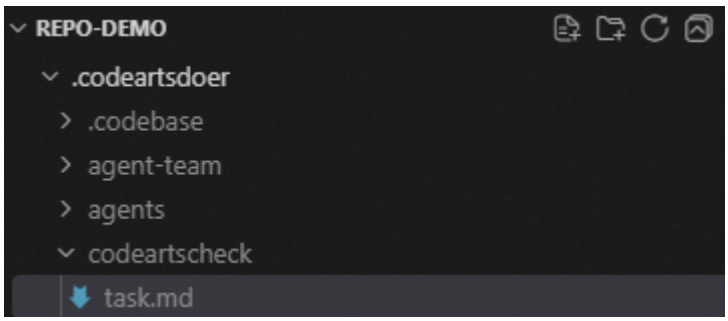
Step 8 On the **Installed** tab, the MCP server CodeArts Check MCP has been configured and takes effect, as shown in the following figure.

----End

Locally Configuring a Check Task ID

Step 1 As shown in the following figure, create a **codeartscheck** folder under .**codeartsdoer** and create a **task.md** file in the **codeartscheck** folder.

Figure 6-1 Creating a task.md file



Step 2 Log in to **CodeArts console**, and select **AP-Singapore**.

Step 3 Click **Go to Workspace** in the upper right corner to go to the CodeArts project homepage.

Step 4 Click the **req-demo** card to go to the homepage of the project.

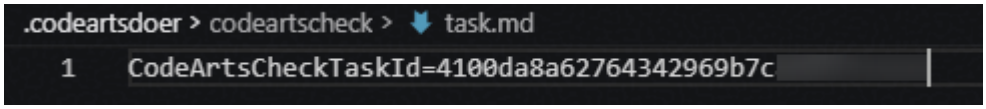
Step 5 In the navigation pane, choose **Code** > **Check** to go to the CodeArts Check homepage.

Step 6 On the **Tasks** tab, click **repo-demo-check** to go to the **Overview** page of the scan task repo-demo-check.

Step 7 Copy the ID of the scan task (that is, the character string between "task" and "detail" in the URL of the current browser).

Step 8 As shown in the following figure, enter the task ID in the **task.md** file and save the file.

Figure 6-2 Filling in the task.md file



----End

6.6 Obtaining the Scan Report and Fixing the Code

Obtaining the Security Scan Report of the Latest Commit and Fixing the Code

Step 1 In the upper left corner of CodeArts Agent IDE, click **Space** to switch to CodeArts Agent Space.

Step 2 Click **Coding**.

Step 3 At the bottom of the input box, select **AgentTeam** to switch to the agent team mode. The selected model is displayed on the right. You can select a model as required from the drop-down list box.

Step 4 Enter the following prompt in the dialog box and click  or press **Enter**:

```
Obtain the security scan report of the latest commit and fix the code.
```

CodeArts Agent IDE will call CodeArts Check MCP to return the path of the obtained report. You can view the code check report locally. The built-in sub-agent will continue to fix the code. Wait until the code fix is complete.

----End

Committing the Fixed Code

Step 1 In the upper left corner of CodeArts Agent IDE, click **IDE** to switch to IDE.

Step 2 Run the following command in the terminal to view the local modifications and confirm the modified files:

```
git status
```

The following information is displayed, indicating the local modifications.

```
On branch develop-dev
Your branch is up to date with 'origin/develop-dev'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   src/main/java/com/huawei/codearts/controller/TreeController.java
        modified:   src/main/java/com/huawei/codearts/tree/MaxPathSum.java
        modified:   src/main/java/com/huawei/codearts/tree/MaxPathSumRunner.java
        modified:   src/main/java/com/huawei/codearts/tree/TreeNode.java
        modified:   src/test/java/com/huawei/codearts/tree/MaxPathSumTest.java

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .agent-team/
        .arts/
```

Step 3 Run the following command on the terminal to pull the latest code from the remote branch:

```
# Pull updates from all remote branches.
git fetch origin
# Merge the latest code of the remote develop-dev branch to the current local branch.
git pull origin develop-dev
```

- If the following information is displayed, no conflict occurs. Proceed with **Step 4**.

  ```
  nothing added to commit but untracked files present (use "git add" to track)
  From https://test.huawei.com/bb5f2eb7ded44e0bbc0ff4bd3d1d244c/repo-demo
   * branch            develop-dev -> FETCH_HEAD
  Already up to date.
  ```

- If a conflict occurs, the terminal displays the conflicting file. You can manually open the file to modify the conflicting content. After the modification, run the following command and then go to **Step 4**.

  ```
  git add .
  git commit -m "fix: Merge the remote code conflicts."
  ```

Step 4 Run the following command in the terminal to add the modifications to the staging area:

```
git add .
```


If no command output is displayed, the modified files are successfully submitted to the staging area.

Step 5 Run the following command in the terminal to commit the code locally:

```
git commit -m "fix: Fix the scanned code issues."
```

If the following information is displayed, the code is successfully committed:

```
[develop-dev 55878b5] fix: Fix the scanned code issues.
 Committer: chenzhu (F) <test.com>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 9 files changed, 670 insertions(+), 24 deletions(-)
 create mode 100644 .agent-team/ses_07c6a4e4fffe71tWtQ6qZI4mD2/codefix/
origin_report_4545383827873645935/index.json
 create mode 100644 .agent-team/ses_07c6a4e4fffe71tWtQ6qZI4mD2/codefix/
origin_report_4545383827873645935/subReport_0.md
 create mode 100644 .agent-team/ses_07c6a4e4fffe71tWtQ6qZI4mD2/codefix/
origin_report_4545383827873645935/subReport_1.md
 create mode 100644 .arts/settings.json
```

Step 6 Run the following command to push the code to the develop-dev branch on the remote cloud:

```
git push origin develop-dev
```

If the following information is displayed, the local code has been successfully committed to the cloud repository repo-demo.

```
Enumerating objects: 47, done.
Counting objects: 100% (47/47), done.
Delta compression using up to 16 threads
Compressing objects: 100% (18/18), done.
Writing objects: 100% (30/30), 6.25 KiB | 112.00 KiB/s, done.
Total 30 (delta 6), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Start Git Hooks Checking                                    [PASSED]
remote:
remote: To create a merge request for develop-dev, visit:
remote:   https://test.com/codehub/2112135977/newmerge?branchName=develop-
dev&projectNamespace=bb5f2eb7ded44e0bbc0ff4bd3d1d244c/repo-demo
remote:
To https://test.com/bb5f2eb7ded44e0bbc0ff4bd3d1d244c/repo-demo.git
   e1256e8..55878b5  develop-dev -> develop-dev
```

After the code is successfully committed, CodeArts Check automatically triggers a check task on the cloud.

After the scan is complete, view the latest check report. For details, see **Viewing Check Results**. As shown in the following figure, five issues remain to be fixed.

----End

Fixing Code for the Second Time

Step 1 Use CodeArts Agent IDE to obtain the report and fix the code by referring to **Obtaining the Security Scan Report of the Latest Commit and Fixing the Code**.

Step 2 After the second code modification, commit the modified code by referring to **Committing the Fixed Code**.

After the code is successfully committed, CodeArts Check automatically triggers a check task on the cloud.

After the scan is complete, view the latest check report. For details, see **Viewing Check Results**. As shown in the following figure, all issues have been fixed.

If there are still issues that have not been fixed in the report, refer to **Obtaining the Security Scan Report of the Latest Commit and Fixing the Code**.

----End

6.7 Creating a Merge Request

After completing code development, developers can submit a branch merge request. The committer reviews the request and merges the branches.

Step 1 The developer submits a merge request.

1. Log in to the **CodeArts console**, and select **AP-Singapore**.
2. Click **Go to Workspace** in the upper right corner to go to the CodeArts project homepage.
3. Click the **req-demo** project to go to the homepage of the project.
4. In the navigation pane, choose **Code** > **Repo**. On the CodeArts Repo homepage, click the code repository name **repo-demo**. The **Code** page is displayed.
5. Click **Merge Requests** on the top navigation bar.
6. Click **Create Merge Request**. Select **develop-dev** for **Source Branch** and **master** for **Target Branch**. Specify **Title** and click **OK**.

Step 2 The committer reviews the merge request and merges the code.

1. Open the **Merge Requests** page of the code repository repo-demo by referring to the steps **Step 1.1** through **Step 1.5**.
2. Click the title of the target merge request. The **Details** page is displayed.
3. Click **Merge** in the upper right corner to merge the code.

----End

6.8 Related Documents

Failed to Obtain the Code Security Scanning Report

Symptom

When a user attempts to obtain the scanning report, IDE displays an error message indicating that the CodeArts Check task has been submitted but fails to be executed due to invalid IAM authentication information.

Solution

This issue occurs because the values of **environment.ak** and **environment.sk** in the **mcp_settings.json** file are incorrect.

If the issue persists after you change the values of **environment.ak** and **environment.sk** in the **mcp_settings.json** file, **submit a service ticket** to get help from Huawei Cloud customer service.

7 Generating Common Logic Code

Many bread-and-butter tasks, such as regular expressions (regex) and time processing, have straightforward logic but thorny implementation details. Their complex coding rules require developers to wade through relevant documentation.

You can let CodeArts Agent do the heavy lifting for common algorithms, freeing yourself to tackle intricate logic.

Generating Regex or String Processing Functions

Step 1 In IntelliJ IDEA, write language-specific comments.

```
/*
Use a regular expression to extract all numeric characters from the input string and return them as an array of digits.
 */
```

Even when the function logic is clear (in this example), crafting a regex often turns into a documentation scavenger hunt. That switching burns time.

Step 2 Place your cursor after the comments and press **Alt+C** to generate the matching processing functions. Then, you can fine-tune the generated code to align with service needs.

If the initial generation is incomplete, press **Tab** to accept what exists. Reinvoke generation with **Alt+C** to append additional code. Repeat this action until the code snippet is fully fleshed out.

----End

Generating Date Processing Functions

Step 1 In IntelliJ IDEA, write language-specific comments.

```
/*
Generate a function that determines whether a given year qualifies as a leap year.
 */
```

Step 2 Place your cursor after the comments and press **Alt+C** to generate the matching processing functions. Then, you can fine-tune the generated code to align with service needs.

If the initial generation is incomplete, press **Tab** to accept what exists. Reinvoke generation with **Alt+C** to append additional code. Repeat this action until the code snippet is fully fleshed out.

----End

8 Generating Code That Mirrors an Existing Implementation

In API architectures, the "Controller" layer is often decoupled from complex service logic. Within a single service, APIs are highly similar. This high degree of uniformity makes it possible to extend service APIs by using existing APIs as blueprints.

Generating Code from Comments

API implementations can share similar structures, as shown in the following figure.

```
@Override
@OperationLog(objectId = "#roleId", objectType = "AuthRole", details = "")
public RetObj deleteRole(String roleId) {
    try {
        return authRoleService.deleteRole(roleId, DevCloudTokenStore.getUserId().toLowerCase(Locale.ENGLISH));
    } catch (Exception e) {
        return RetObjUtil.returnErrorMessage(e.getMessage());
    }
}

@Override
@OperationLog(objectId = "#params.resourceId", objectType = "AuthResource", details = "")
public RetObj addAuthResoure(AuthResourceParam params){
    try {
        return resourceService.addAuthResoure(params, DevCloudTokenStore.getUserId().toLowerCase(Locale.ENGLISH));
    } catch (Exception e) {
        return RetObjUtil.returnErrorMessage(e.getMessage());
    }
}
```

You can add the following comments:

```
/**
 * Method for modifying permission resources
 * @param params input parameters
 * @return Results returned after permission resource modification
 */
```

The following figure shows the comment-driven generation results.

```
@Override
@OperationLog(objectId = "#params.resourceId", objectType = "AuthResource", details = "")
public RetObj updateAuthResource(AuthResourceParam params) {
```


Summary

In this section, the project file already contains clean, well-organized code, which provides context for completion. By adding targeted comments, you can let the tool generate new, structurally aligned code blocks.

9 Compiling Database APIs

MyBatis, as a common database framework, often requires the creation of numerous API classes. In practice, these classes typically share consistent structure, making them ideal candidates for auto-generation. By leveraging existing code, you can rapidly produce new service APIs that align with established context.

Generating Database API Code from Comments

```
// Obtain the IdList of L0
List<ToolAllListCkBillEntity> getL0IdList(@Param("type") String type, @Param("start_time") String
startTime,@Param("end_time") String endTime);
// Obtain the IdList of L1
List<ToolAllListCkBillEntity> getL1IdList(@Param("type") String type, @Param("start_time") String
startTime,@Param("end_time") String endTime);
```

For services with similar logic, the related APIs can be generated automatically.

Summary

This tool can learn the style of the existing codebase and extend it by generating consistent, matching code. With this feature, you can code faster.

10 Querying Unknown Dependencies

When you are unsure about what dependencies or packages your code needs, let CodeArts Agent take the lead. It works right in your Integrated Development Environment (IDE) to help you find the answers, so you do not need to search online.

Procedure

Step 1 Log in to CodeArts Agent by referring to **Quick Start**.

Step 2 Open the chat panel of CodeArts Agent.

Step 3 Type your request into the chat box. For example, ask this question:

```
I have integrated the following APIs into my code. Could you guide me on what dependencies I need to add to the POM file?
import org.junit.Before;
import org.junit.Test;
```

Step 4 Click  .

Step 5 Verify CodeArts Agent's response in the chat panel.

The graph shown here is for reference only. The actual output may vary.

Figure 10-1 Response result

----End

11 Directly Importing Components

For common third-party components, you can ask CodeArts Agent to give you common import methods and related code templates.

Procedure

Step 1 Log in to CodeArts Agent by referring to **Quick Start**.

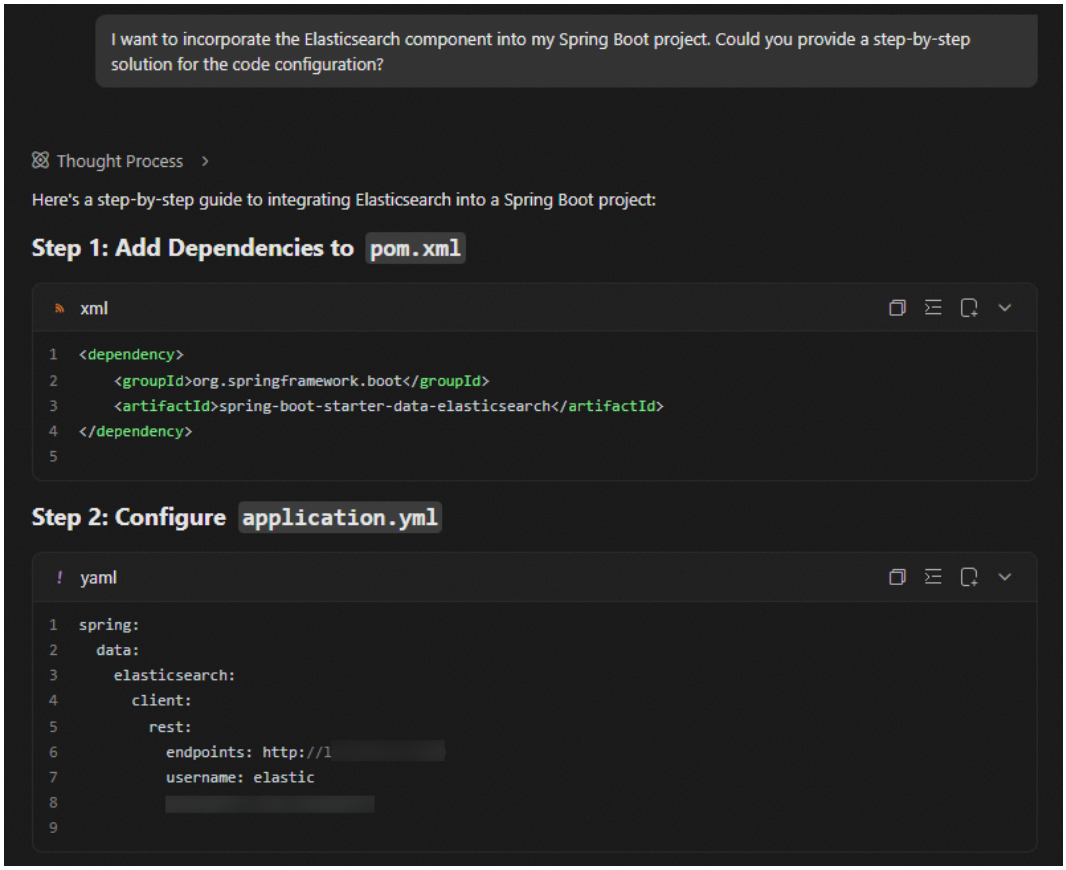
Step 2 Type your request into the chat box of CodeArts Agent. For example, enter this instruction "I want to incorporate the Elasticsearch component into my Spring Boot project. Could you provide a step-by-step solution for the code configuration?"

Step 3 Click  .

Step 4 Verify CodeArts Agent's response in the chat panel.

The graph shown here is for reference only. The actual output may vary.

Figure 11-1 Response result



----End

12 Writing a Good Skill

Without skills, a team ends up with everyone reinventing the wheel. Engineers' prompts are scattered everywhere, project managers (PMs)' requirement statements make the agent guess repeatedly, and executives' experience evaporates quietly with personnel turnover. At first glance, engineers, PMs, and executives care about different things. However, when it comes to using agent tools, they all face the same dilemma: experience cannot be accumulated, and quality depends on individuals.

Essence of skills — encapsulating tacit knowledge into code: To understand a skill, you need to understand that it solves a knowledge management problem, not just an AI tool problem.

There is a classic concept in management: tacit knowledge. An engineer knows how to communicate with the agent to obtain high-quality code review. This is tacit knowledge. The engineer can do it, but cannot clearly explain it, and it is difficult to directly teach it to others. In contrast, explicit knowledge is written in documents and can be read and spread.

The goal of knowledge management in an organization is to convert as much tacit knowledge as possible into explicit knowledge. The traditional approach is to write documents, create SOPs, and conduct training. However, these methods have a fatal flaw: documents will expire, SOPs will be bypassed, and the training effect will be difficult to quantify. More importantly, documents describe how people should do things, not how agents should do things. This distinction is critical in today's world where agents are heavily involved in workflows.

Skills are a more thorough solution. It is not a document describing best practices, but rather **it directly encodes best practices into the execution path of the agent**. When you write the standard framework for competitive product analysis as a skill, anyone in the team can open the agent and say, "Help me analyze this competitive product," and the agent will execute the task according to the verified framework, rather than improvising.

Compared with common prompts, skills have three core differences, as described in **Table 12-1**.

Table 12-1 Comparison between skills and common prompts

Dimension	Common Prompt	Skill
Storage location	Personal notes or chat history.	Team shared library, and structured storage
Activation method	Manually pasted by users	Automatically loaded by agents as needed
Iteration mechanism	No version management, making it difficult to track changes	Version-based management and rollback

This difference means that skills are not just better prompts. They are also a form of institutionalized knowledge storage, which can turn individual wisdom into organizational capabilities.

Case study: How standardization happens

A product team has eight project managers, and competitive product analysis reports are frequently produced in their daily work. Before skills were introduced, the eight project managers had different report styles. Some were good at writing function comparisons, some focused on user experience, and some emphasized business models. The quality of the reports was highly inconsistent, and the executives had to adapt to a new framework each time they read the reports. A seasoned project manager in the team spent two days encapsulating their product analysis framework into a skill. This skill covered four dimensions: function matrix, user journey differences, pricing strategy, and market positioning. Each dimension had a standard analysis perspective and output format.

Three weeks after the skill was launched, the changes were very specific:

- The entire team began using a unified framework for product analysis reports, reducing the average time executives spent reading reports from 20 minutes to 8 minutes.
- The two new project managers were able to produce analysis reports of equivalent quality to those of the experienced project managers within the first week.
- The experience of project managers was no longer "an individual's affair" but became a fundamental capability of the team.

This is the path through which skills enable standardization: **instead of relying on training and regulations to constrain people, the best practices are directly embedded into tools**, so that each use naturally follows the optimal path.

Understanding a skill does not require reading the code first. It is more like a job description given to the agent: telling it when to start work and how to perform its duties. This section breaks down the three core components of a skill and demonstrates the three-layer architecture of progressive loading.

Three core components of a skill: A simple skill only needs three elements: name, description, and execution instructions.

- Name (skill identifier): A name is the unique ID of a skill, and is also the way it is referenced in the system and logs. The naming principle is simple: verb + noun, clearly expressing what the skill does. "Review-code", "analyze-competitor", and "generate-prd" are good names. "Helper", "assistant", and "tool1" are bad names. Vague names will become a maintenance nightmare in the skill library.
- Description (skill description): The description is the most critical and often underestimated part of a skill. When deciding whether to activate a skill, the agent relies solely on the description. A vague description is useless. If the description fails to trigger the skill in the right scenario, even the most sophisticated execution instructions are useless.
- Execution instructions: specific execution scripts. Instructions are what the agent actually executes after a skill is activated. They can be a structured step-by-step description, or they can include specific analysis frameworks, output format requirements, and ways to handle boundary conditions. The instructions are like a detailed operation manual for the agent. The more specific they are, the more stable the execution will be.

Take a code review skill as an example. The three elements of a complete skill are as follows:

```
yaml
name: review-code

description: |
  Activate the skill when a user submits a code review request or requests an evaluation of code quality.
  Applicable scenarios include pull request review, single-file code quality check, and quality baseline evaluation before refactoring.
  It is not applicable to code debugging or function development.

instructions: |
  ## Code review procedure

  ### 1. Evaluate the overall structure.
  - Whether the module division is clear
  - Whether the naming conventions are consistent
  - Whether the comment coverage meets the requirements

  ### 2. Check the following dimensions one by one:
  - **Readability**: Whether the code is easy to understand and whether complex logic is commented.
  - **Security**: Whether there are common vulnerabilities (such as SQL injection and XSS)
  - **Performance**: Whether there are obvious performance issues
  - **Test coverage**: Whether there are tests for key paths

  ### 3. Output format
  Output the review report in the following structure:
  - Overall score (1–10)
  - Items that must be modified (Blocker)
  - Items recommended for improvement (Suggestion)
  - Highlights
```

Three-layer architecture for progressive loading: A crucial engineering decision in skill design is that not all content needs to permanently reside in the context window of the agent. A team may have dozens or even hundreds of skills. If the content of every skill permanently resides in the context, it will cause severe context pollution and performance degradation. Skills adopt a three-layer architecture for progressive loading to address this issue.

- Layer 1: Metadata, which contains only the name and description, with a very small size, and is always within the agent's view. The agent relies on this layer to determine whether there are suitable skills available.

- Layer 2: **SKILL.md** body, which contains complete execution instructions. The context is read only after the skill is activated (that is, the description is successfully matched). This layer is the main body of the skill and contains all the instructions on how to do something.
- Layer 3: Bundled resources, which are auxiliary materials such as scripts, reference documents, and output templates. These resources are loaded only when needed during execution, preventing unnecessary context consumption.

The advantage of this architecture is that the increase in the number of skills will not linearly increase the context burden of the agent. For example, if a team has 50 skills, the agent's resident context contains only 50 lightweight description abstracts. Only the one or two skills activated for the current task consume the context.

Additional resource types and design principles: In addition to the three basic elements, the third layer of a skill can carry three types of resources, each with its own applicable scenarios.

- Automation scripts: They are suitable for processing fixed, mechanical operations that require no decision-making. For example, a skill for generating API documentation can be accompanied by a script that automatically extracts API information from code comments. The agent can directly invoke this script during execution, eliminating the need to manually describe the extraction logic each time.
- Reference documents: They are suitable for skills that require extensive background knowledge. For example, a compliance check skill can be accompanied by an abstract of the company's compliance manual, and a technical solution review skill can be accompanied by architecture design specifications. It is recommended that a table of contents be added to long reference documents to facilitate targeted retrieval by the agent, instead of scanning the entire document.
- Output templates: They are suitable for scenarios with strict requirements on the output format. For example, a weekly report generation skill can be accompanied by a fixed Markdown template, and a competitive product analysis skill can be accompanied by a standard table structure. Templates can significantly reduce the agent's "freedom" in formatting, making the outputs more consistent.

