Prompts 解析流程设计思路

针对 `IPrompt_v3_for_promptsFactory[]` 数据结构的解析，我们可以采用以下几种思路来将其转化为精准的大模型提示词：

1. 模板化拼接法

根据不同的类型，使用预定义的模板字符串进行拼接：

```
function parsePrompts(prompts: IPrompt_v3_for_promptsFactory[]): string
  const selectedPrompts = prompts.filter(p => p.status === 'selected');
  let finalPrompt = '';

  // 按类型分组并按特定顺序拼接
  const context = selectedPrompts.filter(p => p.type === EPrompt_v3_for_
  if (context.length) finalPrompt += `## Background\n${context.map(p =>

  const role = selectedPrompts.filter(p => p.type === EPrompt_v3_for_pro
  if (role.length) finalPrompt += `## Your Role\n${role.map(p => p.title

  // 依此类推处理其他类型...

  return finalPrompt;
}
```

2. 结构化组装法

将不同类型的提示组织成结构化的JSON对象，然后根据预定义的模式转换为文本：

```
function parsePrompts(prompts: IPrompt_v3_for_promptsFactory[]): string
  const selectedPrompts = prompts.filter(p => p.status === 'selected');
  const promptStructure = {
    context: selectedPrompts.filter(p => p.type === EPrompt_v3_for_promp
    role: selectedPrompts.filter(p => p.type === EPrompt_v3_for_promptsF
```

```
    // 其他类型...
  };

  return convertStructureToText(promptStructure);
}
```

3. 优先级排序法

根据不同类型的重要性和逻辑顺序进行排序，然后拼接：

```
function parsePrompts(prompts: IPrompt_v3_for_promptsFactory[]): string
  const selectedPrompts = prompts.filter(p => p.status === 'selected');

  // 定义类型优先级
  const typePriority = {
    [EPrompt_v3_for_promptsFactory_type.CONTEXT]: 1,
    [EPrompt_v3_for_promptsFactory_type.ROLE]: 2,
    [EPrompt_v3_for_promptsFactory_type.OBJECTIVE]: 3,
    // 其他类型优先级...
  };

  // 按优先级排序
  selectedPrompts.sort((a, b) => typePriority[a.type] - typePriority[b.t

  // 拼接最终提示词
  return selectedPrompts.map(p => `[${p.type}]: ${p.title}`).join('\n\n'
}
```

4. 条件动态组装法

根据存在的提示类型动态调整组装策略：

```
function parsePrompts(prompts: IPrompt_v3_for_promptsFactory[]): string
  const selectedPrompts = prompts.filter(p => p.status === 'selected');
  let finalPrompt = '';
```

```
  // 检查是否有特定类型
  const hasRole = selectedPrompts.some(p => p.type === EPrompt_v3_for_pr
  const hasObjective = selectedPrompts.some(p => p.type === EPrompt_v3_f

  // 根据存在的类型动态调整模板
  if (hasRole && hasObjective) {
    // 角色导向的任务模板
    finalPrompt = constructRoleBasedPrompt(selectedPrompts);
  } else if (hasObjective) {
    // 目标导向的任务模板
    finalPrompt = constructObjectiveBasedPrompt(selectedPrompts);
  } else {
    // 默认模板
    finalPrompt = constructDefaultPrompt(selectedPrompts);
  }

  return finalPrompt;
}
```

5. 标签过滤增强法

利用标签（tags）属性进行更精细的过滤和增强：

```
function parsePrompts(prompts: IPrompt_v3_for_promptsFactory[]): string
  const selectedPrompts = prompts.filter(p => p.status === 'selected');

  // 根据标签分组
  const promptsByTag = {};
  selectedPrompts.forEach(p => {
    if (p.tags) {
      p.tags.forEach(tag => {
        if (!promptsByTag[tag]) promptsByTag[tag] = [];
        promptsByTag[tag].push(p);
      });
    }
  });
```

```
  // 根据标签组合构建特定的提示词部分
  // ...

  return finalPrompt;
}
```

6. 历史感知递归解析法

处理带有历史记录的提示，递归解析历史内容：

```
function parsePrompts(prompts: IPrompt_v3_for_promptsFactory[], isRecur
  const selectedPrompts = prompts.filter(p => p.status === 'selected');
  let finalPrompt = '';

  for (const prompt of selectedPrompts) {
    finalPrompt += `[${prompt.type}]: ${prompt.title}\n`;

    // 递归处理历史记录
    if (prompt.history && prompt.history.length > 0) {
      finalPrompt += `\nHistory:\n${parsePrompts(prompt.history, true)}\
    }
  }

  return finalPrompt;
}
```

7. 模型特定优化法

根据目标大模型的特性定制提示词格式：

```
function parsePrompts(prompts: IPrompt_v3_for_promptsFactory[], modelTy
  const selectedPrompts = prompts.filter(p => p.status === 'selected');

  switch (modelType) {
```

```
    case 'gpt4':
      return formatForGPT4(selectedPrompts);
    case 'claude':
      return formatForClaude(selectedPrompts);
    case 'llama':
      return formatForLlama(selectedPrompts);
    default:
      return formatDefault(selectedPrompts);
  }
}
```

8. 交互式构建法

允许用户在构建过程中交互式地调整提示词：

```
async function buildPromptInteractively(prompts: IPrompt_v3_for_prompts
  const selectedPrompts = prompts.filter(p => p.status === 'selected');
  let finalPrompt = '';

  // 展示初步构建的提示词
  finalPrompt = basicParse(selectedPrompts);

  // 允许用户调整
  const userFeedback = await getUserFeedback(finalPrompt);

  // 根据反馈调整
  return adjustPromptBasedOnFeedback(finalPrompt, userFeedback);
}
```

这些方法可以单独使用，也可以组合使用，以创建最适合特定用例的提示词解析
